PowerDNS operation

PowerDNS BV

pdns@powerdns.com

PowerDNS operation
by

Published v1.0 $Date: 2002/04/09 09:35:56 $

How to install, launch and operate the PowerDNS nameserver

Table of Contents

List of Tables

Chapter 1. The PowerDNS dynamic nameserver

The PowerDNS daemon is a versatile nameserver which supports a large number of backends. These backends can either be plain zonefiles or be more dynamic in nature.

Prime examples of backends include relational databases, but also loadbalancing and failover algorithms.

The company is called PowerDNS BV, the nameserver daemon is called PDNS.

1.1. Function & design of PDNS

PDNS is an authoritative only nameserver. It will answer questions about domains it knows about, but will not go out on the net to resolve queries about other domains. However, it can use a recursing backend to provide that functionality.

When PDNS answers a question, it comes out of the database, and can be trusted as being authoritative. There is no way to pollute the cache or to confuse the daemon.

PDNS has been designed to serve both the needs of small installations by being easy to setup, as well as for serving very large query volumes on large numbers of domains.

Another prime goal is security. By the use of language features, the PDNS source code is very small (in the order of 10.000 lines) which makes auditing easy. In the same way, library features have been used to mitigate the risks of buffer overflows.

Finally, PDNS is able to give a lot of statistics on its operation which is both helpful in determining the scalability of an installation as well as for spotting problems.

1.2. Release notes

Before proceeding, you should check the release notes for your PDNS version, as specified in the name of the distribution file.

1.2.1. 1.99.8 Early Access Prerelease

A lot of infrastructure work gearing up to 2.0. Some stability bugs fixed and a lot of new features.

Bugs fixed:

- Bindbackend was overly complex and crashed on some systems on startup. Simplified launch code.
- SOA fields were not always properly filled in, causing default values to go out on the wire
- Obscure bug triggered by malicious packets (we know who you are) in SOA finding code fixed.
- Magic serial number calculation contained a double free leading to instability.
- Standards violation, questions for domains for which PDNS was unauthoritative now get a SERVFAIL answer. Thanks to the IETF Namedroppers list for helping out with this.
- Slowly launching backends were being relaunched at a great rate when queries were coming in while launching backends.
- MySQL-on-unix-domain-socket on SMP systems was overwhelmed by the quick connection rate on launch, inserted a small 50ms delay.
- Some SMP problems appear to be compiler related. Shifted to GCC 3.0.4 for Linux.
- Ran ispell on documentation.

Feature enhancements:

- Recursing backend. See Chapter 11. Allows recursive and authoritative DNS on the same IP address.
- NAPTR support, which is especially useful for the ENUM/E.164 community.
- Zone transfers can now be allowed per netmask instead of only per IP address.
- Preliminary support for slave operation included. Only for the adventurous right now! See Chapter 12
- All record types now documented, see Chapter 15.

1.2.1.1. Known bugs

Wildcard CNAMES do not work as they do with bind.

1.2.1.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend

The Oraclebackend is available on request and will most likely need to be tailored to your installation.

1.2.2. 1.99.7 Early Access Prerelease

Named.conf parsing got a lot of work and many more bind configurations can now be parsed. Furthermore, error reporting was improved. Stability is looking good.

Bugs fixed:

- Bind parser got confused by filenames with underscores and colons.
- Bind parser got confused by spaces in quoted names
- FreeBSD version now stops and starts when instructed to do so.
- Wildcards were off by default, which violates standards. Now on by default.
- --oracle was broken in zone2sql

Feature enhancements:

- Line number counting goes on as it should when including files in named.conf
- Added --no-config to enable users to start the pdns daemon without parsing the configuration file.
- zone2sql now has --bare for unformatted output which can be used to generate insert statements for different database layouts
- zone2sql now has --gpgsql, which is an alias for --mysql, to output in a format useful for the default Generic PgSQL backend
- zone2sql is now documented.

1.2.2.1. Known bugs

Wildcard CNAMES do not work as they do with bind.

1.2.2.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend

Some of these features will be present in newer releases.

1.2.3. 1.99.6 Early Access Prerelease

This version is now running on dns-eu1.powerdns.net and working very well for us. But please remain cautious before deploying!

Bugs fixed:

- Webserver neglected to show log messages

- TCP question/answer miscounted multiple questions over one socket. Fixed misnaming of counter
- Packetcache now detects clock skew and times out entries
- named.conf parser now reports errors with line number and offending token
- Filenames in named.conf can now contain :

Feature enhancements:

- The webserver now by default does not print out configuration statements, which might contain database backends. Use **webserver-print-arguments** to restore the old behaviour.
- Generic PostgreSQL backend is now included. Still rather beta.

1.2.3.1. Known bugs

FreeBSD version does not stop when requested to do so.

Wildcard CNAMES do not work as they do with bind.

1.2.3.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend

Some of these features will be present in newer releases.

1.2.4. 1.99.5 Early Access Prerelease

The main focus of this release is stability and TCP improvements. This is the first release PowerDNS-the-company actually considers for running on its production servers!

Major bugs fixed:

- Zone2sql received a floating point division by zero error on named.confs with less than 100 domains.
- Huffman encoder failed without specific error on illegal characters in a domain
- Fixed huge memory leaks in TCP code.
- Removed further file descriptor leaks in guardian respawning code
- Pipebackend was too chatty.

- pdns_server neglected to close fds 0, 1 & 2 when daemonizing

Feature enhancements:

- bindbackend can be instructed not to check the ctime of a zone by specifying **bind-check-interval=0**, which is also the new default.
- **pdns_server --list-modules** lists all available modules.

Performance enhancements:

- TCP code now only creates a new database connection for AXFR.
- TCP connections timeout rather quickly now, leading to less load on the server.

1.2.4.1. Known bugs

FreeBSD version does not stop when requested to do so.

Wildcard CNAMES do not work as they do with bind.

1.2.4.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend, gpgsqlbackend

Some of these features will be present in newer releases.

1.2.5. 1.99.4 Early Access Prerelease

A lot of new named.confs can now be parsed, zone2sql & bindbackend have gained features and stability.

Major bugs fixed:

- Label compression was not always enabled, leading to large reply packets sometimes.
- Database errors on TCP server lead to a nameserver reload by the guardian.
- MySQL backend neglected to close its connection properly.
- BindParser miss parsed some IP addresses and netmasks.

- Truncated answers were also truncated on the packetcache, leading to truncated TCP answers.

Feature enhancements:

- Zone2sql and the bindbackend now understand the Bind $GENERATE{} syntax.
- Zone2sql can optionally gloss over non-existing zones with **--on-error-resume-next**.
- Zone2sql and the bindbackend now properly expand @ also on the right hand side of records.
- Zone2sql now sets a default TTL.
- DNS UPDATEs and NOTIFYs are now logged properly and sent the right responses.

Performance enhancements:

- 'Fancy records' are no longer queried for on ANY queries - this is a big speedup.

1.2.5.1. Known bugs

FreeBSD version does not stop when requested to do so.

Zone2sql refuses named.confs with less than 100 domains.

Wildcard CNAMES do not work as they do with bind.

1.2.5.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend, gpgsqlbackend

Some of these features will be present in newer releases.

1.2.6. 1.99.3 Early Access Prerelease

The big news in this release is the BindBackend which is now capable of parsing many more named.conf Bind configurations. Furthermore, PDNS has successfully parsed very large named.confs with large numbers of small domains, as well as small numbers of large domains (TLD).

Zone transfers are now also much improved.

Major bugs fixed:

- zone2sql leaked file descriptors on each domain, used wrong Bison recursion leading to parser stack overflows. This limited the amount of domains that could be parsed to 1024.
- zone2sql can now read all known zonefiles, with the exception of those containing $GENERATE
- Guardian relaunching a child lost two file descriptors
- Don't die on a connection reset by peer during zone transfer.
- Webserver does not crash anymore on ringbuffer resize

Feature enhancements:

- AXFR can now be disabled, and re-enabled per IP address
- --help accepts a parameter, will then show only help items with that prefix.
- zone2sql now accepts a --zone-name parameter
- BindBackend maturing - 9500 zones parsed in 3.5 seconds. No longer case sensitive.

Performance enhancements:

- Implemented RFC-breaking AXFR format (which is the industry standard). Zone transfers now zoom along at wirespeed (many megabits/s).

1.2.6.1. Known bugs

FreeBSD version does not stop when requested to do so.

BindBackend cannot parse zones with $GENERATE statements.

1.2.6.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend, gpgsqlbackend

Some of these features will be present in newer releases.

1.2.7. 1.99.2 Early Access Prerelease

Major bugs fixed:

- Database backend reload does not hang the daemon anymore
- Buffer overrun in local socket address initialisation may have caused binding problems
- setuid changed the uid to the gid of the selected user
- zone2sql doesn't coredump on invocation anymore. Fixed lots of small issues.
- Don't parse configuration file when creating configuration file. This was a problem with reinstalling.

Performance improvements:

- removed a lot of unnecessary gettimeofday calls
- removed needless select(2) call in case of listening on only one address
- removed 3 useless syscalls in the fast path

Having said that, more work may need to be done. Testing on a 486 saw packet rates in a simple setup (question/wait/answer/question..) improve from 200 queries/second to over 400.

Usability improvements:

- Fixed error checking in init.d script (**show**, **mrtg**)
- Added 'uptime' to the mrtg output
- removed further GNUisms from installer and init.d scripts for use on FreeBSD
- Debian package and apt repository, thanks to Wichert Akkerman.
- FreeBSD /usr/ports, thanks to Peter van Dijk (in progress).

Stability may be an issue as well as performance. This version has a tendency to log a bit too much which slows the nameserver down a lot.

1.2.7.1. Known bugs

Decreasing a ringbuffer on the website is a sure way to crash the daemon. Zone2sql, while improved, still has problems with a zone in the following format:

```
name          IN            A        1.2.3.4
              IN            A        1.2.3.5
```

To fix, add 'name' to the second line.

Zone2sql does not close filedescriptors.

FreeBSD version does not stop when requested via the init.d script.

1.2.7.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend, gpgsqlbackend
- fully functioning bindbackend - will try to parse named.conf, but probably fail

Some of these features will be present in newer releases.

1.2.8. 1.99.1 Early Access Prerelease

This is the first public release of what is going to become PDNS 2.0. As such, it is not of production quality. Even PowerDNS-the-company does not run this yet.

Stability may be an issue as well as performance. This version has a tendency to log a bit too much which slows the nameserver down a lot.

1.2.8.1. Known bugs

Decreasing a ringbuffer on the website is a sure way to crash the daemon. Zone2sql is very buggy.

1.2.8.2. Missing features

Features present in this document, but disabled or withheld from the current release:

- gmysqlbackend, oraclebackend, gpgsqlbackend
- fully functioning bindbackend - will not parse configuration files

Some of these features will be present in newer releases.

1.3. Acknowledgements

PowerDNS is grateful for the help of the following individuals:

- Dave Aaldering
- Wichert Akkerman

- Antony Antony
- Peter van Dijk
- Koos van den Hout
- Andre Koopal
- Eric Veldhuyzen
- Paul Wouters
- Thomas Wouters

Thanks!

Chapter 2. Installing

After unpacking the PDNS distribution the files need to be moved to appropriate locations.

PDNS can be installed in a variety of directories, which can easily be customised to local policy. Two ways are available - manual and via a menu.

The menu is invoked by executing the ’./choosepaths’ script and answering the questions. The manual way involves editing the ’pathconfig’ file. The choice is up to you.

After deciding paths, change to root and execute the ’installer’ script. This will:

- Configure the PowerDNS binary so it knows where the configuration directory is
- If necessary, create the configuration directory
- Write sample configuration file (not overwriting existing one)
- Write a SysV-style init.d script in the configured directory
- Move binaries and libraries to the configured places

2.1. Possible problems at this point

At this point some things may have gone wrong. Typical errors include:

error while loading shared libraries: libstdc++.so.x: cannot open shared object file: No such file or directory

Errors looking like this indicate a mismatch between your PDNS distribution and your Unix operating system. Download the static PDNS distribution for your operating system and try again. Please contact <pdns@powerdns.com> if this is impractical.

2.2. Testing your install

After installing, it is a good idea to test the basic functionality of the software before configuring database backends. For this purpose, PowerDNS contains the ’bindbackend’ which has a domain built in example.com, which is officially reserved for testing. To test, edit `pdns.conf` and add the following if not already present:

```
launch=bind
bind-example-zones
```

This configures powerdns to 'launch' the bindbackend, and enable the example zones. To fire up PDNS in testing mode, execute: **/etc/init.d/pdns monitor**, where you may have to substitute the location of your SysV init.d location you specified earlier. In monitor mode, the pdns process runs in the foreground and is very verbose, which is perfect for testing your install. If everything went all right, you can query the example.com domain like this:

```
host www.example.com 127.0.0.1
```

www.example.com should now have IP address 1.2.3.4. The **host** command can usually be found in the dnsutils package of your operating system. Alternate command is: **dig www.example.com A @127.0.0.1** or even **nslookup www.example.com 127.0.0.1**, although nslookup is not advised for DNS diagnostics.

- example.com SOA record
- example.com NS record pointing to ns1.example.com
- example.com NS record pointing to ns2.example.com
- example.com MX record pointing to mail.example.com
- example.com MX record pointing to mail1.example.com
- mail.example.com A record pointing to 4.3.2.1
- mail1.example.com A record pointing to 5.4.3.2
- ns1.example.com A record pointing to 4.3.2.1
- ns2.example.com A record pointing to 5.4.3.2
- host-0 to host-9999.example.com A record pointing to 2.3.4.5

When satisfied that basic functionality is there, type **QUIT** to exit the monitor mode. The adventurous may also type **SHOW *** to see some internal statistics. In case of problems, you will want to read the following section.

2.2.1. Typical errors

At this point some things may have gone wrong. Typical errors include:

binding to UDP socket: Address already in use

This means that another nameserver is listening on port 53 already. You can resolve this problem by determining if it is safe to shutdown the nameserver already present, and doing so. If uncertain, it is also possible to run PDNS on another port. To do so, add **local-port=5300** to `pdns.conf`, and try again. This however implies that you can only test your nameserver as clients expect the nameserver to live on port 53.

binding to UDP socket: Permission denied

You must be superuser in order to be able to bind to port 53. If this is not a possibility, it is also possible to run PDNS on another port. To do so, add **local-port=5300** to `pdns.conf`, and try again.

This however implies that you can only test your nameserver as clients expect the nameserver to live on port 53.

Unable to launch, no backends configured for querying

PDNS did not find the **launch=bind** instruction in pdns.conf.

Chapter 3. Running PDNS

PDNS is normally controlled via a SysV-style init.d script, often located in `/etc/init.d` or `/etc/rc.d/init.d`. This script accepts the following commands:

monitor

Monitor is a special way to view the daemon. It executes PDNS in the foreground with a lot of logging turned on, which helps in determining startup problems. Besides running in the foreground, the raw PDNS control socket is made available. All external communication with the daemon is normally sent over this socket. While useful, the control console is not an officially supported feature. Commands which work are: **QUIT**, **SHOW ***, **SHOW varname**, **RPING**.

start

Start PDNS in the background. Launches the daemon but makes no special effort to determine success, as making database connections may take a while. Use **status** to query success. You can safely run **start** many times, it will not start additional PDNS instances.

restart

Restarts PDNS if it was running, starts it otherwise.

status

Query PDNS for status. This can be used to figure out if a launch was successful. The status found is prefixed by the PID of the main PDNS process. In case of problems,

stop

Requests that PDNS stop. Again, does not confirm success. Success can be ascertained with the **status** command.

dump

Dumps a lot of statistics of a running PDNS daemon. It is also possible to single out specific variable by using the **show** command.

show variable

Show a single statistic, as present in the output of the **dump**.

mrtg

See the performance monitoring chapter.

Chapter 4. Configure database connectivity

The default PDNS distribution comes with a simple MySQL backend built in, which we will now use for demonstrating database connectivity. This backend is called 'mysql', and needs to be configured in `pdns.conf`. Add the following lines, adjusted for your local setup:

```
launch=mysql
mysql-host=127.0.0.1
mysql-user=root
mysql-dbname=pdnstest
```

Remove any earlier **launch** statements. Also remove the **bind-example-zones** statement as the **bind** module is no longer launched.

WARNING! Make sure that you can actually resolve the hostname of your database without accessing the database! It is advised to supply an IP address here to prevent chicken/egg problems!

Now start PDNS using the monitor command:

```
# /etc/init.d/pdns monitor
(...)
15:31:30 PowerDNS 1.99.0 (Mar 12 2002, 15:00:28) starting up
15:31:30 About to create 3 backend threads
15:31:30 [MySQLbackend] Failed to connect to database: Error: Unknown database 'pdnstest'
15:31:30 [MySQLbackend] Failed to connect to database: Error: Unknown database 'pdnstest'
15:31:30 [MySQLbackend] Failed to connect to database: Error: Unknown database 'pdnstest'
```

This is as to be expected - we did not yet add anything to MySQL for PDNS to read from. At this point you may also see other errors which indicate that PDNS either could not find your MySQL server or was unable to connect to it. Fix these before proceeding.

General MySQL knowledge is assumed in this chapter, please do not interpret these commands as DBA advice!

4.1. Configuring MySQL

Connect to MySQL as a user with sufficient privileges and issue the following commands:

```
# mysql
mysql> CREATE DATABASE pdnstest;
mysql> use pdnstest;

mysql> CREATE TABLE records (
id int(11) NOT NULL auto_increment,
```

```
domain_id int(11) default NULL,
name varchar(255) default NULL,
type varchar(6) default NULL,
content varchar(255) default NULL,
ttl int(11) default NULL,
prio int(11) default NULL,
change_date int(11) default NULL,
PRIMARY KEY (id),
KEY name_index(name),
KEY nametype_index(name,type),
KEY domainid_index(domain_id)
);
```

Now we have a database and an empty table. PDNS should now be able to launch in monitor mode and display no errors:

```
# /etc/init.d/pdns monitor
(...)
15:31:30 PowerDNS 1.99.0 (Mar 12 2002, 15:00:28) starting up
15:31:30 About to create 3 backend threads
15:39:55 [MySQLbackend] MySQL connection succeeded
15:39:55 [MySQLbackend] MySQL connection succeeded
15:39:55 [MySQLbackend] MySQL connection succeeded
```

A sample query sent to the database should now return quickly without data:

```
$ host www.test.com 127.0.0.1
www.test.com A record currently not present at localhost
```

And indeed, the control console now shows:

```
Mar 12 15:41:12 We're not authoritative for 'www.test.com', sending unauth normal respons
```

Now we need to add some records to our database:

```
# mysql pdnstest
mysql>
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'test.com','localhost ahu@ds9a.nl 1','SOA',86400,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'test.com','dns-us1.powerdns.net','NS',86400,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'test.com','dns-eu1.powerdns.net','NS',86400,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'www.test.com','199.198.197.196','A',120,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'mail.test.com','195.194.193.192','A',120,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'localhost.test.com','127.0.0.1','A',120,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
VALUES (1,'test.com','mail.test.com','MX',120,25);
```

If we now requery our database, **www.test.com** should be present:

```
$ host www.test.com 127.0.0.1
www.test.com         A 199.198.197.196

$ host -v -t mx test.com 127.0.0.1
Address: 127.0.0.1
Aliases: localhost

Query about test.com for record types MX
Trying test.com ...
Query done, 1 answer, authoritative status: no error
test.com             120 IN MX 25 mail.test.com
Additional information:
mail.test.com        120 IN A 195.194.193.192
```

To confirm what happened, issue the command **SHOW *** to the control console:

```
% show *
corrupt-packets=0,latency=0,packetcache-hit=2,packetcache-miss=5,packetcache-size=0,
qsize-a=0,qsize-q=0,servfail-packets=0,tcp-answers=0,tcp-queries=0,
timedout-packets=0,udp-answers=7,udp-queries=7,
%
```

The actual numbers will vary somewhat. Now enter **QUIT** and start PDNS as a regular daemon, and check launch status:

```
# /etc/init.d/pdns start
pdns: started
# /etc/init.d/pdns status
pdns: 8239: Child running
# /etc/init.d/pdns dump
pdns: corrupt-packets=0,latency=0,packetcache-hit=0,packetcache-miss=0,
packetcache-size=0,qsize-a=0,qsize-q=0,servfail-packets=0,tcp-answers=0,
tcp-queries=0,timedout-packets=0,udp-answers=0,udp-queries=0,
```

You now have a working database driven nameserver! To convert other zones already present, use the **zone2sql** described in Appendix A.

4.1.1. Common problems

Most problems involve PDNS not being able to connect to the database.

Can’t connect to local MySQL server through socket ’/tmp/mysql.sock’ (2)

> Your MySQL installation is probably defaulting to another location for its socket. Can be resolved by figuring out this location (often `/var/run/mysqld.sock`), and specifying it in the configuration file with the **mysql-socket** parameter.
>
> Another solution is to not connect to the socket, but to 127.0.0.1, which can be achieved by specifying **mysql-host=127.0.0.1**.

Host ’x.y.z.w’ is not allowed to connect to this MySQL server

These errors are generic MySQL errors. Solve them by trying to connect to your MySQL database with the MySQL console utility **mysql** with the parameters specified to PDNS. Consult the MySQL documentation.

Chapter 5. Dynamic resolution using the PipeBackend

Also included in the PDNS distribution is the PipeBackend. The PipeBackend is primarily meant for allowing rapid development of new backends without tight integration with PowerDNS. It allows end-users to write PDNS backends in any language. A perl sample is provided. The PipeBackend is also very well suited for dynamic resolution of queries. Example applications include DNS based loadbalancing, geo-direction, DNS based failover with low TTLs.

5.1. Deploying the PipeBackend with the BindBackend

Included with the PDNS distribution is the example.pl backend which has knowledge of the example.com zone, just like the BindBackend. To install both, add the following to your `pdns.conf`:

```
launch=pipe,bind
bind-example-zones
pipe-command=location/of/backend.pl
```

Please adjust the **pipe-command** statement to the location of the unpacked PDNS distribution. Now launch PDNS in monitor mode, and perform some queries. Note the difference with the earlier experiment where only the BindBackend was loaded. The PipeBackend is launched first and thus gets queried first. The sample backend.pl script knows about:

- webserver.example.com A records pointing to 1.2.3.4, 1.2.3.5, 1.2.3.6
- www.example.com CNAME pointing to webserver.example.com
- MBOXFW (mailbox forward) records pointing to powerdns@example.com. See the smtpredir documentation for information about MBOXFW.

For more information about how to write exciting backends with the PipeBackend, see Appendix A.

Chapter 6. Monitoring PDNS performance

In a production environment, you will want to be able to monitor PDNS performance. For this purpose, currently two methods are available, the webserver and the init.d **dump**, **show** and **mrtg**, commands.

6.1. Webserver

To launch the internal webserver, add a **webserver** statement to the pdns.conf. This will instruct the PDNS daemon to start a webserver on localhost at port 8081, without password protection. Only local users (on the same host) will be able to access the webserver by default. The webserver lists a lot of information about the PDNS process, including frequent queries, frequently failing queries, lists of remote hosts sending queries, hosts sending corrupt queries etc. The webserver does not allow remote management of the daemon. The following nameserver related configuration items are available:

webserver

If set to anything but 'no', a webserver is launched.

webserver-address

Address to bind the webserver to. Defaults to 127.0.0.1, which implies that only the local computer is able to connect to the nameserver! To allow remote hosts to connect, change to 0.0.0.0 or the physical IP address of your nameserver.

webserver-password

If set, viewers will have to enter this plaintext password in order to gain access to the statistics.

webserver-port

Port to bind the webserver to. Defaults to 8081.

6.2. Via init.d commands

As mentioned before, the init.d commands **dump**, **show** and **mrtg** fetch data from a running PDNS process. Especially **mrtg** is powerful - it outputs data in a format that is ready for processing by the MRTG graphing tool.

MRTG can make insightful graphics on the performance of your nameserver, enabling the operator to easily spot trends. MRTG can be found on http://people.ee.ethz.ch/~oetiker/webtools/mrtg/mrtg.html (http://people.ee.ethz.ch/~oetiker/webtools/mrtg/mrtg.html)

A sample mrtg.conf:

```
Interval: 5
WorkDir: /var/www/mrtg
WriteExpires: yes
Options[_]: growright,nopercent
XSize[_]: 600

#--------------------------------------------------------------

Target[udp-queries]: '/etc/init.d/pdns mrtg udp-queries udp-answers'
Options[udp-queries]: growright,nopercent,perminute
MaxBytes[udp-queries]: 600000
AbsMax[udp-queries]: 600000
Title[udp-queries]: Queries per minute
PageTop[udp-queries]: <H2>Queries per minute</H2>
WithPeak[udp-queries]: ymwd
YLegend[udp-queries]: queries/minute
ShortLegend[udp-queries]: q/m
LegendI[udp-queries]: udp-questions
LegendO[udp-queries]: udp-answers


Target[perc-failed]: '/etc/init.d/pdns mrtg udp-queries udp-answers'
Options[perc-failed]: growright,dorelpercent,perminute
MaxBytes[perc-failed]: 600000
AbsMax[perc-failed]: 600000
Title[perc-failed]: Queries per minute, with percentage success
PageTop[perc-failed]: <H2>Queries per minute, with percentage success</H2>
WithPeak[perc-failed]: ymwd
YLegend[perc-failed]: queries/minute
ShortLegend[perc-failed]: q/m
LegendI[perc-failed]: udp-questions
LegendO[perc-failed]: udp-answers


Target[packetcache-rate]: '/etc/init.d/pdns mrtg packetcache-hit udp-queries'
Options[packetcache-rate]: growright,dorelpercent,perminute
Title[packetcache-rate]: packetcache hitrate
MaxBytes[packetcache-rate]: 600000
AbsMax[packetcache-rate]: 600000
PageTop[packetcache-rate]: <H2>packetcache hitrate</H2>
WithPeak[packetcache-rate]: ymwd
YLegend[packetcache-rate]: queries/minute
ShortLegend[packetcache-rate]: q/m
LegendO[packetcache-rate]: total
LegendI[packetcache-rate]: hit

Target[packetcache-missrate]: '/etc/init.d/pdns mrtg packetcache-miss udp-queries'
Options[packetcache-missrate]: growright,dorelpercent,perminute
Title[packetcache-missrate]: packetcache MISSrate
MaxBytes[packetcache-missrate]: 600000
AbsMax[packetcache-missrate]: 600000
PageTop[packetcache-missrate]: <H2>packetcache MISSrate</H2>
WithPeak[packetcache-missrate]: ymwd
```

```
YLegend[packetcache-missrate]: queries/minute
ShortLegend[packetcache-missrate]: q/m
LegendO[packetcache-missrate]: total
LegendI[packetcache-missrate]: MISS

Target[latency]: '/etc/init.d/pdns mrtg latency'
Options[latency]: growright,nopercent,gauge
MaxBytes[latency]: 600000
AbsMax[latency]: 600000
Title[latency]: Query/answer latency
PageTop[latency]: <H2>Query/answer latency</H2>
WithPeak[latency]: ymwd
YLegend[latency]: usec
ShortLegend[latency]: usec
LegendO[latency]: latency
LegendI[latency]: latency

Target[recursing]: '/etc/init.d/pdns mrtg recursing-questions recursing-answers'
Options[recursing]: growright,nopercent,gauge
MaxBytes[recursing]: 600000
AbsMax[recursing]: 600000
Title[recursing]: Recursive questions/answers
PageTop[recursing]: <H2>Recursing questions/answers</H2>
WithPeak[recursing]: ymwd
YLegend[recursing]: queries/minute
ShortLegend[recursing]: q/m
LegendO[recursing]: recursing-questions
LegendI[recursing]: recursing-answers
```

Chapter 7. Security settings & considerations

7.1. Settings

PDNS has several options to easily allow it to run more securely. Most notable are the **chroot**, **setuid** and **setgid** options which can be specified.

7.1.1. Running as a less privileged identity

By specifying **setuid** and **setgid**, PDNS changes to this identity shortly after binding to the privileged DNS ports. These options are highly recommended. It is suggested that a separate identity is created for PDNS as the user 'nobody' is in fact quite powerful on most systems.

Both these parameters can be specified either numerically or as real names. You should set these parameters immediately if they are not set!

7.1.2. Jailing the process in a chroot

The **chroot** option secures PDNS to its own directory so that even if it should become compromised and under control of external influences, it will have a hard time affecting the rest of the system.

Even though this will hamper hackers a lot, chroot jails have been known to be broken.

When chrooting PDNS, take care that backends will be able to get to their files. Many databases need access to a UNIX domain socket which should live within the chroot. It is often possible to hardlink such a socket into the chroot dir.

The default PDNS configuration is best chrooted to `./`, which boils down to the configured location of the controlsocket.

This is achieved by adding the following to pdns.conf: **chroot=./**, and restarting PDNS.

7.2. Considerations

In general, make sure that the PDNS process is unable to execute commands on your backend database. Most database backends will only need SELECT privilege. Take care to not connect to your database as the 'root' or 'sa' user, and configure the chosen user to have very slight privileges.

Databases empathic-ally do not need to run on the same machine that runs PDNS! In fact, in benchmarks it has been discovered that having a separate database machine actually improves performance.

Separation will enhance your database security highly. Recommended.

Chapter 8. Virtual hosting

It may be advantageous to run multiple separate PDNS installations on a single host, for example to make sure that different customers cannot affect each others zones. PDNS fully supports running multiple instances on one host.

To generate additional PDNS instances, copy the init.d script `pdns` to `pdns-name`, where `name` is the name of your virtual configuration. Must not contain a - as this will confuse the script.

When you launch PDNS via this renamed script, it will seek configuration instructions not in `pdns.conf` but in `pdns-name.conf`, allowing for separate specification of parameters.

Be aware however that the init.d **force-stop** will kill all PDNS instances!

Chapter 9. Performance related settings

Different backends will have different characteristics - some will want to have more parallel instances than others. In general, if your backend is latency bound, like most relational databases are, it pays to open more backends.

This is done with the **distributor-threads** setting. Of special importance is the choice between 1 or more backends. In case of only 1 thread, PDNS reverts to unthreaded operation which may be a lot faster, depending on your operating system and architecture.

Another very important setting **cache-ttl**. PDNS caches entire packets it sends out so as to save the time to query backends to assemble all data. The default setting of 10 seconds may be low for high traffic sites, a value of 60 seconds rarely leads to problems.

To determine if PDNS is unable to keep up with packets, determine the value of the **qsize-q** variable. This represents the number of packets waiting for database attention. During normal operations the queue should be small.

If it is known that backends will not contain CNAME records, the **skip-cname** setting can be used to prevent the normally mandatory CNAME lookup that is needed at least once for each DNS query.

Much the same holds for the **wildcards** setting. On by default, each non-existent query will lead to a number of additional wildcard queries. If it is known that the backends do not contain wildcard records, performance can be improved by adding **wildcards=no** to `pdns.conf`.

9.1. PacketCache

PDNS by default uses the 'PacketCache' to recognise identical questions and supply them with identical answers, without any further processing. The default time to live is 10 seconds. It has been observed that the utility of the packet cache increases with the load on your nameserver.

Not all backends may benefit from the packetcache. If your backend is memory based and does not lead to context switches, the packetcache may actually hurt performance.

The size of the packetcache can be observed with **/etc/init.d/pdns show packetcache-size**

Chapter 10. Migrating to PDNS

Before migrating to PDNS a few things should be considered.

PDNS is not a recursing nameserver on its own

If PDNS receives a question for which it is not authoritative, it can't go out on the net to figure out an answer. However, because many installations are expected to be both authoritative and recursing, PDNS can use a separate recursing backend to provide non-authoritative answers. See Chapter 11 for more details.

PDNS does not operate as a 'slave' server

PDNS will happily read your Bind named.conf but will not, right now, honour statements regarding master/slave operation. The recommended setup is to replicate nameserver data by using your database's native replication facility, or, when using zonefiles, rsync.

To migrate, the **zone2sql** tool is provided.

10.1. Zone2sql

Zone2sql parses Bind named.conf files and zonefiles and outputs SQL on standard out, which can then be fed to your database.

Zone2sql understands the Bind master file extension '$GENERATE' and will also honour '$ORIGIN' and '$TTL'.

By default, zone2sql outputs code suitable for the mysqlbackend, which can also be read by PostgreSQL incidentally. The following commands are available:

--bare

Output in a bare format, suitable for further parsing. The output is formatted as follows:

```
domain_id<TAB>'qname'<TAB>'qtype'<TAB>'content'<TAB>prio<TAB>ttl
```

--gpgsql

Output in format suitable for the default configuration of the Generic PostgreSQL backend.

--help

List options.

--mysql

Output in format suitable for the default configuration of the MySQL backend. Default.

--named-conf=...

Parse this named.conf to find locations of zones.

--on-error-resume-next

Ignore missing files during parsing. Dangerous.

--oracle

Output in format suitable for the default configuration of the Generic Oracle backend.

--startid

Supply a value for the first domain_id generated. Defaults at 0.

--verbose

Be verbose during conversion.

--zone=...

Parse only this zone file. Conflicts with **--named-conf** parameter.

--zone-name=...

When parsing a single zone without $ORIGIN statement, set this as the zone name.

Chapter 11. Recursion

(only available from 1.99.8 and onwards)

PDNS is an authoritative nameserver. It answers questions with data from its backends. Besides handing out authoritative answers, DNS also needs so called 'recursion', where a nameserver gets a question for it has no authoritative answer available, necessitating questions to other nameservers.

Although PDNS is an authoritative nameserver, a provision has been made to cater for installations that require both authoritative DNS and recursion on one IP address.

By specifying the **recursor** option in the configuration file, questions requiring recursive treatment will be handed over to the IP address specified. An example configuration might be **recursor=130.161.180.1**, which designates 130.161.180.1 as the nameserver to handle recursive queries.

Any recursing nameserver is suitable but we highly advise the use of the DJBDNS dnscache (http://cr.yp.to/djbdns/dnscache.html).

Take care not to point **recursor** to PDNS, which leads to a very tight packet loop!

By specifying **allow-recursion**, recursion can be restricted to netmasks specified. The default is to allow recursion from everywhere. Example: **allow-recursion=192.168.0.0/24, 10.0.0.0/8, 1.2.3.4**.

11.1. Details

Questions carry a number of flags. One of these is called 'Recursion Desired'. If PDNS is configured to allow recursion, AND such a flag is seen, AND the IP address of the client is allowed to recurse via PDNS, then the packet is handed to the recursing backend.

If a Recursion Desired packet PDNS is configured to allow recursion, but not to the IP address of the client, resolution will proceed as if the RD flag were unset and the answer will indicate that recursion was not available.

Recursive questions and answers are not stored in the Packet Cache as recursing backends are generally well equipped to cache questions themselves.

Chapter 12. Slave operation

As of 1.99.8 this is not for the feint of heart, but please test and let us know on the mailinglists how you fare! Currently this is only working reasonably well for the PostgreSQL backend. To test, issue the following on a clean database:

```
create table domains (
 id  SERIAL PRIMARY KEY,
 name  VARCHAR(255) NOT NULL,
 master  VARCHAR(20) NOT NULL,
 last_check  INT DEFAULT NULL,
 type  VARCHAR(6) NOT NULL
);
CREATE INDEX name_index ON domains(name);
GRANT UPDATE ON domains TO pdns;
GRANT SELECT ON domains_id_seq TO pdns;

CREATE TABLE records (
        id              SERIAL PRIMARY KEY,
        domain_id       INT DEFAULT NULL,
        name            VARCHAR(255) DEFAULT NULL,
        type            VARCHAR(6) DEFAULT NULL,
        content         VARCHAR(255) DEFAULT NULL,
        ttl             INT DEFAULT NULL,
        prio            INT DEFAULT NULL,
        change_data     INT DEFAULT NULL,
        CONSTRAINT domain_exists
        FOREIGN KEY(domain_id) REFERENCES domains(id)
        ON DELETE CASCADE
);

CREATE INDEX name_index ON records(name);
CREATE INDEX nametype_index ON records(name,type);
CREATE INDEX domain_id ON records(domain_id);

GRANT ALL ON records TO pdns;
GRANT ALL ON records_id_seq TO pdns;
```

Now launch PDNS with the **slave** option in `pdns.conf`. It should launch normally and report that 'All slave domains are fresh'.

Now connect to your database as user pdns. To become a slave of the 'powerdns.com' domain, execute this:

```
insert into domains (name,master,type) values ('powerdns.com','213.244.168.217','SLAVE');
```

And wait awhile for PDNS to pick up the addition - which happens within one minute. Typical output is:

```
Apr 09 13:34:29 All slave domains are fresh
Apr 09 13:35:29 1 slave domain needs checking
Apr 09 13:35:29 Domain powerdns.com is stale, master serial 1, our serial 0
Apr 09 13:35:30 [gPgSQLBackend] Connecting to database with connect string 'dbname=pdns use
Apr 09 13:35:30 AXFR started for 'powerdns.com'
unknown datatype in answer: 28
Apr 09 13:35:30 AXFR done for 'powerdns.com'
Apr 09 13:35:30 [gPgSQLBackend] Closing connection
```

Note the 'unknown datatype in answer' - the secondary code does not yet know how to handle AAAA.

To reiterate, as of 1.99.8 this is not yet a mature and stable feature! The *really* adventurous may even find that this also works with the bind backend, to some extent.

12.1. Details

On launch, PDNS requests from all backends a list of domains which have not been checked recently for changes. This should happen every **refresh** seconds, as specified in the SOA record. All domains that are unfresh are then checked for changes over at their master. If the SOA there is higher, the domain is retrieved and inserted into the database. In any case, after the check the domain is declared 'unstale', and will only be checked again after **refresh** seconds have passed.

PDNS also reacts to notifies by immediately checking if the zone has updated and if so, retransfering it.

All backends which implement this feature must make sure that they can handle transactions so as to not leave the zone in a half updated state. MySQL configured with either BerkeleyDB or InnoDB meets this requirement, as do PostgreSQL and Oracle. The Bindbackend implements transaction semantics by renaming files if and only if they have been retrieved completely and parsed correctly.

Chapter 13. Index of all settings

All PDNS settings are listed here, excluding those that originate from backends, which are documented in the relevant chapters.

allow-axfr-ips=...

When not allowing AXFR (disable-axfr), DO allow from these IP addresses or netmasks.

cache-ttl=...

Seconds to store packets in the PacketCache. See Section 9.1.

chroot=...

If set, chroot to this directory for more security. See Chapter 7.

config-dir=...

Location of configuration directory (pdns.conf)

config-name=...

Name of this virtual configuration - will rename the binary image. See Chapter 8.

control-console=...

Debugging switch - don’t use.

daemon=...

Operate as a daemon

default-soa-name=...

name to insert in the SOA record if none set in the backend

disable-axfr=...

Do not allow zone transfers

disable-tcp=...

Do not listen to TCP queries. Breaks RFC compliance.

distributor-threads=...

Default number of Distributor (backend) threads to start. See Chapter 9.

fancy-records=...

Process URL and MBOXFW records

guardian | --guardian=yes | --guardian=no

Run within a guardian process. See Section B.2.

help

Provide a helpful message

launch=...

Which backends to launch and order to query them in. See Section B.3.

load-modules=...

Load this module - supply absolute or relative path. See Section B.3.

local-address=...

Local IP address to which we bind

local-port=...

The port on which we listen

loglevel=...

Amount of logging. Higher is more. Do not set below 3

module-dir=...

Default directory for modules. See Section B.3.

no-config

Do not attempt to read the configuration file.

out-of-zone-additional-processing | --out-of-zone-additional-processing=yes | --out-of-zone-additional-processing=no

Do out of zone additional processing

queue-limit=...

Maximum number of miliseconds to queue a query. See Chapter 9.

recursor=...

If set, recursive queries will be handed to the recursor specified here. See Chapter 11.

setgid=...

If set, change group id to this gid for more security. See Chapter 7.

setuid=...

If set, change user id to this uid for more security. See Chapter 7.

skip-cname | --skip-cname=yes | --skip-cname=no

Do not perform CNAME indirection for each query. Has performance implications. See Chapter 7.

smtpredirector=...

Our smtpredir MX host.

socket-dir=...

Where the controlsocket will live. See Section B.1.

urlredirector=...

Where we send hosts to that need to be url redirected

webserver | --webserver=yes | --webserver=no

Start a webserver for monitoring. See Chapter 6.

webserver-address=...

IP Address of webserver to listen on. See Chapter 6.

webserver-password=...

Password required for accessing the webserver. See Chapter 6.

webserver-port=...

Port of webserver to listen on. See Chapter 6.

wildcards=...

Hon or wildcards in the database. On by default. Turning this off has performance implications, see Chapter 9.

Chapter 14. Index of all internal metrics

14.1. Counters & variables

A number of counters and variables are set during PDNS operation. These can be queried with the init.d **dump**, **show** and **mrtg** commands, or viewed with the webserver.

corrupt-packets
: Number of corrupt packets received

latency
: Average number of microseconds a packet spends within PDNS

packetcache-hit
: Number of packets which were answered out of the cache

packetcache-miss
: Number of times a packet could not be answered out of the cache

packetcache-size
: Amount of packets in the packetcache

qsize-a
: Size of the queue before the transmitting socket.

qsize-q
: Number of packets waiting for database attention

servfail-packets
: Amount of packets that could not be answered due to database problems

tcp-answers
: Number of answers sent out over TCP

tcp-questions
: Number of questions received over TCP

timedout-questions
: Amount of packets that were dropped because they had to wait too long internally

udp-answers

Number of answers sent out over UDP

udp-questions

Number of questions received over UDP

14.1.1. Ring buffers

Besides counters, PDNS also maintains the ringbuffers. A ringbuffer records events, each new event gets a place in the buffer until it is full. When full, earlier entries get overwritten, hence the name 'ring'.

By counting the entries in the buffer, statistics can be generated. These statistics can currently only be viewed using the webserver and are in fact not even collected without the webserver running.

The following ringbuffers are available:

Log messages (logmessages)

All messages logged

Queries for existing records but for a type we don't have (noerror-queries)

Queries for, say, the AAAA record of a domain, when only an A is available. Queries are listed in the following format: name/type. So an AAA query for pdns.powerdns.com looks like pdns.powerdns.com/AAAA.

Queries for non-existing records within existing domains(nxdomain-queries)

If PDNS knows it is authoritative over a domain, and it sees a question for a record in that domain that does not exist, it is able to send out an authoritative 'no such domain' message. Indicates that hosts are trying to connect to services really not in your zone.

UDP queries received (udp-queries)

All UDP queries seen.

Remote server IP addresses (remotes)

Hosts querying PDNS. Be aware that UDP is anonymous - person A can send queries that appear to be coming from person B.

Remotes sending corrupt packets (remote-corrupts)

Hosts sending PDNS broken packets, possibly meant to disrupt service. Be aware that UDP is anonymous - person A can send queries that appear to be coming from person B.

Remotes querying domains for which we are not auth (remote-unauth)

> It may happen that there are misconfigured hosts on the internet which are configured to think that a PDNS installation is in fact a resolving nameserver. These hosts will not get useful answers from PDNS. This buffer lists hosts sending queries for domains which PDNS does not know about.

Queries that could not be answered due to backend errors (servfail-queries)/term>

> For one reason or another, a backend may be unable to extract answers for a certain domain from its storage. This may be due to a corrupt database or to inconsistent data. When this happens, PDNS sends out a 'servfail' packet indicating that it was unable to answer the question. This buffer shows which queries have been causing servfails.

Queries for domains that we are not authoritative for (unauth-queries)

> If a domain is delegated to a PDNS instance, but the backend is not made aware of this fact, questions come in for which no answer is available, nor is the authority. Use this ringbuffer to spot such queries.

Chapter 15. Supported record types and their storage

This chapter lists all record types PDNS supports, and how they are stored in backends. The list is mostly alphabetical but some types are grouped.

A

The A record contains an IP address. It is stored as a decimal dotted quad string, for example: ’213.244.168.210’.

AAAA

The AAAA record contains an IPv6 address. It is stored as a decimal dotted quad string, for example: ’3ffe:8114:2000:bf0::1’.

CNAME

The CNAME record specifies the canonical name of a record. It is stored plainly. Like all other records, it is not terminated by a dot. A sample might be ’webserver-01.yourcompany.com’.

HINFO

Hardware Info record, used to specify CPU and operating system. Stored with a single space separating these two, example: ’i386 Linux’.

MX

The MX record specifies a mail exchanger host for a domain. Each mail exchanger also has a priority or preference. This should be specified in the separate field dedicated for that purpose, often called ’prio’.

NAPTR

Naming Authority Pointer, RFC 2915. Stored as follows:

```
’100  50  "s"  "z3950+I2L+I2C"     ""  _z3950._tcp.gatech.edu’.
```

The fields are: order, preference, flags, service, regex, replacement. Note that the replacement is not enclosed in quotes, and should not be. The replacement may be omitted, in which case it is empty. See also RFC 2916 for how to use NAPTR for ENUM (E.164) purposes.

NS

Nameserver record. Specifies nameservers for a domain. Stored plainly: ’ns1.powerdns.com’, as always without a terminating dot.

PTR

Reverse pointer, used to specify the host name belonging to an IP or IPv6 address. Name is stored plainly: ’www.powerdns.com’. As always, no terminating dot.

RP

Responsible Person record, as described in RFC 1183. Stored with a single space between the mailbox name and the more-information pointer. Example 'peter.powerdns.com peter.people.powerdns.com', to indicate that peter@powerdns.com is responsible and that more information about peter is available by querying the TXT record of peter.people.powerdns.com.

SOA

The Start of Authority record is one of the most complex available. It specifies a lot about a domain: the name of the master nameserver ('the primary'), the hostmaster and a set of numbers indicating how the data in this domain expires and how often it needs to be checked. Further more, it contains a serial number which should rise on each change of the domain.

The stored format is:

```
primary hostmaster serial refresh retry expire default_ttl
```

Besides the primary and the hostmaster, all fields are numerical. PDNS has a set of default values:

Table 15-1. SOA fields

primary	**default-soa-name** configuration option
hostmaster	hostmaster@domain-name
serial	0
refresh	10800 (3 hours)
retry	3600 (1 hour)
expire	604800 (1 week)
default_ttl	3600 (1 hour)

The fields have complicated and sometimes controversial meanings. The 'serial' field is special. If left at 0, the default, PDNS will perform an internal list of the domain to determine highest change_date field of all records within the zone, and use that as the zone serial number. This means that the serial number is always raised when changes are made to the zone, as long as the change_date field is being set.

TXT

The TXT field can be used to attach textual data to a domain. Text is stored plainly.

Appendix A. Backends in detail

This appendix lists several of the available backends in more detail

A.1. PipeBackend protocol

Questions come in over a file descriptor, by default standard input. Answers are sent out over another file descriptor, standard output by default.

A.1.1. Handshake

PowerDNS sends out 'HELO\t1', indicating that it wants to speak the protocol as defined in this document, version 1. A PowerDNS CoProcess must then send out a banner, prefixed by 'OK\t', indicating it launched successfully. If it does not support the indicated version, it should respond with FAIL, but not exit. Suggested behaviour is to try and read a further line, and wait to be terminated.

A.1.2. Questions

Questions come in three forms and are prefixed by a tag indicating the kind:

Q

Regular queries

AXFR

List requests, which mean that an entire zone should be listed

PING

Check if the coprocess is functioning

The question format:

```
type qname qclass qtype id ip-address
```

Fields are tab separated, and terminated with a single \n. Type is the tag above, qname is the domain the question is about. qclass is always 'IN' currently, denoting an INternet question. qtype is the kind of information desired, the record type, like A, CNAME or AAAA. id can be specified to help your backend find an answer if the id is already known from an earlier query. You can ignore it. ip-address is the ip-address of the nameserver asking the question.

A.1.3. Answers

Each answer starts with a tag, possibly followed by a TAB and more data.

DATA

Indicating a successful line of DATA

END

Indicating the end of an answer - no further data

FAIL

Indicating a lookup failure. Also serves as 'END'. No further data.

LOG

For specifying things that should be logged. Can only be sent after a query and before an END line. After the tab, the message to be logged

So letting it be known that there is no data consists if sending 'END' without anything else. The answer format:

```
DATA qname qclass qtype ttl id content
```

A sample dialogue may look like this:

```
Q www.ds9a.nl IN CNAME -1 213.244.168.210
DATA www.ds9a.nl IN CNAME 3600 1 ws1.ds9a.nl
Q ws1.ds9a.nl IN CNAME -1 213.244.168.210
END
Q wd1.ds9a.nl IN A -1 213.244.168.210
DATA ws1.ds9a.nl IN A 3600 1 1.2.3.4
DATA ws1.ds9a.nl IN A 3600 1 1.2.3.5
DATA ws1.ds9a.nl IN A 3600 1 1.2.3.6
END
```

This would correspond to a remote webserver 213.244.168.210 wanting to resolve the IP address of www.ds9a.nl, and PowerDNS traversing the CNAMEs to find the IP addresses of ws1.ds9a.nl Another dialogue might be:

```
Q ds9a.nl IN SOA -1 213.244.168.210
DATA ds9a.nl IN SOA 86400 1 ahu.ds9a.nl ...
END
AXFR 1
DATA ds9a.nl IN SOA 86400 1 ahu.ds9a.nl ...
DATA ds9a.nl IN NS 86400 1 ns1.ds9a.nl
DATA ds9a.nl IN NS 86400 1 ns2.ds9a.nl
DATA ns1.ds9a.nl IN A 86400 1 213.244.168.210
DATA ns2.ds9a.nl IN A 86400 1 63.123.33.135
.
.
END
```

This is a typical zone transfer.

A.1.4. Sample perl backend

```
#!/usr/bin/perl -w
# sample PowerDNS Coprocess backend
#

use strict;


$|=1; # no buffering

my $line=<>;
chomp($line);

unless($line eq "HELO\t1") {
print "FAIL\n";
print STDERR "Recevied '$line'\n";
<<>>;;
exit;
}
print "OK Sample backend firing up\n"; # print our banner

while(<>)
{
# print STDERR "$$ Received: $_";
chomp();
my @arr=split(/\t/);
if(@arr<6) {
print "LOG PowerDNS sent unparseable line\n";
print "FAIL\n";
next;
}

my ($type,$qname,$qclass,$qtype,$id,$ip)=split(/\t/);

if($qtype eq "A" && $qname eq "webserver.example.com") {
# print STDERR "$$ Sent A records\n";
print "DATA $qname $qclass $qtype 3600 -1 1.2.3.4\n";
print "DATA $qname $qclass $qtype 3600 -1 1.2.3.5\n";
print "DATA $qname $qclass $qtype 3600 -1 1.2.3.6\n";
}
elsif($qtype eq "CNAME" && $qname eq "www.example.com") {
# print STDERR "$$ Sent CNAME records\n";
print "DATA $qname $qclass CNAME 3600 -1 webserver.example.com\n";
}
elsif($qtype eq "MBOXFW") {
# print STDERR "$$ Sent MBOXFW records\n";
print "DATA $qname $qclass MBOXFW 3600 -1 powerdns\@example.com\n";
```

```
}


# print STDERR "$$ End of data\n";
print "END\n";
}
```

A.2. MySQL backend

The MySQL Backend as present in PDNS is fixed - it requires a certain database schema to function. This schema corresponds to this create statement:

```
         CREATE TABLE records (
  id int(11) NOT NULL auto_increment,
  domain_id int(11) default NULL,
  name varchar(255) default NULL,
  type varchar(6) default NULL,
  content varchar(255) default NULL,
  ttl int(11) default NULL,
  prio int(11) default NULL,
  change_date int(11) default NULL,
  PRIMARY KEY (id),
  KEY name_index(name),
  KEY nametype_index(name,type),
  KEY domainid_index(domain_id)
  );
```

Every domain should have a unique domain_id, which should remain identical for all records in a domain. Records with a domain_id that differs from that in the domain SOA record will not appear in a zone transfer.

The change_date may optionally be updated to the time_t (the number of seconds since midnight UTC at the start of 1970), and is in that case used to auto calculate the SOA serial number in case that is unspecified.

A.2.1. Configuration settings

WARNING! Make sure that you can actually resolve the hostname of your database without accessing the database! It is advised to supply an IP address here to prevent chicken/egg problems!

mysql-dbname

: Database name to connect to

mysql-host

: Database host to connect to

mysql-password

: Password to connect with

mysql-socket

: MySQL socket to use for connecting

mysql-user

: MySQL user to connect as

A.2.2. Notes

It has been observed that InnoDB tables outperform the default MyISAM tables by a large margin. Furthermore, the default number of backends (3) should be raised to 10 or 15 for busy servers.

A.3. Generic MySQL backend

MySQL backend with easily configurable SQL statements, allowing you to graft PDNS on any MySQL database of your choosing. FIXME: write more

A.4. Generic PgSQL backend

PostgreSQL backend with easily configurable SQL statements, allowing you to graft PDNS on any PostgreSQL database of your choosing. Because all database schemas will be different, a generic backend is needed.

To configure this backend, 9 SQL queries need to be specified which are all very similar. 4 queries are needed for regular lookups, 4 for 'fancy records' which are disabled by default, 1 is needed for zone transfers.

The 4+4 regular queries must return the following 6 fields, in this exact order:

content

This is the 'right hand side' of a DNS record. For an A record, this is the IP address for example.

ttl

TTL of this record, in seconds. Must be a real value, no checking is performed.

prio

For MX records, this should be the priority of the mail exchanger specified.

qtype

The ASCII representation of the qtype of this record. Examples are 'A', 'MX', 'SOA', 'AAAA'. Make sure that this field returns an exact answer - PDNS won't recognise 'A ' as 'A'. This can be achieved by using a VARCHAR instead of a CHAR.

domain_id

Each domain must have a unique domain_id. No two domains may share a domain_id, all records in a domain should have the same. A number.

name

Actual name of a record. Must not end in a '.' and be fully qualified - it is not relative to the name of the domain!

Please note that the names of the fields are not relevant, but the order is!

As said earlier, there are 8 SQL queries for regular lookups. If so called 'MBOXFW' fancy records are not used, four remain:

basic-query

Default: **select content,ttl,prio,type,domain_id,name from records where qtype='%s' and name='%s'** This is the most used query, needed for doing 1:1 lookups of qtype/name values. First %s is replaced by the ASCII representation of the qtype of the question, the second by the name.

id-query

Default: **select content,ttl,prio,type,domain_id,name from records where qtype='%s' and name='%s' and id=%d** Used for doing lookups within a domain. First %s is replaced by the qtype, the %d which should appear after the %s by the numeric domain_id.

any-query

For doing ANY queries. Also used internally. Default: **select content,ttl,prio,type,domain_id,name from records where name='%s'** The %s is replaced by the qname of the question.

any-id-query

For doing ANY queries within a domain. Also used internally. Default: **select content,ttl,prio,type,domain_id,name from records where name='%s' and domain_id=%d** The %s is replaced by the name of the domain, the %d by the numerical domain id.

If PDNS is used with so called 'Fancy Records', the 'MBOXFW' record exists which specifies an email address forwarding instruction, wildcard queries are sometimes needed. This is not enabled by default. A wildcard query is an internal concept - it has no relation to *.domain-type lookups. You can safely leave these queries blank.

wildcard-query

Can be left blank. See above for an explanation. Default: **select content,ttl,prio,type,domain_id,name from records where qtype='%s' and name like '%s'**

wildcard-id-query

Can be left blank. See above for an explanation. Default: **select content,ttl,prio,type,domain_id,name from records where qtype='%s' and name like '%s' and domain_id=%d** Used for doing lookups within a domain.

wildcard-any-query

For doing wildcard ANY queries. Default: **select content,ttl,prio,type,domain_id,name from records where name like '%s'**

wildcard-any-id-query

For doing wildcard ANY queries within a domain. Default: **select content,ttl,prio,type,domain_id,name from records where name like '%s' and domain_id=%d**

The last query is for listing the entire contents of a zone. This is needed when performing a zone transfer, but sometimes also internally:

list-query

To list an entire zone. Default: **select content,ttl,prio,type,domain_id,name from records where domain_id=%d**

The template queries are expanded using the C function 'snprintf' which implies that substitutions are performed on the basis of %-place holders. To place a a % in a query which will not be substituted, use %%.

The default queries correspond to the following database schema:

```
CREATE TABLE records (
        id              SERIAL PRIMARY KEY,
        domain_id       INT DEFAULT NULL,
        name            VARCHAR(255) DEFAULT NULL,
        type            VARCHAR(6) DEFAULT NULL,
        content         VARCHAR(255) DEFAULT NULL,
        ttl             INT DEFAULT NULL,
        prio            INT DEFAULT NULL,
        change_data     INT DEFAULT NULL);

CREATE INDEX name_index ON records(name);
```

```
CREATE INDEX nametype_index ON records(name,type);
CREATE INDEX domain_id ON records(domain_id);
GRANT SELECT ON records TO pdns;
GRANT SELECT ON records_id_seq TO pdns;
```

A.4.1. Settings

The queries above are specified in pdns.conf. For example, the basic-query would appear as:

```
gpgsql-basic-query=select content,ttl,prio,type,domain_id,name from records where qtype
```

Queries can span multiple lines, like this:

```
gpgsql-basic-query=select content,ttl,prio,type,domain_id,name from records \
where qtype='%s' and name='%s'
```

Do not wrap statements in quotes as this will not work. Besides the query related settings, the following configuration options are available:

gpgsql-dbname

Database name to connect to

gpgsql-host

Database host to connect to. WARNING: When specified as a hostname a chicken/egg situation might arise where the database is needed to resolve the IP address of the database. It is best to supply an IP address of the database here.

gpgsql-password

Password to connect with

gpgsql-user

PgSQL user to connect as

A.5. Generic Oracle backend

Oracle backend with easily configurable SQL statements, allowing you to graft PDNS on any Oracle database of your choosing. FIXME: write more

A.6. Bind zone file backend

The BindBackend started life as a demonstration of the versatility of PDNS but quickly gained in importance when there appeared to be demand for a Bind 'workalike'.

The BindBackend parses a Bind-style named.conf and extracts information about zones from it. It makes no attempt to honour other configuration flags, which you should configure (when available) using the PDNS native configuration.

--help=bind

Outputs all known parameters related to the bindbackend

bind-example-zones

Loads the 'example.com' zone which can be queried to determine if PowerDNS is functioning without configuring database backends.

bind-config=

Location of the Bind configuration file to parse.

bind-check-interval=

How often to check for zone changes. See 'Operation' section.

bind-enable-huffman

Enable Huffman compression on zone data. Currently saves around 20% of memory actually used, but slows down operation somewhat.

A.6.1. Operation

On launch, the BindBackend first parses the named.conf to determine which zones need to be loaded. These will then be parsed and made available for serving, as they are parsed. So a named.conf with 100.000 zones may take 20 seconds to load, but after 10 seconds, 50.000 zones will already be available. While a domain is being loaded, it is not yet available, to prevent incomplete answers.

Reloading is currently done only when a request for a zone comes in, and then only after **bind-check-interval** seconds have passed after the last check. If a change occurred, access to the zone is disabled, the file is reloaded, access is restored, and the question is answered. For regular zones, reloading is fast enough to answer the question which lead to the reload within the DNS timeout.

If **bind-check-interval** is specified as zero, no checks will be performed.

A.6.2. Performance

The BindBackend does not benefit from the packet cache as it is fast enough on its own. Furthermore, on most systems, there will be no benefit in using multiple CPUs for the packetcache, so a noticeable speedup can be attained by specifying **distributor-threads=1** in `pdns.conf`.

A.6.3. Master/slave configuration

Currently disabled in prereleases. But see Chapter 12.

Appendix B. PDNS internals

PDNS is normally launched by the init.d script but is actually a binary called `pdns_server`. This file is started by the **start** and **monitor** commands to the init.d script. Other commands are implemented using the controlsocket.

B.1. Controlsocket

The controlsocket is the means to contact a running PDNS daemon, or as we now know, a running `pdns_server`. Over this sockets, instructions can be sent using the `pdns_control` program. Like the `pdns_server`, this program is normally accessed via the init.d script.

B.2. Guardian

When launched by the init.d script, `pdns_server` wraps itself inside a 'guardian'. This guardian monitors the performance of the inner `pdns_server` instance which shows up in the process list of your OS as `pdns_server-instance`. It is also this guardian that `pdns_control` talks to. A **STOP** is interpreted by the guardian, which causes the guardian to sever the connection to the inner process and terminate it, after which it terminates itself. The init.d script **DUMP** and **SHOW** commands need to access the inner process, because the guardian itself does not run a nameserver. For this purpose, the guardian passes controlsocket requests to the control console of the inner process. This is the same console as seen with init.d **MONITOR**.

B.3. Modules & Backends

PDNS has the concept of backends and modules. Non-static PDNS distributions have the ability to load new modules at runtime, while the static versions come with a number of modules built in, but cannot load more.

Related parameters are:

--help

Outputs all known parameters, including those of launched backends, see below.

--launch=backend,backend1,backend1:name

Launches backends. In its most simple form, supply all backends that need to be launched. If you find that you need to launch single backends multiple times, you can specify a name for later instantiations. In this case, there are 2 instances of backend1, and the second one is called 'name'. This means that **--backend1-setting** is available to configure the first or main instance, and **--backend1-name-setting** for the second one.

--load-modules=/directory/libyourbackend.so

If backends are available in nonstandard directories, specify their location here. Multiple files can be loaded if separated by commas. Only available in non-static PDNS distributions.

--list-modules

Will list all available modules, both compiled in and in dynamically loadable modules.

To run on the commandline, use the **pdns_server** binary. For example, to see options for the gpgsql backend, use the following:

```
$ /usr/sbin/pdns_server --launch=gpgsql --help=gpgsql
```

B.4. How PDNS translates DNS queries into backend queries

A DNS query is not a straightforward lookup. Many DNS queries need to check the backend for additional data, for example to determine of an unfound record should lead to an NXDOMAIN ('we know about this domain, but that record does not exist') or an unauthoritative response.

Simplified, without CNAME processing and wildcards, the algorithm is like this:

When a query for a **qname**/**qtype** tuple comes in, it is requested directly from the backend. If present, PDNS adds the contents of the reply to the list of records to return. A question tuple may generate multiple answer records.

Each of these records is now investigated to see if it needs 'additional processing'. This holds for example for MX records which may point to hosts for which the PDNS backends also contain data. This involves further lookups for A or AAAA records.

After all additional processing has been performed, PDNS sieves out all double records which may well have appeared. The resulting set of records is added to the answer packet, and sent out.

A zone transfer works by looking up the **domain_id** of the SOA record of the name and then listing all records of that **domain_id**. This is why all records in a domain need to have the same domain_id.

When a query comes in for an unknown domain, PDNS starts looking for SOA records of all subdomains of the qname, so no.such.powerdns.com turns into a SOA query for no.such.powerdns.com, such.powerdns.com, powerdns.com, com, ". When a SOA is found, that zone is consulted for relevant NS instructions which lead to a referral. If nothing is found within the zone, an authoritative NXDOMAIN is sent out.

If no SOA was found, an unauthoritative no-error is returned.

In reality, each query for a question tuple first involves checking for a CNAME, unless that resolution has been disabled with the **skip-cname** option.

PDNS breaks strict RFC comparability by not always checking for the presence of a SOA record first. This is unlikely to lead to problems though.