openNMS

an introduction

What is OpenNMS

OpenNMS is an enterprise ready Network Management System written in Java that runs as an Apache Tomcat application.

- Configuration via XML
- Automated network discovery
- Capability discovery and polling for most common services
  - Oracle, SNMP, ICMP, HTTP, PostgreSQL, Informix, Sybase, SMTP, IMAP, Citrix, DHCP, M$-SQL, HTTPS, DominoIIOP, DNS, SSH, LDAP, POP3, FTP, MySQL
  - Additional services can usually be discovered and monitored with only a little additional configuration.
- Multi-user, including support of shift schedules.
- Downtime model and scheduled outage support
- Event logging and notification, including notice escalation
- SNMP Performance monitoring
- Flexible Reporting

Requirements

- Java Virtual Machine
  - Version 1.4 or greater
- Perl 5
  - DBD & DBD:Pg
- Fast Processor
  - 300MHz or greater
- Available RAM
  - Minimum of 192Mb
  - 256Mb minimum recommended
- Available Disk Space
  - Installation is ~25Mb
  - 8Mb per managed device
  - ~25Mb in /tmp
  - Ample space for log files
    - These get very large very fast, use logrotated
- RedHat Linux is recommended

```
perl -MCPAN -e shell
install DBI
install DBD::Pg
```

Advice:
Get a good XML editor, don't try and manage the XML configuration files with your old text editor. I recommend:
http://www.nongnu.org/mlview/

Configuring PostgreSQL

OpenNMS requires PostgreSQL 7.1.x or greater. If PostgreSQL is not already installed the OpenNMS download site provides packages of a version known to work with OpenNMS.

You also should have the complete set of PostgreSQL compantion packages installed including JDBC and Perl support.

postgresql.conf
```
tcpip_socket = true
max_connections = 256
shared_buffers = 1024
```

OpenNMS' extremely threaded nature results in many conncurrent connections, and to maintain integrity exploits PostrgreSQL's full RDMS feature support. This may require the default parameters provided with most PostgreSQL packages to be adjusted.

A band of daemons joined in...

OpenNMS is implemented as a collection of daemon processes.

Each daemon handles a specific task and may spawn multiple working threads, this allows OpenNMS to manage large networks which may have many silumtaneous events occuring.

The deamons (and threads) communicate via a virtual bus to which they may submit “events” and on which they listen for the type of events that concern them.

```
OpenNMS.Eventd      : running
OpenNMS.Trapd       : running
OpenNMS.Dhcpd       : running
OpenNMS.Actiond     : running
OpenNMS.Capsd       : running
OpenNMS.Notifd      : running
OpenNMS.Outaged     : running
OpenNMS.Rtcd        : running
OpenNMS.Pollerd     : running
OpenNMS.Collectd    : running
OpenNMS.Threshd     : running
OpenNMS.Discovery   : running
```

Discovery

- The first process that OpenNMS is referred to as “discovery”
  - During discovery ICMP packets are used to locate hosts and devices on the network.
    - The parameters for discovery can be adjusted based upon the power of your OpenNMS and the size and latency of your network.
      - threads (default: 1), packets-per-second (default: 1), initial-sleep-time (default: 300000, 5 minutes), restart-sleep-time (default: 86400000, 24 hours), timeout (default: 800), and retries (default: 3)
    - The range of addresses scanned must be configured before OpenNMS is started.
  - If discovery locates a responsive address it places a NewSuspect event on the bus for processing by the other deamons.
  - The status of the discovery process is recorded in discovery.log

Discovery Control

/opt/OpenNMS/etc/discovery-configuration.xml

```
<discovery-configuration threads="1" packets-per-second="1"
        initial-sleep-time="300000" restart-sleep-time="86400000"
        retries="3" timeout="800">
        <include-range retries="2" timeout="3000">
                <begin>192.168.0.1</begin>
                <end>192.168.0.254</end>
        </include-range>
        <include-url>file:/opt/OpenNMS/etc/include</include-url>
</discovery-configuration>
```

```
192.168.26.1
192.168.21.1
192.168.10.1
192.168.10.56
192.168.10.18
```

The first task in setting up OpenNMS itself is to define what addresses should be included in the discovery process.

Capabilities

- Capabilities are services available from a device or host
  - httpd, DNS, DHCP, PostgreSQL, etc....
- The capsd daemon lists on OpenNMS's virtual bus for NewSuspect events and queues appropriate addresses for investigation.
  - The process of capability scanning can be tuned.
    - rescan-frequency (default: 24 hours), initial-sleep-time (default: 5 minutes), management-policy (managed or unmanaged), max-suspect-thread-pool-size (default:6), max-rescan-thread-pool-size (default: 3), abort-protocol-scans-if-no-route (default: false)
  - Various protocols are supported via plugins defined in the capsd daemon's configuration so that additional protocols may be easily defined.
  - Services are then attached to protocols, possibly with additional paramaters.
    - For example: A server speaking HTTP protocol on port 80 is an HTTP server, while a server speaking HTTP protocol on port 3128 may be a web cache server.

capsd Configuration

/opt/OpenNMS/etc/capsd-configuration.xml:

```
<capsd-configuration rescan-frequency="86400000"
        initial-sleep-time="300000"
        management-policy="managed"
        max-suspect-thread-pool-size = "6"
        max-rescan-thread-pool-size = "3"
        abort-protocol-scans-if-no-route = "false">
        <protocol-plugin protocol="ICMP"
              class-name="org.opennms.netmgt.capsd.IcmpPlugin"
              scan="on" user-defined="false">
                <property key="timeout" value="2000"/>
                <property key="retry" value="2"/>
        </protocol-plugin>
. . .
```

capsd Configuration

Capability scanning CIFS/SMB services requires valid credentials.

/opt/OpenNMS/etc/capsd-configuration.xml: (continued)

```
. . .
    <smb-config>
        <smb-auth user="pcnet" password="pcnet" type="domain">BACKBONE</smb-auth>
    </smb-config>
    <ip-management policy="managed">
        <range begin="192.0.0.0" end="192.255.255.255"/>
        <include-url>file:/opt/OpenNMS/etc/include</include-url>
    </ip-management>
    <ip-management policy="unmanaged">
        <specific>0.0.0.0</specific>
        <range begin="127.0.0.0" end="127.255.255.255"/>
        <range begin="10.0.0.0" end="10.255.255.255"/>
    </ip-management>
</capsd-configuration>
```

Capability scans can be disabled based upon address ranges.

Actually, the managed policy is assumed, it is defined here only for clarity.

Adding A Capability: Squid

PostgreSQL opennms database:

```
insert into service(serviceid,servicename) values (28,'Squid');
```

/opt/OpenNMS/etc/capsd-configuration.xml:

```
. . .
        <protocol-plugin protocol="Squid"
              class-name="org.opennms.netmgt.capsd.HttpPlugin"
              scan="on" user-defined="true">
                <property key="ports" value="3128"/>
                <property key="timeout" value="30000"/>
                <property key="retry" value="2"/>
        </protocol-plugin>
. . .
```

The services table defines available servies in tandem with the capsd configuration file and also associates each service with a integer primary key for use in the database.

The Capacity Scan

The progress of capability scans can be monitored via the log file.

```
2002-08-01 12:12:09,797 DEBUG [Capsd Rescan Pool-fiber2] IfCollector: 192.168.1.5
testing plugin Squid
2002-08-01 12:12:09,798 DEBUG [Capsd Rescan Pool-fiber2] HttpPlugin:
org.opennms.netmgt.capsd.HttpPlugin.isServer: attempt 0 to connect 192.168.1.5:3128,
timeout=30000
2002-08-01 12:12:09,808 DEBUG [Capsd Rescan Pool-fiber2] IfCollector: 192.168.1.5
protocol Squid supported? true
2002-08-01 12:12:09,809 DEBUG [Capsd Rescan Pool-fiber2] IfCollector: 192.168.1.5
plugin Squid completed!
```

SNMP Capability

```
<protocol-plugin protocol=”SNMP” class-name=”org.opennms.netmgt.capsd.SnmpPlugin”
scan=”on” user-defined=”false”>
  <property key=”force version” value=”SNMPv1”/>
  <property key=”timeout” value=”2000”/>
  <property key=”retry” value=”3”/>
</protocol-plugin>
```

- If a device or host supports SNMP OpenNMS will proceed with gather performance data.
  - The capabilities test uses SNMPv1 (see above) as both SNMPv1 and SNMPv2 provides will respond to SNMPv1 queries.
    - This DOES NOT EFFECT the method used to collect SNMP data; SNMPv2 will be used with SNMPv2 capable provides, including support of GET-BULK commands.
  - Management of SNMP is also controlled via the snmp-config.xml configuration file.
    - Some SNMP providers can produce false positive; they appear to be willing to provide information to the OpenNMS host when they actually are not.

SNMP Capability

```
<snmp-config retry=”3” timeout=”800” read-community=”public” write-
community=”private”>
  <definition version=”v2c”>
    <specification>192.168.0.5</specific>
  </definition>
  <definition retry=”4” timeout=”2000”>
    <range begin=”192.168.1.1” end=”192.168.1.254”/>
    <range begin=”192.168.3.1” end=”192.168.3.254”/>
  </definition>
  <definition read-community=”bubba” write-community=”zeke”>
    <range begin=”192.168.2.1” write=”192.168.2.254”>
  </definition>
  <definition port=”1161”>
    <specific>192.168.5.50</specific>
  </definition>
</snmp-config>
```

Version 1 is “v1”, while version two is “v2c”.

Polling

- Polling is how OpenNMS provides status information for the discovered services on a host or devices.
  - A poll of a service starts with an attempt to establish communication with that service, and if it failes a given number of retries seperated by some interval.
    - retry, timeout
  - A poll itself is repeated after some interval.
    - interval
  - When a service does not respond a NodeLostService event is generated.
    - If all the services on an interface are lost a InterfaceDown event is generated instead of the NodeLostService events.
      - If all the interfaces on a node down a NodeDown event is generated.
        - If note-outage status=”on” in the poller configuration then NodeDown suppresses NodeLostService and InterfaceDown events.
          - Once a node is down only the critical-service is polled (by default ICMP) until that service is restored, then polling of all services is resumed.

Advanced Polling

- Polling is broken into “packages”.
  - If you create no poller packages you are using the default package “example1”.
  - Packages may include diffrent services to monitor, intervals to montior, outage calendars, downtime models, etc...
    - An outage calendar is a predefined period where the services on a particular interface are anticipated to be down, and OpenNMS should manage notifications accordingly.

      -
        ```
        <outage name=”maintenance” type=”monthly”>
          <time day=”1” begins=”23:30:00” ends=”23:45:00”/>
          <time day=”15” begins=”23:30:00” ends=”23:45:00”/>
          <time day=”30” begins=”23:30:00” ends=”23:45:00”/>
          <interface address=”192.168.1.12”/>
          <interface address=”192.168.1.5/>
        </outage>
        ```

There is support for weekly, monthly, and specific (date) outage calendars.

    - A downtime model determines how OpenNMS manages the frequency of polling when a service goes down; often polling rapidly at first expecting return of service and polling less frequency as time goes on.

Poller Packages

/opt/OpenNMS/etc/poller-configuration.xml

```
<poller-configuration threads="30" serviceUnresponsiveEnabled="false">
    <node-outage status="on"
            pollAllIfNoCriticalServiceDefined="true">
        <critical-service name="ICMP"/>
    </node-outage>
    <package name="example1">
        <filter>IPADDR IPLIKE *.*.*.*</filter>
        <specific>0.0.0.0</specific>
        <include-range begin="0.0.0.0" end="255.255.255.255"/>
        <include-url>file:/opt/OpenNMS/etc/include</include-url>
        <service name="DominoIIOP" interval="300000" user-defined="false" status="on">
            <parameter key="retry" value="3"/>
            <parameter key="timeout" value="3000"/>
            <parameter key="ior-port" value="80"/>
            <parameter key="port" value="63148"/>
        </service>
        . . .
    </package>
    <monitor service="DominoIIOP"   class-name="org.opennms.netmgt.poller.DominoIIOPMonitor"/>
    . . .  </poller-configuration>
```

Poller Package Name

Applies To . . .

Polled service definition

More Service Definitions

Adding A Polled Service: Squid

/opt/OpenNMS/etc/poller-configuration.xml

```
<poller-configuration threads="30" serviceUnresponsiveEnabled="false">
  <node-outage status="on" pollAllIfNoCriticalServiceDefined="true">
    <critical-service name="ICMP"/>
  </node-outage>
  <package name="example1">
      . . .
+         <service name="Squid" interval="300000" user-defined="true" status="on">
+               <parameter key="timeout" value="3000"/>
+               <parameter key="retry" value="3"/>
+               <parameter key="ports" value="3128"/>
+               <parameter key="url" value="/"/>
+         </service>
     . . .
    </package>
    . . .
+  <monitor service="Squid"       class-name="org.opennms.netmgt.poller.HttpMonitor"/>
      . . .
</poller-configuration>
```

How often to poll
300000ms = 5 minutes

How long without the response constitutes a failure.

Failure this number of times consitures a failure

Downtime Models

```
<downtime interval=”30000” begin=”0” end=”300000”/>
<downtime interval=”300000” begin=”300000” end=”43200000”/>
<downtime interval=”600000” begin=”43200000” end=”432000000”/>
<downtime begin=”432000000” delete=”true”/>
```

A poller packages downtime model determines the frequency for which it polls services with outages, the above downtime model results in the following behaviour -

- Outage begins until five minutes later.
  - Poll every 30 seconds
- After five minutes until twelve hours
  - Poll every five minutes
- After twelve hours until five days
  - Poll every ten minutes
- After five days mark the service as unmanaged and stop polling.

Users

users.xml

```
<ns37:user>
  <ns37:user-id>adam</ns37:user-id>
  <ns37:full-name>Adam Tauno Williams</ns37:full-name>
  <ns37:user-comments></ns37:user-comments>
  <ns37:password>6407D7E05568D44BFB0886E52E171B28</ns37:password>
  <ns37:contact type="email" info="adam@morrison-ind.com"/>
  <ns37:contact type="pagerEmail" info="9999999999@vtext.com"/>
  <ns37:contact type="numericPage" info="" serviceProvider=""/>
  <ns37:contact type="textPage" info="" serviceProvider=""/>
  <ns37:duty-schedule>MoTuWeThFrSaSu0-2359</ns37:duty-schedule>
</ns37:user>
```

groups.xml

```
<ns46:group>
  <ns46:name>Admin</ns46:name>
  <ns46:comments>The administrators</ns46:comments>
  <ns46:user>admin</ns46:user>
  <ns46:user>adam</ns46:user>
</ns46:group
```

OpenNMS supports the common concept of users & groups, information provided here is used later for delivery of notices.

Notifiers

- OpenNMS can send notifications of events to users via a variety of means
  - Numeric Pagers
  - Mobile Phones
  - User Defined
    - WinPop
    - SMS

```
<command>
  <name>email</name>
  <execute>/bin/mail</execute>
  <comment>Sending via email</comment>
  <argument streamed="false">
    <substitution>-s</substitution>
      <switch>-subject</switch>
   </argument>
  <argument streamed="false">
     <switch>-email</switch>
   </argument>
  <argument streamed="true">
     <switch>-tm</switch>
  </argument>
</command>
```

```
<command>
   <name>windowsPopup</name>
   <execute>/usr/bin/smbclient</execute>
   <comment>Sending via WinPop</comment>
   <argument streamed="false">
     <substitution>-U OpenNMS</substitution>
   </argument>
   <argument streamed="false">
     <substitution>-M</substitution>
     <switch>-pemail</switch>
   </argument>
   <argument streamed="true">
     <switch>-tm</switch>
   </argument>
</command>
```

Notifications

```
<ns16:notification name="Node sagbcm1 DNS lost" status="on">
  <ns16:uei>http://uei.opennms.org/products/bluebird/nodes/nodeLostService</ns16:uei>
  <ns16:description>Node sagbcm1 DNS lost</ns16:description>
  <ns16:rule>IPADDR IPLIKE 192.168.19.18&amp;  (isDNS )</ns16:rule>
  <ns16:destinationPath>Email-CIS</ns16:destinationPath>
  <ns16:text-message>Node sagbcm1 lost DNS service
  Time: %time%</ns16:text-message>
  <ns16:subject>Notice #%noticeid%</ns16:subject>
</ns16:notification>
```

- OpenNMS can generate notifications from events.
  - Every notice generated has a unique id.
  - Notification generation rules can be created from the Web UI for simple rules.
  - Notices have body (message) and subject very much like an E-mail message.

Starting OpenNMS

PostgreSQL must already be running, and receiving connections via TCP/IP.

```
$ service tomcat4 start
$ service opennms start
. . .
$ service opennms status
OpenNMS.Poller                          : running
OpenNMS.Eventd                          : running
OpenNMS.OutageManager                   : running
OpenNMS.Discovery                       : running
OpenNMS.Actiond                         : running
OpenNMS.Capsd                           : running
OpenNMS.Dhcpd                           : running
OpenNMS.Notifd                          : running
OpenNMS.RTCViewCategoryManager          : running
OpenNMS.Trapd                           : running
```

TIPS

Send newSuspect

A Perl script is provided for generating newSuspect messages. This allows for automated or scripted processes to notify OpenNMS that it needs to inspect a given host.

```
/opt/OpenNMS/bin/send-event.pl --interface ip-address \
  http://uei.opennms.org/products/bluebird/internal/discovery/newSuspect
```

This script requires the Perl module Getopt::Mixed

Cleaning up XML

OpenNMS uses Castor for XML parsing and reading and writing the files to disk. One result of this is that XML files modified via the web interface are rewritten to disk as one long line - hardly friendly to editing/viewing with a tool like vi.

OpenNMS provides a Perl utility to reformat XML files with nice indention, etc...

```
/opt/OpenNMS/bin/xml.reader.pl -w /opt/OpenNMS/etc/capsd-configuration.xml
```

This will replace (-w) the XML file capsd-configuration.xml with a nicely formatted version of itself.

For more information on Castor see -

http://castor.exolab.org/

No SNMP Data For Linux Server(s)

The UCD SNMP packages that ship with most Linux distributions, by default, make available the system tree to the public community.

This causes OpenNMS to correctly detect the presence of the SNMP service on that host, but for no performance data to be collected, confusing many initial users.

Add the following line to /etc/snmp/snmpd.conf -

```
view all  included .1    80
```

and service restart snmpd. Performance data should now begin to be collected.

Of course, you just openened your system up to SNMP queiries, potentially to the whole world, so go back and do some reading.