NAME

nsnotifyd — handle DNS NOTIFY messages by running a command

SYNOPSIS

nsnotifyd [**−46dVw**] [**−l** *facility*] [**−P** *pidfile*] [**−u** *user*] [**−R** *min*:*max*] [**−r** *min*:*max*] [**−s** *authority*] [**−a** *addr*] [**−p** *port*] ⟨*command*⟩ ⟨*zone*⟩ ...

DESCRIPTION

The **nsnotifyd** daemon monitors a set of DNS *zone*s and runs a *command* when any of them change. It listens for DNS NOTIFY messages so it can respond to changes promptly. It also uses each zone's SOA refresh and retry parameters to poll for updates if **nsnotifyd** does not receive NOTIFY messages more frequently.

You should specify zone names without the trailing dot. The root zone can be specified as '`.`' or `root`.

Note: **nsnotify** (without '`d`') is a client for sending DNS NOTIFY messages whereas **nsnotifyd** (with '`d`') is a daemon for handling DNS NOTIFY messages.

OPTIONS

−4 Use IPv4 only (apart the system resolver).

−6 Use IPv6 only (apart the system resolver).

−a *address*
Listen on *address* for NOTIFY messages. The default is `127.0.0.1`.

You can specify an IP address or hostname. A hostname is looked up using the system resolver. If it resolves to multiple addresses then one arbitrary address is chosen, constrained by the **−4** or **−6** options.

−d Debugging mode.

Use once to prevent **nsnotifyd** from daemonizing and to make it print log messages to stderr.

Use twice to get dumps of DNS packets.

−l *facility*
Set the `syslog`(3) facility. The default is **daemon**.

−P *path*
Write the **nsnotifyd** PID to the given *path* after daemonizing and before dropping privilege.

−p *port*
Listen on *port*, which may be a service name or a UDP port number. The default is the **domain** service, UDP port 53.

−R *interval*
Override SOA **refresh** interval.

−R *min*:*max*
Restrict SOA **refresh** intervals to be between *min* and *max*.

−r *interval*
Override SOA **retry** interval.

−r *min*:*max*
Restrict SOA **retry** intervals to be between *min* and *max*.

−s *authority*
: Specify an authoritative server to use for zone SOA refresh queries. By default **nsnotifyd** does periodic refreshes using the system recursive resolver, so its refresh queries may get stale cached answers.

  You can specify an IP address or hostname. A hostname is looked up using the system resolver, constrained by the **−4** or **−6** options.

−u *user*
: Drop privilege to *user* after daemonizing.

−V Print details about this version of **nsnotifyd**.

−w Accept NOTIFY messages for unknown zones that are not given on the command line. (Wildcard mode.)

Interval syntax

Time parameters for the **−R** and **−r** options are in seconds, or you can use a combination of the following time units, as in DNS master files. For example, `1h1m1s` is 3661 seconds.

w	weeks
d	days
h	hours
m	minutes
s	seconds

DETAILS

Startup

Before daemonizing, **nsnotifyd** makes SOA queries for each *zone* to initialize its refresh and retry timers.

Daemonizing is configured using the **−P** *pidfile* and **−u** *user* options, or disabled with the **−d** debugging option.

When daemonizing, **nsnotifyd** does *not* change its working directory. This allows the *command* to be context-sensitive.

Server

The **nsnotifyd** daemon acts as a very simple UDP-only DNS server. The only DNS queries handled by **nsnotifyd** are NOTIFY messages. It rejects other queries with a **REFUSED** response code, or **FORMERR** if the query is too mangled.

Normally **nsnotifyd** only accepts NOTIFY messages for *zones* given on the command line. NOTIFY messages are accepted for unknown zones if you use the **−w** wildcard option.

Messages are logged via `syslog`(3).

Zone refresh

When **nsnotifyd** receives a NOTIFY, or when a refresh or retry timer expires, it makes a SOA query to see if the zone has changed. The SOA query is sent to the source of the NOTIFY or, if a timer expired, to the server given in the **−s** option.

If the NOTIFY message was accepted for an unknown zone because you used the **−w** wildcard option, **nsnotifyd** makes a SOA query to verify the zone exists and to get its serial number, and runs the command if it succeeds. (It is unable to verify the zone has changed in this case.)

Some jitter is applied to SOA refresh and retry timers, so polling can occur up to 10% earlier than specified.

Command invocation

When the SOA reply indicates the zone's serial number has increased, **nsnotifyd** runs the *command* with two or three arguments:

1. the *zone* name without the trailing dot, except for the root zone '`.`';
2. its new serial number;
3. the source address of the NOTIFY, or no third argument if the update was found via a periodic refresh or retry.

When the command exits successfully, **nsnotifyd** updates its copy of the zone's SOA parameters. It will next poll the zone on its refresh interval.

If the SOA query or command fails, **nsnotifyd** does not update its SOA parameters, and and will next poll the zone on its retry interval.

Unknown zones that were not mentioned on the command line are not polled.

EXAMPLE - metazones

Metazones allow you to use standard DNS mechanisms - AXFR, IXFR, NOTIFY, UPDATE - to control the configuration of multiple name servers, instead of using a separate out-of-band distribution system.

For details, see the `metazone`(1) manual.

EXAMPLE - zone revision history

Say you have a zone, **example.org**, which is updated dynamically, and you want to automatically record its history in a `git`(1) repository.

Setup git

On a server that is authoritative for **example.org**, run the following commands:

```
$ mkdir zone-history
$ cd zone-history
$ git init
$ touch example.org
$ git add example.org
$ git commit -m 'add example.org (empty)'
```

Monitor the zone

The **nsnotify2git** script is designed to work with **nsnotifyd** to record the history of a set of zones. Continuing the transcript,

```
$ nsnotifyd -P nsnotifyd.pid -p 5533 nsnotify2git example.org
```

Send notifies

To configure BIND to send notifies to **nsnotifyd**, so it detects changes more efficiently, look in your `named.conf`(5) file for

```
zone example.org {
    ...
};
```

Inside the zone clause, add or modify the `also-notify` setting so it includes the address and port used by **nsnotifyd**, like

```
also-notify { 127.0.0.1 port 5533; };
```

Update the zone

Now, when the zone changes, **nsnotifyd** will quickly record the change in your `git` repository.

```
$ nsupdate -l
> add example.com 3600 IN TXT "foo"
> send
> quit
$ git log --format=%s
example.org IN SOA 1234
add example.org (empty)
```

EXAMPLE - stealth secondary synchronization

A stealth secondary is a server which transfers authoritative copies of a zone, but which is not listed in the zone's NS records. It will not normally get NOTIFY messages to tell it when to update the zone, so must rely on the zone's SOA timers instead.

We would like stealth secondaries to get updates promptly, but without extra manual configuration of `also-notify` lists.

To do this, **nsnotifyd** includes **nsnotify-liststealth** which analyzes a BIND log file to extract lists of AXFR and IXFR clients for each zone (excluding clients that use TSIG), and **nsnotify** which takes zone and a list of clients that should be notified. The **nsnotify2stealth** script bridges between **nsnotifyd** and these two helpers.

Create working directory

The working directory contains the client lists, one per zone, and a symlink to the log file used by BIND. You only need to run this command once when creating the directory.

```
$ mkdir notify-stealth
$ cd notify-stealth
$ ln -s /var/log/messages .log
```

This directory will also contain a `.pid` file for **nsnotifyd**, and occasionally a `.once` file to stop **nsnotify2stealth** from running more than one **nsnotify-liststealth** at a time.

Pre-populate the directory

This gets us a file per zone, each containing a list of clients for that zone. The **nsnotify2stealth** script will automatically update the client lists once per day.

```
$ nsnotify-liststealth .log
```

Monitor the zones

Because we have a file per zone, we can invoke **nsnotifyd** with a glob instead of listing the zones explicitly. The special files (`.log .once .pid`) are dotted so that the glob works as expected.

```
$ nsnotifyd -P .pid -p 5533 nsnotify2stealth *
```

Send notifies

You will also need to reconfigure BIND to send notifies to **nsnotifyd**, as described in the previous example.

Tune BIND

If you have a lot of stealth secondaries, **nsnotify2stealth** can cause a large flood of zone transfers. You may need to change BIND's capacity settings as described in the ISC Knowledge Base article cited in the **SEE ALSO** section below.

EXAMPLE - bump-in-the-wire DNSSEC

The nsdiff(1) utility creates an nsupdate(1) script from the differences between two versions of a zone. It can be used as an alternative to BIND's **inline-signing** option, amongst other things.

You can use **nsnotifyd** together with **nsdiff** to implement a zone signer that operates as a "bump in the wire" between a DNSSEC-unaware hidden master server and the zone's public name servers.

Configure your hidden master server to send notifies and allow zone transfers to your signing server:

```
also-notify { signer port 5533; };
allow-transfer { signer; };
```

Configure the signer with dynamic signed master zones, and generate keys for them:

```
zone example.org {
    type master;
    update-policy local;
    auto-dnssec maintain;
};
```

```
$ dnssec-keygen -fk example.org
$ dnssec-keygen example.org
```

Run **nsnotifyd** on the signer to trigger an update of the signed zone as soon as an update occurs on the hidden master:

```
$ nsnotifyd -P nsnotifyd.pid -p 5533 nsnotify2update example.org
```

Configure your public name servers to transfer your zones from the signer instead of from the hidden master.

BUGS

The **nsnotifyd** daemon is not very secure.

It accepts any well-formed NOTIFY message, regardless of the source. It does not support TSIG authentication (RFC 2845) for access control.

The **nsnotifyd** daemon only handles one query at a time, which prevents it from becoming a fork bomb. However, you can easily overwhelm it with more notifications than it can handle. A spoofed NOTIFY will make **nsnotifyd** send a SOA query to the spoofed source address and wait for a reply (which will probably not arrive), during which time it is unresponsive.

You should configure **nsnotifyd** to listen on a loopback address (which is the default) or use a packet filter to block unwanted traffic.

The **nsnotifyd** daemon is not aware of the authoritative servers for a zone, so it cannot filter spurious NOTIFY messages. It has a very simplistic mechanism for choosing which servers to query when refreshing a zone.

The **nsnotifyd** daemon cannot accept NOTIFY messages over TCP (RFC 5966). It does not support EDNS (RFC 6891). However, NOTIFY messages and responses are very small, so following these specifications should not be necessary in practice.

SEE ALSO

`git`(1), `metazone`(1), `named`(8), `named.conf`(5), `nsdiff`(1), `nsnotify`(1), `nspatch`(1), `nsupdate`(1), `syslog`(3).

Cathy Almond, "Tuning BIND for zone transfers", *Internet Systems Consortium, ISC Knowledge Base*, AA-00726, https://kb.isc.org/article/AA-00726.

STANDARDS

Paul Mockapetris, *Domain names - concepts and facilities*, RFC 1034, November 1987.

Paul Mockapetris, *Domain names - implementation and specification*, RFC 1035, November 1987.

Robert Elz and Randy Bush, *Serial number arithmetic*, RFC 1982, August 1996.

Paul Vixie, *A mechanism for prompt notification of zone changes (DNS NOTIFY)*, RFC 1996, August 1996.

AUTHOR

Tony Finch ⟨`dot@dotat.at`⟩ ⟨`fanf2@cam.ac.uk`⟩
at Cambridge University Information Services