Table of Contents

Introduction

There are several ways to build a cluster for high availability. Also there are several technologies involved on the cluster design.

For this reason, the technology selection will be based on the available resources for the selected operating system (i.e. RHEL), and the simplification of the OpenNMS maintenance.

This cluster is going to work on a active-passive mode, that means, only one machine will be running at a time, and when this machine cannot perform its job, the cluster manager will redirect the resources to the standby machine.

For *RHEL/CentOS 6*, the default cluster services are based on *CMAN/Ricci/Rgmanager*. On the other hand, for *RHEL/CentOS 7* the cluster services are based on *Pacemaker* (which is available as an option on *RHEL/CentOS 6.6*). Both technologies use *Corosync* behind the scenes.

The common elements for *OpenNMS* are the configuration directory, the data directory and the *PostgreSQL* database. The common directories are going to exist on a separate server and shared through *NFS* (it can be another technology such as *DRBD* or *GFS2*). The database will be deployed on a separate set of servers.

The need for a redundant, high(er) available database-setup is important. When you're using *PostgreSQL*, you can setup streaming replication quite easily in order to have your data redundant on two or more nodes.

To achieve redundancy, I've chosen to use the built-in streaming replication of *PostgreSQL*. Another tool, `repmgr`, will be used for easier control of the replication between those redundant *PostgreSQL-instances*. The tool doesn't replicate data itself but it allows you to easily control the replication, the standby-server(s) and monitor the status of the whole replication process.

More information about repmgr.

In order to use both copies of the database-data, I will use *pgpool-II*. *Pgpool-II* can pool the connections to the nodes and will monitor their status and trigger failover (by repmgr) if needed. *Pgpool-II* has the capability to load balance traffic, based on the type of SQL-query. For example, a *SELECT query* can perfectly be executed on a slave (read-only) node and save resources on the master (read-write) node.

From the client perspectives, they will see only one server (i.e the instance on which *pgpool-II* is running), and the failover, the standby recovery, etc. will be hidden for the users; in this case, the *OpenNMS* servers.

More information about pgpool-II.