Data protection for LAMP applications

Paddy Sreenivasan (paddy@zmanda.com)

Today, LAMP has become the most common open source platform for developing web and enterprise applications. Protecting the application data is of paramount importance to system administrators. This paper discusses various open source data protection techniques for LAMP applications. The paper also provides integrated open source solutions that can be used for protecting LAMP application data.

We have seen an increase in the number of web and enterprise applications based on LAMP application stack. LAMP applications use Linux or BSD as the operating system, Apache as the web server, MySQL or PostgreSQL as the application database and PHP, Perl or Python as the programming language. JBOSS is used as the middleware for some of these applications. The common attributes of these layers are:

- They are open source and have a large developer and user community behind these projects.
- They are available on multiple hardware platforms and are very well integrated.

An often overlooked aspect in the LAMP application solution is the protection of the application and configuration data. This paper examines how to use available open source tools to protect the LAMP application data. This paper does not discuss the security aspects of the application data or securing the LAMP application servers. It is also important to test the data recovery scenarios before the actual need arises.

LAMP Application data characteristics

From a data protection perspective, one of the key characteristics of LAMP application and configuration data is that they reside on file systems or in MySQL or PostgreSQL databases. Apache configuration is usually stored under */etc/httpd/* directory. PHP configuration files are stored under */etc/php/* directory. Both MySQL and PostgreSQL data, index and other configuration files are also commonly stored in file systems. Another characteristic is that LAMP applications may be distributed on multiple servers. Distributed LAMP application servers provides more scalability and availability for the applications. Enterprise and Web applications require highest levels of availability with very little downtime available for backups.

These characteristics of LAMP applications data, (stored in databases as well as filesystems, and potentially distributed across servers), make them challenging to backup. The system administrator needs to pay attention to the data of all layers of LAMP application in order to get a consistent data backup for the whole LAMP application stack.

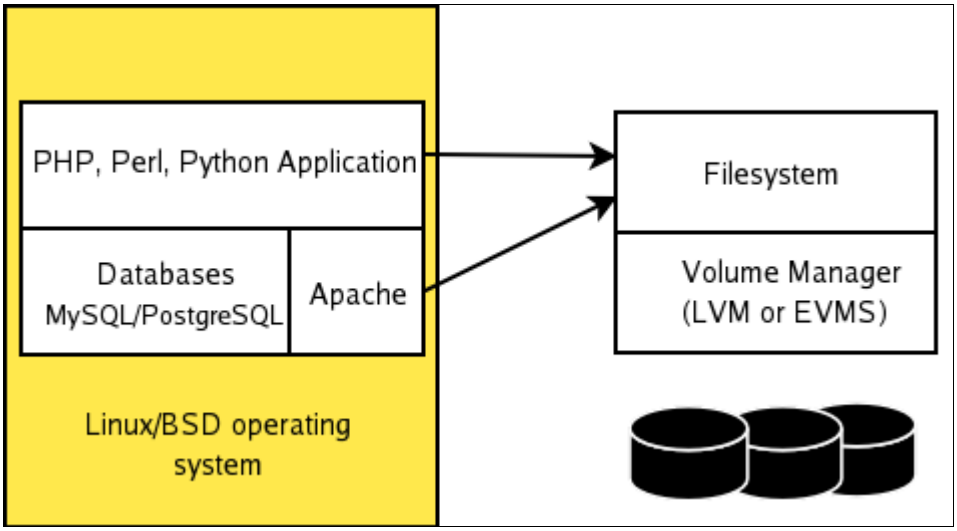

Figure: LAMP application and where the application data resides

The type of data protection solution implemented depends on the impact on application performance, application availability, type of failures to recover from and the cost of implementing the solution. One of the key requirements for backed up data is the level of data consistency required. There are two levels of data consistency:

- Application consistency: The backed up data is completely consistent i.e, the application does not need to do any internal recovery when the backed up data is restored.
- Crash consistency: The backed up data is equivalent to the data in the persistent storage when the application fails (aborts in a unexpected manner). The application (or other components such as file system) will have to do internal recovery when the backed up data is restored.

Very few applications support the capability of temporarily quiescing the application I/O at a consistent point in time so that the data can be backed up. Database servers do provide such capability.

Recovery from backed up data is done for recovering from operator errors, disk/filesystem failures, application data corruption due to bugs, or complete application host failures. The data protection solution used depends on the type of failures to recover from and how quickly the data should be recovered.

Consistent backups of MySQL

MySQL provides two tools for backing up the databases. These tools are open source and are available as part of MySQL distribution. The *mysqldump* tool creates a logical backup of the database whereas *mysqlhotcopy* tool creates a physical (or raw) backup of the database. The logical backup archive contains the MySQL statements to reconstruct the database data. Whereas, the physical backup archive contains database data, indexes and logs in raw format (database specific format). Both utilities do not maintain database referential integrity i.e, all the tables are not backed up at the same time (or same consistency point). The tool used to backup MySQL depends on the storage engines used by the tables in the database. Storage engines that support transactions such as InnoDB, BDB, Solid, MySQL's Falcon (in future) can be backed up as a transaction without any impact to the database application availability. These backups are usually referred to as "hot" or "live" backups. Storage engines that do not support transactions

such as MyISAM, ARCHIVE, MERGE have to be backed up by obtaining a global read lock on the database and flushing active database data and logs to the disk.

Logical full backups with *mysqldump*

Logical backup consists of SQL statements (CREATE TABLE, INSERT INTO) that can re-create the complete database or a table in the database. The *mysqldump* tool creates logical backups of MySQL database or tables for all the storage engines. Command line options used for doing optimal mysqldump backups are dependent on whether the underlying storage engine supports transactions or not.

mysqldump [options] db_name [tables]
mysqldump [options] --databases db_name1 [db_name2 db_name3...]
mysqldump [options] --all-databases

Common set of mysqldump options that are used for backing up databases are:

--opt and *--extended-insert* options generates optimized SQL statements. These options will speed up the restore operation and create smaller backup files. *--opt* option is necessary to create correct DDL statements (CREATE TABLE).

--lock-all-tables and *--flush-logs* options lock all tables in the database and flushes the data. All pending transactions are committed. These options are necessary for non-transactional storage engines such as MyISAM, ARCHIVE and MERGE. Obtaining read lock on all tables in an active database during the backup may not be feasible. If you are using snapshot or replication data consistency mechanisms described in later sections of this paper, it may not be necessary to lock all tables and flush logs.

--single-transaction can create consistent backups for transactional storage engines such as InnoDB and BDB. Either the *lock-all-tables* option or the *single-transaction* option should be used.

--master-data=2 writes the binary log file name and position to the backup output as an SQL statement. It can help in point-in-time recovery.

Example: *mysqldump* command options to back up all databases containing only MyISAM tables
--opt --extended-insert --lock-all-tables --flush-logs --routines --triggers --master-data=2 --all-databases

One of the advantages of the logical backup is the data can be restored to another database server. Logical backup is independent of the server architecture, filesystem type, and can be recovered on a completely different database server. It is also possible to correct operator errors (such as erroneous DROP TABLE statements) from logical backups. This is significant advantage compared to Raw backups. The *mysqldump* tool can also be run on a different machine from the MySQL server.

The disadvantage of logical backups is the backup file size can be larger than the actual database (sometimes it is twice the size of the actual database). The restoration time from the logical backup can be significant compared to Raw backups. All SQL statements in the logical backup have to be re-run during the restore process.

Raw (physical) full backups with *mysqlhotcopy*

A raw backup consists of the database data in a binary format and will be different for machine architectures. The *mysqlhotcopy* tool is a perl script that makes a copy of the data, index and data dictionary files to the local machine or to a remote machine. This tool takes advantage of the fact that database tables data, index and log files are stored in the database data directory configured by the database administrator. The *mysqlhotcopy* tool works only for storage engines that do not support transactions such as MyISAM, MERGE, ARCHIVE and will not work with storage engines that support transactions such as InnoDB because database tables may not be stored in the database data directory. The tool obtains a read lock on the tables/database being backed up and flushes the database pages to the disk before doing a backup.

mysqlhotcopy --flushlog db_name_1 ... db_name_n /path/to/new_directory

Restoring from raw backups is significantly faster than the logical backups because the SQL statements need not be executed. The raw backup size is same as the size of database data and logs. Since it is a raw backup, it is not possible to correct operator errors.

Most of the engines that support transactions provides tools that helps create raw backups. The SOLID storage engine provides its own backup tool that does binary backups. InnoDB provides a commercial tool (*ibbackup*) that creates raw backups for InnodB tables. InnoDB also provides *innobackup* script that takes care of backing of InnoDB tables, data dictionary (*.frm* files), and other MyISAM tables using *mysqldump* tool.

Incremental backups

MySQL server can maintain binary logs which contain all events (MySQL statements) that updated data or could have updated data. Binary logging must be enabled to do incremental MySQL backups. Binary logs keep track of all changes to the data in the database and can be replayed later to restore the database. Enabling database logging can reduce the database performance. The binary log location can be put in a different disk from the MySQL data directory so that the performance impact is reduced. This will also help improve availability.

Example: Binary log files with prefix "backup_logs" are created
 # mysqld --log-bin=backup_logs

Binary logs are rotated when the MySQL server is restarted. Rotate binary logs to mark when the incremental backup was done. The *mysqladmin flush-logs* command rotates the binary logs. The binary logs since the last full MySQL backup or the incremental backup can be backed up using network backup tool.

Data consistency using snapshots

The MySQL database data directory can be stored in logical volumes that are managed by the Logical Volume Manager (LVM) or the Enterprise Volume Management System (EVMS). Using logical volumes to store data and index files allows the volume to be extended to multiple physical disks, file systems can be extended using the filesystem *growfs* command. Logical volumes can be mirrored to increase database availability. Snapshot of logical volume contents to another logical volume can be done in LVM as well as EVMS. The snapshot volume is a copy-on-write

volume i.e, the data is written to the snapshot volume on when the data block on the original volume changes. No data copying is done when snapshot of logical volume is created.

To create a consistent copy of the database, a snapshot of the logical volume that contains the MySQL database data directory is created.
It is necessary to flush tables and obtain a read lock before taking a snapshot. Obtaining a read lock on an active database might take some time. Since snapshots are copy-on-write copies, a snapshot operation does not take much time. The lock can be released after the snapshot operation and is not held during the backup processes. In case of tables with transactional storage engines such as InnoDB, it is not necessary to obtain a read lock. When backed copy of the database is restored, the database will be recovered from the transaction logs. Obtaining read lock on a transactional storage engine table will commit all pending transactions and there will be a significant impact on the performance.

The following steps show how to do backups using snapshots to achieve data consistency:

Step 1: Obtain a read lock on the database and flush logs. This step is necessary if the database being backed up contains tables with non-transactional storage engines or specific tables with non-transactional storage engine have to be backed up.

mysql> FLUSH TABLES WITH READ LOCK

Step 2: Create a snapshot volume for the logical volume where the database data directory resides (*mysqlvolume* in this example) The following LVM command creates a snapshot volume "mysqlbackup" of size 50MB. The data changes during the backup cannot exceed 50 MB. Usually, a snapshot volume size of 15-20% of the original volume size is created. If the database is really active, a larger snapshot volume size might be required.

 # lvcreate -L50M -s -n mysqlbackup /dev/mysqlvolume

Step 3: Release the read lock on the database if lock was obtained on Step 1.

mysql> UNLOCK TABLES

Step 4: Mount the snapshot volume in a different directory.

 # mount /dev/mysqlbackup /backup

Step 4: Use the network based backup tool, Amanda, described in the next section, to backup the database backup files from /backup directory and application files to backup media such as disks, tapes, optical media.

Step 5: Remove the snapshot logical volume.

 # lvremove -f /dev/mysqlbackup

Using EVMS instead of LVM for logical volumes, provides additional advantages and reduces the complexity of the backup procedure. EVMS allows the snapshot volume to be expanded. This will take care of long backup windows in case the initial size of snapshot volume is small. With EVMS, a snapshot volume can be created permanently. The snapshot volume can be reinitialized

as the step 1 and the snapshot volume need not be removed at the end of backup. The snapshot volume can be used to rollback changes to the original volume. The snapshot volume acts as a data backup and the snapshot rollback is a quick restore mechanism.

Filesystem freeze and unfreeze operations should be used along with data snapshots to provide better data consistency for the database and the application files. Filesystem freeze operation is supported by file systems such as XFS, GFS and must be used in conjunction with logical volume manager snapshots to obtain a consistent view of the database and application files. Filesystem freeze operation should be done before step 2 and unfreeze operation should be done after step 2. If ext3fs filesystem is used as the filesystem for the LAMP application and the filesystem data and meta-data journaling is enabled, file system freeze and unfreeze operation is not required.

Following command freezes the XFS filesytem */var/lib/mysql* in preparation for the backup:

 # xfs_freeze -f /var/lib/mysql

Following command unfreezes the XFS filesystem:

 # xfs_freeze -u /var/lib/mysql

Data consistency using replication

MySQL allows creation of replication slaves for the master server. Multiple replication slaves can be created for a master server and a replication server can act as a master server. All the transactions that are executed on the master server are replayed on the slave server asynchronously. MySQL commands are sent to the slave server for execution. The slave server can be behind the master replication server if the slave is slow in executing the MySQL commands. This feature is available for all storage engines. This is referred to as statement-based replication.

MySQL 5.1 supports row-based replication in addition to statement-based replication. The master server creates a binary log of how the table rows are affected by each statement and the binary log is replicated to the slave. Row-based replication handles replication of stored routines and triggers, such as non-deterministic user defined functions, correctly. Statement-based replication produce smaller logs and are easier to audit. A "mixed" mode of replication is also supported in MySQL 5.1 that uses statement-based replication by default and uses row-based replication for MySQL statements that use functions that will give different results when executed on master and slave server.

One use for replication server is for backup. Replication can be used for database high-availability and load balancing. The replication slave has a consistent view of the database and backed up using any of the MySQL backup methods described in the above sections. The following steps assumes that replication slave has been set up for backup purpose.

Step 1: Stop the slave replication and check the slave status on the slave server to see if the replication has stopped

mysql> STOP SLAVE;

Step 2: Capture the master log file name and position from "slave status" command

mysql> SHOW SLAVE STATUS;

Step 3: Perform any of the above MySQL backup methods to capture the database to files.

Step 4: Add "CHANGE MASTER" MySQL statement to the backup file using the values obtained in step 2. This step marks the master log file and the position in the log file. Marking the log position makes backup restoration easier.

Step 5: Use Amanda described in the next section to backup the database backup files and application files to backup media such as disks, tapes, optical media.

Step 6: Start slave process

mysql> START SLAVE;

Using replication for backup does not take care of operator errors because such errors are also replicated. Replication solution is more expensive than snapshot solution because it needs additional MySQL host. On the other hand, if replication is being done for availability or load balancing, it can be easily used for backup purpose as well.

Consistent backups from PostgreSQL

PostgreSQL provides multi-version concurrency control. Multi-version concurrency control makes creating database consistent backups relatively easier. Database consistency using snapshots can be also performed. This would be equivalent for recovering from a PostgreSQL server crash.

Logical full backups

There are two tools - *pg_dump*, *pg_dumpall* that create logical backups of the database into a backup file. To avoid file size restrictions due to file system limits, the output of these commands can be sent to *split* command. These tools can be run on any machine (not only on the machine running PostgreSQL postmaster server).

Example: Dumping a postgres database to multiple files with names starting with "pg_backup"
 # pg_dump postgresdb | split -b 1024m - pg_backup

Raw (Physical) full backups - Continuous archiving

PostgreSQL 8.1 keeps track of all changes made to the database data files in Write Ahead Logs (WAL) under pg_xlog sub-directory of the cluster data directory. Write ahead logs can be archived in a different directory before the logs are reused by the database. These logs can be copied to a directory backed up by the network backup tools as described later in the paper. The following example shows how WAL can be archived to a different directory.

Example: Copy WAL to /backup/postgres/WAL directory. Add the following shell command

to *the postgresql.conf* file
archive_command = 'cp -i %p /backup/postgres/WAL/%f < /dev/null'

Note: It is necessary to backup configuration files such as *postgresql.conf*, *pg_hba.conf* and *pg_ident.conf* separately.

To do a full raw online backup, perform the following steps as the database super user:

Step 1: Execute the SQL command
SELECT pg_start_backup('backup_id1');

The backup information (backup label, time of command execution, name of first WAL segment file) are stored in the */backup/postgres/backup_id1* file.

Step 2: Perform backup of the file system backup of the database data directory (*/usr/local/pgsql/data*). Exclude the WAL files under *pg_xlog* sub-directory. These files are already backed up by the archive command.

Step 3: Execute the SQL command to inform that backup is complete
SELECT pg_stop_backup();

Stopping backup creates a backup history file in the WAL archive location. The starting WAL file is used as part of the backup history file. WAL archival operation will be complete after the backup stop command is executed.

All files under the WAL archival and the database data directory must be backed up using the network backup tool as described later in the paper.

Incremental backups

All automatically archived WAL logs can be used as incremental backups. Use the network backup tool to perform incremental backups of WAL archive directory.

Network based backup and recovery

After getting a consistent data copy using replication or snapshots, network based backup and recovery tools can be used to backup the data into various media such as tapes, disks, optical devices and NAS appliances. Amanda is the most popular open source backup and archiving tool. Amanda is an active sourceforge project with tens of thousands of users and hundreds of developers. This open source project has been part of public domain since 1991 and is available as part of various Linux and BSD distributions. Latest packages are available at http://www.zmanda.com/downloads.html As of writing of this paper, Amanda 2.5 is the latest and stable release. Amanda can be used to backup multiple servers running the LAMP applications in the same backup run or in different backup runs. Amanda server can run on one of the LAMP application servers or on a dedicated Amanda backup server. Amanda backup server will be utilized only during a backup run and will be idle at other times.

One of the unique features of Amanda is the consistent backup window. Amanda attempts to backup the same amount of data from the various backup clients running LAMP applications in

each backup run. Amanda will distribute the full backup for each client (actually each backup unit) over the backup cycle days to achieve same amount of backup size each run. For system administrators it provides consistent backup window for each backup run without having to tune the backup configuration.

In the following figure, three client file systems are being backed up daily by Amanda. In the first backup cycle, a full backup for all filesystems are performed by Amanda. Amanda based on the amount of backup daily media available and amount of data changes in each client file system constructs a backup plan of backup levels (full, incremental backup levels 1-9) for each backup run. Full backup is backup level 0. All file systems have at least a full backup done every backup cycle days (Amanda configuration parameter). As the figure shows, the amount of data backed up is almost the same for each run resulting in consistent backup windows.

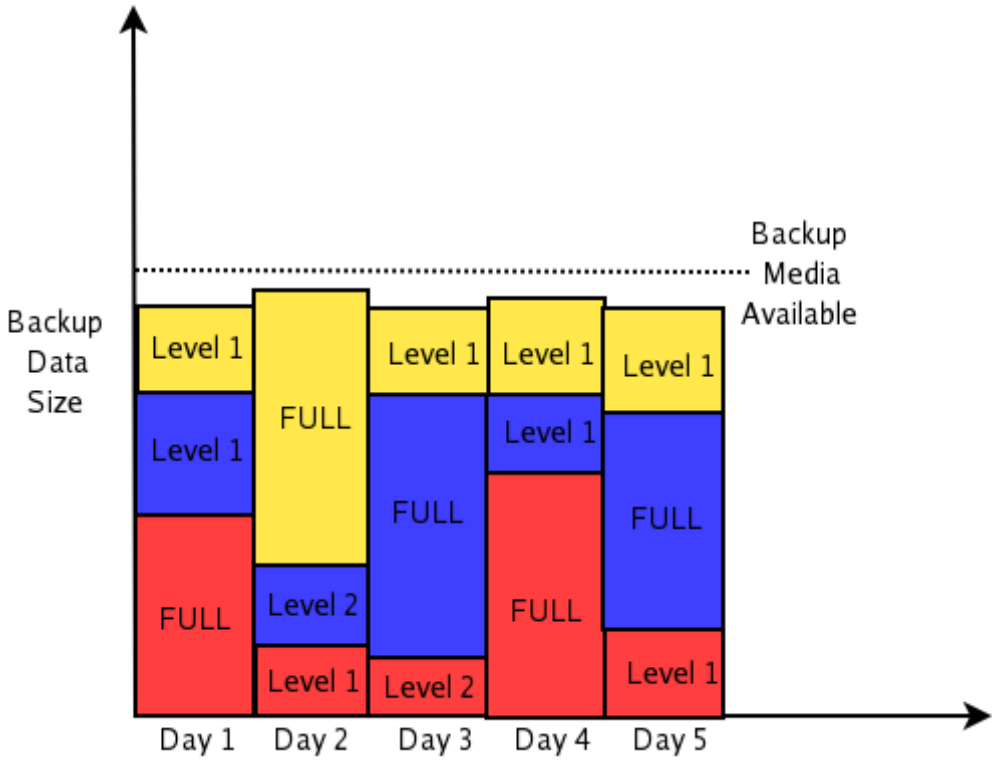

Figure : Amount of data backed up each daily run

Amanda uses platform tools for backup, data compression and encryption. It uses tools such as GNU tar, file system dump and Schily tar for backup and can use any tool that is available in the client. Amanda uses the tool's data format on the backup media. This feature allows the possibility of restoring data without using Amanda. All Amanda backup media can be read using *dd* and *mt* commands. In fact, the Amanda media file header has the command to restore the media as a text string.

Example: Using *dd* command to display Amanda tape file header

 # dd if=<amanda_media_dev> bs=32k count=1
AMANDA: FILE 20060228 natasha /boot lev 1 comp N program /bin/gtar
To restore, position tape at start of file and run:
dd if=<tape> bs=32k skip=1 | /bin/gtar -f... -

1+0 records in
1+0 records out

Amanda can backup to tapes, disks, optical media, media changers and RAIT. RAIT stands for redundant array of tapes (in fact, it works for any media volume). It is similar to RAID in concept, the backup data is striped across multiple media volumes and the parity information in another media volume. RAIT with 2 volume set is same as mirroring backup data across two media volumes. Using RAIT with a two volume set of disk and tape media, the backup can be done to disk (for immediate recovery needs) and to tape (for archival needs) simultaneously.

Amanda allows lots of configuration flexibility in specifying on how backup should be done up to a granularity of a file. Users can specify type of compression, encryption, whether they should be performed on Amanda client or server, which network to be used for backups for each backup unit. To reduce the complexity introduced by the configuration flexibility, configuration tools are available for first-time Amanda users.

Amanda supports both data consistency mechanisms for LAMP application backup - snapshots as well as replication. Amanda server is scalable and can backup hundreds of LAMP application servers in each backup run. The commands to prepare the data for consistent backups can be done as pre-backup action before Amanda backup. Since any platform commands can be used for backing up applications using Amanda, it is easy to use different data protection mechanisms for different LAMP application servers in the same Amanda configuration. Amanda developers are working on a Application API that will make addition of new backup programs for applications easier.

Some Amanda users backup LAMP application data including the MySQL databases to a temporary directory in a file system. Later, the file system directory is backed up by Amanda during the backup run. This method allows quicker recovery from the temporary filesystem and recovery from Amanda media for longer term needs.

Amanda users and developers use the Amanda wiki (http://wiki.zmanda.com/) for documenting the project as well as how they incorporate Amanda in the overall IT processes.

Recovery process

Data from the backup archive can be recovered using Amanda recovery tools - *amrecover* and *amrestore* commands. The *amrecover* tool allows users to browse the index database and allows users to choose the backups to restore from. This tool can be run on any machine where Amanda client software is installed. It is always advisable to restore backups to a temporary directory location. Application files can be restored to the correct location on the LAMP application server. MySQL and PostgreSQL database can be restored from *mysqldump* and *pg_dump* backup files respectively. Full backup restoration can be done only when the database and application process are not running.

Example: Complete restoration for database "database1" using *mysqldump* backup file "mysql_backup"
 # mysql database1 < mysql_backup

Example: Complete restoration of a Postgres database from *pg_dump* backup files with starting with "pg_backup"

 # cat pg_backup. | psql postgresdb*

To restore PostgreSQL database from the raw backup files, recreate the PostgreSQL cluster data directory from the restored files. It is advisable to make copy of the existing contents before restoring files from the backup. The *pg_xlog* sub-directory can contain WAL that were not archived. Recovery process is controlled using *recovery.conf* file in the cluster data directory. This file contains the location where archived WAL logs can be found.

Example: recovery.conf file showing where the archived WAL logs will be copied from
restore_command = 'cp /backup/postgres/WAL/%f %p'

Incremental restores

Incremental restores can be done from the MySQL binary log files restored to the temporary directory by Amanda network backup tool. The incremental restore can be done from a start time to an end time or starting log position to ending log position in the binary logs. The log position can be used to fix operator errors using the backup files.

Example: Incremental restore of all MySQL database changes done till May 1, 2006
 # mysqlbinlog --stop-date="2006-05-01 12:00:00" backup-logs.[0-9] | mysql -u <user name> -p <password>*

PostgreSQL incremental restores can be done using settings in *recovery.conf* in the cluster data directory. It is possible to specify end timestamps for recovery or end transaction id for recovery. When the postmaster process is restarted and the *restore_command* is used to retrieve the archived WAL segments up to the specified stop point either the date/time stamp or the specific transaction id.

It is critical to test the data recovery procedure for LAMP application on a regular basis.

Backup security

Security is important for any data protection process. The data must be stored in the backup media in a secure manner using encryption. The keys used for encryption should be tracked and kept in a secure database that is also backed up. Amanda's flexibility in encrypting data on the backup client where LAMP applications are running or on the Amanda server provides more options in securing the data. Secure communication should be used during the backup using tools such as OpenSSH or network tunneling.

MySQL and PostgreSQL database backup and restore users should be created with minimum privileges required to backup and restore the database. For Example: Minimum privileges for backup user for *mysqlhotcopy* and *mysqldump* tools are SELECT, RELOAD, LOCK TABLES. The database user and password must be stored in a secure manner.

Conclusion

This paper provides a window into protecting the critical data stored in LAMP applications using open source tools and examples of putting the tools together to create a complete solutions.

Several open source data protection tools are available for LAMP applications. These tools can be used to create secure backups for LAMP applications. Increased use of LAMP technologies for web applications, data warehousing applications and enterprise applications make it necessary to protect the customer data used by these applications.

The many new and exciting open source developments in the data protection field, such as creating backups while maintaining database referential integrity and using storage available over internet as backup media, allow creation of just the right LAMP backup solution for an individual sites' needs.

Links

- MySQL backup tools: http://dev.mysql.com/doc/refman/5.0/en/disaster-prevention.html
- MySQL backup forums: http://forums.mysql.com/list.php?28
- PostgreSQL backup tools: http://www.postgresql.org/docs/8.1/static/backup-online.html
- Amanda wiki: http://wiki.zmanda.org/
- Amanda forums: http://forums.zmanda.org/