PowerDNS manual

PowerDNS BV

pdns@powerdns.com

PowerDNS manual
by

Published v1.0 $Date: 2002/05/03 22:05:42 $

> It is a book about a Spanish guy called Manual. You should read it.
> -- Dilbert

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. About this document

If you are reading this document from disk, you may want to check http://doc.powerdns.com for updates. The PDF version is available on http://doc.powerdns.com/pdf, a text file is on http://doc.powerdns.com/txt/ (http://doc.powerdns.com/txt).

1.3. Release notes

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

1.3.1. 1.99.11 Prerelease

This release is important because it is the first release which is accompanied by an Open Source Backend Development Kit, allowing external developers to write backends for PDNS. Furthermore, a few bugs have been fixed:

- Lines with only whitespace in zone files confused PDNS (thanks Henk Wevers)
- PDNS did not properly parse TTLs with symbolic sufixes in zone files, ie 2H instead of 7200 (thanks Henk Wevers)

1.3.2. 1.99.10 Prerelease

IMPORTANT: there has been a tiny license change involving free public webbased dns hosting, check out the changes before deploying!

PDNS is now feature complete, or very nearly so. Besides adding features, a lot of 'fleshing out' work is done now. There is an important performance bug fix which may have lead to disappointing benchmarks - so if you saw any of that, please try either this version or 1.99.8 which also does not have the bug.

This version has been very stable for us on multiple hosts, as was 1.99.9.

PostgreSQL users should be aware that while 1.99.10 works with the schema as presented in earlier versions, advanced features such as master or slave support will not work unless you create the new 'domains' table as well.

Bugs fixed:

- Wildcard AAAA queries sometimes received an NXDOMAIN error where they should have gotten an empty NO ERROR. Thanks to Jeroen Massar for spotting this on the .TK TLD!
- Do not disable the packetcache for 'recursion desired' packets unless a recursor was configured. Thanks to Greg Schueler for noticing this.
- A failing backend would not be reinstated. Thanks to 'Webspider' for discovering this problem with PostgreSQL connections that die after prolonged inactivity.
- Fixed loads of IPv6 transport problems. Thanks to Marco Davids and others for testing. Considered ready for production now.
- **Zone2sql** printed a debugging statement on range $GENERATE commands. Thanks to Rene van Valkenburg for spotting this.

Features:

- PDNS can now act as a master, sending out notifications in case of changes and allowing slaves to AXFR. Big rewording of replication support, domains are now either 'native', 'master' or 'slave'. See Chapter 12 for lots of details.
- **Zone2sql** in PostgreSQL mode now populates the 'domains' table for easy master, slave or native replication support.
- Ability to disable those annoying Windows DNS Dynamic Update messages from appearing in the log. See `log-failed-updates` in Chapter 14.
- Ability to run on IPv6 transport only
- Logging can now happen under a 'facility' so all PDNS messages appear in their own file. See Section 6.3.
- Different OS releases of PDNS now get different install path defaults. Thanks to Mark Lastdrager for nagging about this and to Nero Imhard and Frederique Rijsdijk for suggesting saner defaults.
- Infrastructure for 'also-notify' statements added.

1.3.3. 1.99.9 Early Access Prerelease

This is again a feature and an infrastructure release. We are nearly feature complete and will soon start work on the backends to make sure that they are all master, slave and 'superslave' capable.

Bugs fixed:

- PDNS sometimes sent out duplicate replies for packets passed to the recursing backend. Mostly a problem on SMP systems. Thanks to Mike Benoit for noticing this.
- Out-of-bailiwick CNAMES (ie, a CNAME to a domain not in PDNS) caused a 'ServFail' packet in 1.99.8, indicating failure, leading to hosts not resolving. Thanks to Martin Gillstrom for noticing this.
- Zone2sql balked at zones editted under operating sytems terminating files with ^Z (Windows). Thanks Brian Willcott for reporting this.
- PostgreSQL backend logged the password used to connect. Now only does so in case of failure to connect. Thanks to 'Webspider' for noticing this.
- Debian unstable distribution wrongly depended on home compiled PostgreSQL libraries. Thanks to Konrad Wojas for noticing this.

Features:

- When operating as a slave, AAAA records are now supported in the zone. They were already supported in master zones.
- IPv6 transport support - PDNS can now listen on an IPv6 socket using the **local-ipv6** setting.

- Very silly randombackend added which appears in the documentation as a sample backend. See Appendix C.
- When transferring a slave zone from a master, out of zone data is now rejected. Malicious operators might try to insert bad records otherwise.
- 'Supermaster' support for automatic provisioning from masters. See Section 12.2.1.
- Recursing backend can now live on a non-standard (!=53) port. See Chapter 11.
- Slave zone retrieval is now queued instead of immediate, which scales better and is more resilient to temporary failures.
- **max-queue-length** parameter. If this many packets are queued for database attention, consider the situation hopeless and respawn.

Internal:

- SOA records are now 'special' and each backend can optionally generate them in special ways. PostgreSQL backend does so when operating as a slave.
- Writing backends is now a lot easier. See Appendix C.
- Added Bindbackend to internal regression tests, confirming that it is compliant.

1.3.4. 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 Section 12.2
- All record types now documented, see Chapter 16.

1.3.4.1. Known bugs

Wildcard CNAMES do not work as they do with bind.

Recursion sometimes sends out duplicate packets (fixed in 1.99.9 snapshots)

Some stability issues which are caught by the guardian

1.3.4.2. 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.

1.3.5. 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.

1.3.5.1. Known bugs

Wildcard CNAMES do not work as they do with bind.

1.3.5.2. 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.

1.3.6. 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.

1.3.6.1. Known bugs

FreeBSD version does not stop when requested to do so.

Wildcard CNAMES do not work as they do with bind.

1.3.6.2. 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.

1.3.7. 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.

1.3.7.1. Known bugs

FreeBSD version does not stop when requested to do so.

Wildcard CNAMES do not work as they do with bind.

1.3.7.2. 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.

1.3.8. 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.

1.3.8.1. 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.

1.3.8.2. 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.

1.3.9. 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).

1.3.9.1. Known bugs

FreeBSD version does not stop when requested to do so.

BindBackend cannot parse zones with $GENERATE statements.

1.3.9.2. 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.

1.3.10. 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.

1.3.10.1. 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.

1.3.10.2. 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.

1.3.11. 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.

1.3.11.1. Known bugs

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

1.3.11.2. 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.

1.4. Acknowledgements

PowerDNS is grateful for the help of the following people or institutions:

- Dave Aaldering
- Wichert Akkerman
- Antony Antony
- Mike Benoit (NetNation Communication Inc.)
- Peter van Dijk
- Koos van den Hout
- Andre Koopal
- Eric Veldhuyzen
- Paul Wouters
- Thomas Wouters
- IETF Namedroppers mailinglist

Thanks!

(these people don’t share the blame for any errors or mistakes in powerdns - those are all ours)

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.

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 6.

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. Logging & 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. Furthermore, PDNS can perform a configurable amount of operational logging. This chapter also explains how to configure syslog for best results.

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
```

6.3. Operational logging using syslog

(**logging-facility** is available from 1.99.10 and onwards)

This chapter assumes familiarity with syslog, the unix logging device. PDNS logs messages with different levels. The more urgent the message, the lower the ’priority’. By default, PDNS will only log messages with an urgency of 3 or lower, but this can be changed using the **loglevel** setting in the configuration file. Setting it to 0 will eliminate all logging, 9 will log everything.

By default, logging is performed under the 'DAEMON' facility which is shared with lots of other programs. If you regard nameserving as important, you may want to have it under a dedicated facility so PDNS can log to its own files, and not clutter generic files.

For this purpose, syslog knows about 'local' facilities, numbered from LOCAL0 to LOCAL7. To move PDNS logging to LOCAL0, add **logging-facility=0** to your configuration.

Furthermore, you may want to have separate files for the differing prioties - preventing lower priority messages from obscuring important ones.

A sample syslog.conf might be:

```
local0.info                     -/var/log/pdns.info
local0.warn                     -/var/log/pdns.warn
local0.err                      /var/log/pdns.err
```

Where local0.err would store the really important messages. For performance and diskspace reasons, it is advised to audit your syslog.conf for statements also logging PDNS activities. Many syslog.confs have a '*.*' statement to /var/log/syslog, which you may want to remove.

For performance reasons, be especially certain that no large amounts of synchronous logging take place. Under Linux, this is indicated by filenames not starting with a '-' - indicating a synchronous log, which hurts performance.

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 with all backends

Only the PostgreSQL backend has, of version 1.99.9, the ability to act as a slave.

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 which 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.

It is also possible to use a resolver living on a different port. To do so, specify a recursor like this: **recursor=130.161.180.1:5300**.

Chapter 12. Master/Slave operation & replication

PDNS offers full master and slave semantics for replicating domain information. Furthermore, PDNS can benefit from native database replication.

12.1. Native replication

Native replication is the default, unless other operation is specifically configured. Native replication basically means that PDNS will not send out DNS update notifications, nor will react to them. PDNS assumes that the backend is taking care of replication unaided.

MySQL replication has proven to be very robust and well suited, even over transatlantic connections between badly peering ISPs. Other PDNS users employ Oracle replication which also works very well.

To use native replication, configure your backend storage to do the replication and do not configure PDNS to do so.

12.2. Slave operation

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 serial number there is higher, the domain is retrieved and inserted into the database. In any case, after the check the domain is declared 'fresh', 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.

12.2.1. Supermaster automatic provisioning of slaves

PDNS can recognize so called 'supermasters'. A supermaster is a host which is master for domains and for which we are to be a slave. When a master (re)loads a domain, it sends out a notification to its slaves. Normally, such a notification is only accepted if PDNS already knows that it is a slave for a domain.

However, a notification from a supermaster carries more persuasion. When PDNS determines that a notification comes from a supermaster and it is is bonafide, PDNS can provision the domain automatically, and configure itself as a slave for that zone.

To enable this feature, a backend needs to know about the IP address of the supermaster, and how PDNS will be listed in the set of NS records remotely, and the 'account' name of your supermaster. There is no need to fill this out but it does help keep track of where a domain comes from.

12.3. Master operation

When operating as a master, PDNS sends out notifications of changes to slaves, which react to these notifications by querying PDNS to see if the zone changed, and transferring its contents if it has. Notifications are a way to promptly propagate zone changes to slaves, as described in RFC 1996.

Left open by RFC 1996 is who is to be notified - which is harder to figure out than it sounds. All slaves for this domain must receive a notification but the nameserver only knows the names of the slaves - not the IP addresses, which is where the problem lies. The nameserver itself might be authoritative for the name of its secondary, but not have the data available.

To resolve this issue, PDNS tries multiple tactics to figure out the IP addresses of the slaves, and notifies everybody. In contrived configurations this may lead to duplicate notifications being sent out, which shouldn't hurt.

Some backends may be able to detect zone changes, others may chose to let the operator indicate which zones have changed and which haven't. Consult the documentation for your backend to see how it processes changes in zones.

To help deal with slaves that may have missed notifications, or have failed to respond to them, several override commands are available via the pdns_control tool (Section B.1.1):

pdns_control notify **domain**

This instructs PDNS to notify all IP addresses it considers to be slaves of this domain.

pdns_control notify-host **domain ip-address**

This is truly an override and sends a notification to an arbitrary IP address. Can be used in 'also-notify' situations or when PDNS has trouble figuring out who to notify - which may happen in contrived configurations.

Chapter 13. Fancy records for seamless email and URL integration

PDNS also supports so called 'fancy' records. A Fancy Record is actually not a DNS record, but it is translated into one. Currently, two fancy records are implemented, but not very useful without additional unreleased software. For completeness, they are listed here. The software will become available later on and is part of the Express and PowerMail suite of programs.

These records imply extra database lookups which has a performance impact. Therefore fancy records are only queried for if they are enabled with the **fancy-records** command in `pdns.conf`.

MBOXFW

This record denotes an email forward. A typical entry looks like this:

```
support@yourdomain.com      MBOXFW        you@yourcompany.com
```

When PDNS encounters a request for an MX record for yourdomain.com it will, if fancy records are enabled, also check for the existence of an MBOXFW record ending on '@yourdomain.com', in which case it will hand out a record containing the configured **smtpredirector**. This server should then also be able to access the PDNS database to figure out where mail to support@yourdomain.com should go to.

URL

URL records work in much the same way, but for HTTP. A sample record:

```
yourdomain.com      URL        http://somewhere.else.com/yourdomain
```

A URL record is converted into an A record containing the IP address configured with the **urlredirector** setting. On that IP address a webserver should live that knows how to redirect yourdomain.com to http://somewhere.else.com/yourdomain.

Chapter 14. 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. See Chapter 13.

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. You can specify multiple addresses separated by commas or whitespace.

local-port=...

The port on which we listen. Only one port possible.

log-failed-updates=...

If set to 'no', failed Windows Dynamic Updates will not be logged.

logging-facility=...

If set to a a digit, logging is performed under this LOCAL facility. See Section 6.3. Available from 1.99.9 and onwards.

loglevel=...

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

max-queue-length=...

If this many packets are waiting for database attention, consider the situation hopeless and respawn.

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. See Chapter 13.

socket-dir=...

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

urlredirector=...

Where we send hosts to that need to be url redirected. See Chapter 13.

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 15. Index of all internal metrics

15.1. 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

15.1.1. 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)

> 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 16. 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 16-1. 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
```

'content' is as specified in Chapter 16. 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

Table A-1. MySQL backend capabilities

Native	Yes
Master	No
Slave	No
Superslave	No
Autoserial	Yes

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.

Not included as of PDNS 1.99.10.

A.4. Generic PgSQL backend

Table A-2. Generic PgSQL backend capabilities

Native	Yes - but PostgreSQL does not replicate
Master	Yes
Slave	Yes
Superslave	Yes
Autoserial	Yes

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 cover all needs.

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 setup conforms to the following schema:

```
create table domains (
 id  SERIAL PRIMARY KEY,
 name  VARCHAR(255) NOT NULL,
 master  VARCHAR(20) DEFAULT NULL,
 last_check  INT DEFAULT NULL,
 type  VARCHAR(6) NOT NULL,
 notified_serial INT DEFAULT NULL,
 account         VARCHAR(40) DEFAULT NULL
);
CREATE UNIQUE INDEX name_index ON domains(name);

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);

create table supermasters (
  ip VARCHAR(25) NOT NULL,
  nameserver VARCHAR(255) NOT NULL,
  account VARCHAR(40) DEFAULT NULL
);

GRANT SELECT ON supermasters TO pdns;
GRANT ALL ON domains TO pdns;
GRANT ALL ON domains_id_seq TO pdns;
GRANT ALL ON records TO pdns;
GRANT ALL ON records_id_seq TO pdns;
```

This schema contains all elements needed for master, slave and superslave operation. Depending on which features will be used, the 'GRANT' statements can be trimmed to make sure PDNS cannot subvert the contents of your database.

Zone2sql with the --gpgsql flag also assumes this layout is in place.

A.4.1. Basic functionality

4 queries are needed for regular lookups, 4 for 'fancy records' which are disabled by default and 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.

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**

A.4.2. Fancy records

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**

A.4.3. Settings and specifying queries

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.4.4. Native operation

For native operation, either drop the FOREIGN KEY on the domain_id field, or (recommended), make sure the **domains** table is filled properly. To add a domain, issue the following:

```
insert into domains (name,type) values ('powerdns.com','NATIVE');
```

The records table can now be filled by with the domain_id set to the id of the domains table row just inserted.

A.4.5. Slave operation

The PostgreSQL backend is fully slave capable. 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 a while for PDNS to pick up the addition - which happens within one minute. There is no need to inform PDNS that a new domain was added. 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] Connected to database
Apr 09 13:35:30 AXFR started for 'powerdns.com'
Apr 09 13:35:30 AXFR done for 'powerdns.com'
Apr 09 13:35:30 [gPgSQLBackend] Closing connection
```

From now on, PDNS is authoritative for the 'powerdns.com' zone and will respond accordingly for queries within that zone.

A.4.6. Superslave operation

To configure a supermaster with IP address 10.0.0.11 which lists this installation as 'autoslave.powerdns.com', issue the following:

```
insert into supermasters ('10.0.0.11','autoslave.powerdns.com','internal');
```

From now on, valid notifies from 10.0.0.11 that list a NS record containing 'autoslave.powerdns.com' will lead to the provisioning of a slave domain under the account 'internal'. See Section 12.2.1 for details.

A.4.7. Master operation

The PostgreSQL backend is fully master capable with automatic discovery of serial changes. Raising the serial number of a domain suffices to trigger PDNS to send out notifications. To configure a domain for master operation instead of the default native replication, issue:

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

Make sure that the assigned id in the domains table matches the domain_id field in the records table!

A.5. Generic Oracle backend

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

PowerDNS is currently ascertaining if this backend can be distributed in binary form without violating Oracle licensing.

A.6. Bind zone file backend

Table A-3. Bind zone file backend capabilities

Native	Yes
Master	No
Slave	Yes

Superslave	No
Autoserial	No

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 Section 12.2.

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.1.1. pdns_control

To communicate with PDNS over the controlsocket, the **pdns_control** command is used. The init.d script also calls pdns_control. The syntax is simple: **pdns_control command arguments**. Currently this is most useful for telling backends to rediscover domains or to force the transmission of notifications. See Section 12.3.

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.

Appendix C. Backend writers' guide

PDNS backends are implemented via a simple yet powerful C++ interface. If your needs are not met by the PipeBackend, you may want to write your own. Doing so requires a copy of the PowerDNS Open Source Backend Development kit which can be found on http://downloads.powerdns.com/bdk.

A backend contains zero DNS logic. It need not look for CNAMES, it need not return NS records unless explicitly asked for, etcetera. All DNS logic is contained within PDNS itself - backends should simply return records matching the description asked for.

C.1. Simple read-only native backends

Implementing a backend consists of inheriting from the DNSBackend class. For read-only backends, which do not support slave operation, only the following methods are relevant:

```
class DNSBackend
{
public:

virtual bool lookup(const QType &qtype, const string &qdomain, DNSPacket *pkt_p=0, int zone
virtual bool list(int domain_id)=0;
virtual bool get(DNSResourceRecord &r)=0;
virtual bool getSOA(const string &name, SOAData &soadata);
};
```

Note that the first three methods must be implemented. `getSOA()` has a useful default implementation.

The semantics are simple. Each instance of your class only handles one (1) query at a time. There is no need for locking as PDNS guarantees that your backend will never be called reentrantly.

Some examples, a more formal specification is down below. A normal lookup starts like this:

```
        YourBackend yb;
yb.lookup(QType::CNAME,"www.powerdns.com");
```

Your class should now do everything to start this query. Perform as much preparation as possible - handling errors at this stage is better for PDNS than doing so later on. A real error should be reported by throwing an exception.

PDNS will then call the `get()` method to get **DNSResourceRecord**s back. The following code illustrates a typical query:

```
yb.lookup(QType::CNAME,"www.powerdns.com");

DNSResourceRecord rr;
```

```
while(yb.get(rr))
      cout<<"Found cname pointing to '"+rr.content+"'"<<endl;
}
```

Each zone starts with a Start of Authority (SOA) record. This record is special so many backends will choose to implement it specially. The default `getSOA()` method performs a regular lookup on your backend to figure out the SOA, so if you have no special treatment for SOA records, where is no need to implement your own `getSOA()`.

Besides direct queries, PDNS also needs to be able to list a zone, to do zone transfers for example. Each zone has an id which should be unique within the backend. To list all records belonging to a zone id, the `list()` method is used. Conveniently, the domain_id is also available in the **SOAData** structure.

The following lists the contents of a zone called "powerdns.com".

```
SOAData sd;
if(!yb.getSOA("powerdns.com",sd))  // are we authoritative over powerdns.com?
  return RCode::NotAuth;           // no

yb.list(sd.domain_id);
while(yb.get(rr))
      cout<<rr.qname<<"\t IN "<<rr.qtype.getName()<<"\t"<<rr.content<<endl;
```

Please note that when so called 'fancy records' (see Chapter 13) are enabled, a backend can receive wildcard lookups. These have a % as the first character of the qdomain in lookup.

C.1.1. A sample minimal backend

This backend only knows about the host "random.powerdns.com", and furthermore, only about its A record:

```
/* FIRST PART */
class RandomBackend : public DNSBackend
{
public:
  bool list(int id) {
    return false; // we don't support AXFR
  }

  void lookup(const QType &type, const string &qdomain, DNSPacket *p, int zoneId)
  {
    if(type.getCode()!=QType::A || qdomain!="random.powerdns.com")  // we only know about r
      d_answer="";                                                  // no answer
    else {
```

```
      ostringstream os;
      os<<random()%256<<"."<<random()%256<<"."<<random()%256<<"."<<random()%256;
      d_answer=os.str();                                          // our random ip address
    }
  }

  bool get(DNSResourceRecord &rr)
  {
    if(!d_answer.empty()) {
      rr.qname="random.powerdns.com";                             // fill in details
      rr.qtype=QType::A;                                          // A record
      rr.ttl=86400;                                               // 1 day
      rr.content=d_answer;

      d_answer="";                                                // this was the last an

      return true;
    }
    return false;                                                 // no more data
  }

private:
  string d_answer;
};

/* SECOND PART */

class RandomFactory : public BackendFactory
{
public:
  RandomFactory() : BackendFactory("random") {}

  DNSBackend *make(const string &suffix)
  {
    return new RandomBackend();
  }
};

/* THIRD PART */

class Loader
{
public:
  Loader()
  {
    BackendMakers().report(new RandomFactory);

    L<<Logger::Info<<" [RandomBackend] This is the randombackend ("__DATE__", "__TIME__")
  }
};

static Loader loader;
```

This simple backend can be used as an 'overlay'. In other words, it only knows about a single record, another loaded backend would have to know about the SOA and NS records and such. But nothing prevents us from loading it without another backend.

The first part of the code contains the actual logic and should be pretty straightforward. The second part is a boilerplate 'factory' class which PDNS calls to create randombackend instances. Note that a 'suffix' parameter is passed. Real life backends also declare parameters for the configuration file; these get the 'suffix' appended to them. Note that the "random" in the constructor denotes the name by which the backend will be known.

The third part registers the RandomFactory with PDNS. This is a simple C++ trick which makes sure that this function is called on execution of the binary or when loading the dynamic module.

Please note that the RandomBackend is actually in most PDNS releases. By default it lives on random.example.com, but you can change that by setting **random-hostname**.

C.1.2. Interface definition

Classes:

Table C-1. DNSResourceRecord class

QType qtype	QType of this record
string qname	name of this record
string content	ASCII representation of right hand side
u_int16_t priority	priority of an MX record.
u_int32_t ttl	Time To Live of this record
int domain_id	ID of the domain this record belongs to
time_t last_modified	If unzero, last time_t this record was changed

Table C-2. SOAData struct

string nameserver	Name of the master nameserver of this zone
string hostmaster	Hostmaster of this domain. May contain an @
u_int32_t serial	Serial number of this zone
u_int32_t refresh	How often this zone should be refreshed
u_int32_t retry	How often a failed zone pull should be retried.
u_int32_t expire	If zone pulls failed for this long, retire records
u_int32_t default_ttl	Difficult

int domain_id	The ID of the domain within this backend. Must be filled!
DNSBackend *db	Pointer to the backend that feels authoritative for a domain and can act as a slave

Methods:

void lookup(const QType &qtype, const string &qdomain, DNSPacket *pkt=0, int zoneId=-1)

This function is used to initiate a straight lookup for a record of name 'qdomain' and type 'qtype'. A QType can be converted into an integer by invoking its `getCode()` method and into a string with the `getCode()`.

The original question may or may not be passed in the pointer p. If it is, you can retrieve (from 1.99.11 onwards) information about who asked the question with the `getRemote(DNSPacket *)` method. Alternatively, `bool getRemote(struct sockaddr *sa, socklen_t *len)` is available.

Finally, the domain_id might also be passed indicating that only answers from the indicated zone need apply. This can both be used as a restriction or as a possible speedup, hinting your backend where the answer might be found.

If initiated succesfully, as indicated by returning **true**, answers should be made available over the `get()` method.

Should throw an AhuException if an error occured accessing the database. Returning otherwise indicates that the query was started succesfully. If it is known that no data is available, no exception should be thrown! An exception indicates that the backend considers itself broken - not that no answers are available for a question.

It is legal to return here, and have the first call to `get()` return false. This is interpreted as 'no data'

bool list(int domain_id)

Initiates a list of the indicated domain. Records should then be made available via the `get()` method. Need not include the SOA record. If it is, PDNS will not get confused.

Should return false if the backend does not consider itself authoritative for this zone. Should throw an AhuException if an error occured accessing the database. Returning true indicates that data is or should be available.

bool get(DNSResourceRecord &rr)

Request a DNSResourceRecord from a query started by `get()` of `list()`. If this functions returns **true**, **rr** has been filled with data. When it returns false, no more data is available, and **rr** does not contain new data. A backend should make sure that it either fills out all fields of the DNSResourceRecord or resets them to their default values.

Should throw an AhuException in case a database error occurred.

bool getSOA(const string &name, SOAData &soadata)

If the backend considers itself authoritative over domain `name`, this method should fill out the passed **SOAData** structure and return a positive number. If the backend is functioning correctly, but does not consider itself authoritative, it should return 0. In case of errors, an AhuException should be thrown.

C.2. Reporting errors

To report errors, the Logger class is available which works mostly like an iostream. Example usage is as shown above in the RandomBackend. Note that it is very important that each line is ended with **endl** as your message won't be visible otherwise.

To indicate the importance of an error, the standard syslog errorlevels are available. They can be set by outputting `Logger::Critical`, `Logger::Error`, `Logger::Warning`, `Logger::Notice`, `Logger::Info` or `Logger::Debug` to `L`, in descending order of graveness.

C.3. Declaring and reading configuration details

It is highly likely that a backend needs configuration details. On launch, these parameters need to be declared with PDNS so it knows it should accept them in the configuration file and on the commandline. Furthermore, they will be listed in the output of **--help**.

Declaring arguments is done by implementing the member function `declareArguments()` in the factory class of your backend. PDNS will call this method after launching the backend.

In the `declareArguments()` method, the function `declare()` is available. The exact definitions:

void declareArguments(const string &suffix="")

This method is called to allow a backend to register configurable parameters. The suffix is the sub-name of this module. There is no need to touch this suffix, just pass it on to the declare method.

void declare(const string &suffix, const string &param, const string &explanation, const string &value)

The suffix is passed to your method, and can be passed on to declare. **param** is the name of your parameter. **explanation** is what will appear in the output of --help. Furthermore, a default value can be supplied in the **value** parameter.

A sample implementation:

```
void declareArguments(const string &suffix)
{
    declare(suffix,"dbname","Pdns backend database name to connect to","powerdns");
  declare(suffix,"user","Pdns backend user to connect as","powerdns");
  declare(suffix,"host","Pdns backend host to connect to","");
  declare(suffix,"password","Pdns backend password to connect with","");
}
```

After the arguments have been declared, they can be accessed from your backend using the `mustDo()`, `getArg()` and `getArgAsNum()` methods. The are defined as follows in the DNSBackend class:

void setArgPrefix(const string &prefix)

Must be called before any of the other accessing functions are used. Typical usage is '`setArgPrefix("mybackend"+suffix)`' in the constructor of a backend.

bool mustDo(const string &key)

Returns true if the variable `key` is set to anything but 'no'.

const string& getArg(const string &key)

Returns the exact value of a parameter.

int getArgAsNum(const string &key)

Returns the numerical value of a parameter. Uses `atoi()` internally

Sample usage from the BindBackend, using the **bind-example-zones** and **bind-config** parameters.

```
if(mustDo("example-zones")) {
  insert(0,"www.example.com","A","1.2.3.4");
  /* ... */
}


if(!getArg("config").empty()) {
```

```
  BindParser BP;

  BP.parse(getArg("config"));
}
```

C.4. Read/write slave-capable backends

The backends above are 'natively capable' in that they contain all data relevant for a domain and do not pull in data from other nameservers. To enable storage of information, a backend must be able to do more.

Before diving into the details of the implementation some theory is in order. Slave domains are pulled from the master. PDNS needs to know for which domains it is to be a slave, and for each slave domain, what the IP address of the master is.

A slave zone is pulled from a master, after which it is 'fresh', but this is only temporary. In the SOA record of a zone there is a field which specifies the 'refresh' interval. After that interval has elapsed, the slave nameserver needs to check at the master ff the serial number there is higher than what is stored in the backend locally.

If this is the case, PDNS dubs the domain 'stale', and schedules a transfer of data from the remote. This transfer remains scheduled until the serial numbers remote and locally are identical again.

This theory is implemented by the `getUnfreshSlaveInfos` method, which is called on all backends periodically. This method fills a vector of **SlaveDomain**s with domains that are unfresh and possibly stale.

PDNS then retrieves the SOA of those domains remotely and locally and creates a list of stale domains. For each of these domains, PDNS starts a zonetransfer to resynchronise. Because zone transfers can fail, it is important that the interface to the backend allows for transaction semantics because a zone might otherwise be left in a halfway updated situation.

The following excerpt from the DNSBackend shows the relevant functions:

```
class DNSBackend {
public:
        /* ... */
        virtual bool getDomainInfo(const string &domain, DomainInfo &di);
 virtual bool isMaster(const string &name, const string &ip);
```

```
  virtual bool startTransaction(const string &qname, int id);
  virtual bool commitTransaction();
  virtual bool abortTransaction();
  virtual bool feedRecord(const DNSResourceRecord &rr);
  virtual void getUnfreshSlaveInfos(vector<DomainInfo>* domains);
  virtual void setFresh(int id);
         /* ... */
}
```

The mentioned DomainInfo struct looks like this:

Table C-3. DomainInfo struct

int id	ID of this zone within this backend
string master	IP address of the master of this domain, if any
u_int32_t serial	Serial number of this zone
u_int32_t notified_serial	Last serial number of this zone that slaves have seen
time_t last_check	Last time this zone was checked over at the master for changes
enum {Master,Slave,Native} kind	Type of zone
DNSBackend *backend	Pointer to the backend that feels authoritative for a domain and can act as a slave

These functions all have a default implementation that returns false - which explains that these methods can be omitted in simple backends. Furthermore, unlike with simple backends, a slave capable backend must make sure that the 'DNSBackend *db' field of the SOAData record is filled out correctly - it is used to determine which backend will house this zone.

bool isMaster(const string &name, const string &ip);

If a backend considers itself a slave for the domain **name** and if the IP address in **ip** is indeed a master, it should return true. False otherwise. This is a first line of checks to guard against reloading a domain unnecessarily.

void getUnfreshSlaveInfos(vector<DomainInfo>* domains)

When called, the backend should examine its list of slave domains and add any unfresh ones to the domains vector.

bool getDomainInfo(const string &name, DomainInfo & di)

This is like getUnfreshSlaveInfos, but for a specific domain. If the backend considers itself authoritative for the named zone, `di` should be filled out, and 'true' be returned. Otherwise return false.

bool startTransaction(const string &qname, int id)

When called, the backend should start a transaction that can be committed or rolled back atomically later on. In SQL terms, this function should **BEGIN** a transaction and **DELETE** all records.

bool feedRecord(const DNSResourceRecord &rr)

Insert this record.

bool commitTransaction();

Make the changes effective. In SQL terms, execute **COMMIT**.

bool abortTransaction();

Abort changes. In SQL terms, execute **ABORT**.

bool setFresh()

Indicate that a domain has either been updated or refreshed without the need for a retransfer. This causes the domain to vanish from the vector modified by `getUnfreshSlaveInfos()`.

PDNS will always call `startTransaction()` before making calls to `feedRecord()`. Although it is likely that `abortTransaction()` will be called in case of problems, backends should also be prepared to abort from their destructor.

The actual code in PDNS is currently (1.99.9):

```
    Resolver resolver;
    resolver.axfr(remote,domain.c_str());

    db->startTransaction(domain, domain_id);

    L<<Logger::Error<<"AXFR started for '"<<domain<<"'"<<endl;
    Resolver::res_t recs;

    while(resolver.axfrChunk(recs)) {
      for(Resolver::res_t::const_iterator i=recs.begin();i!=recs.end();++i) {
db->feedRecord(*i);
      }
    }
    db->commitTransaction();
    db->setFresh(domain_id);
    L<<Logger::Error<<"AXFR done for '"<<domain<<"'"<<endl;
```

C.4.1. Supermaster/Superslave capability

A backend that wants to act as a 'superslave' for a master should implement the following method:

```
class DNSBackend
{
   virtual bool superMasterBackend(const string &ip, const string &domain, cons
};
```

This function gets called with the IP address of the potential supermaster, the domain it is sending a notification for and the set of NS records for this domain at that IP address.

Using the supplied data, the backend needs to determine if this is a bonafide 'supernotification' which should be honoured. If it decides that it should, the supplied pointer to 'account' needs to be filled with the configured name of the supermaster (if accounting is desired), and the db needs to be filled with a pointer to your backend.

Supermaster/superslave is a complicated concept, if this is all unclear see Section 12.2.1.

C.5. Read/write master-capable backends

In order to be a useful master for a domain, notifies must be sent out whenever a domain is changed. Periodically, PDNS queries backends for domains that may have changed, and sends out notifications for slave nameservers.

In order to do so, PDNS calls the `getUpdatedMasters()` method. Like the `getUnfreshSlaveInfos()` function mentioned above, this should add changed domain names to the vector passed.

The following excerpt from the DNSBackend shows the relevant functions:

```
 class DNSBackend {
 public:
         /* ... */
  virtual void getUpdatedMasters(vector<DomainInfo>* domains);
  virtual void setNotifed(int id, u_int32_t serial);
         /* ... */
}
```

These functions all have a default implementation that returns false - which explains that these methods can be omitted in simple backends. Furthermore, unlike with simple backends, a slave capable backend must make sure that the 'DNSBackend *db' field of the SOAData record is filled out correctly - it is used to determine which backend will house this zone.

void getUpdatedMasters(vector<DomainInfo>* domains)

When called, the backend should examine its list of master domains and add any changed ones to the DomainInfo vector

bool setNotified(int domain_id, u_int32_t serial)

Indicate that notifications have been queued for this domain and that it need not be considered 'updated' anymore

C.6. HOWTO & Frequently Asked Questions

Writing backends without access to the full PDNS source means that you need to write code that can be loaded by PDNS at runtime. This in turn means that you need to use the same compiler that we do. For linux, thhis is currently GCC 3.0.4, although any 3.0.x compiler is probably fine. In tests, even 3.1 works.

For FreeBSD we use GCC 2.95.2.

Furthermore, your pdns_server executable must be dynamically linked. The default .rpm PDNS contains a static binary so you need to retrieve the dynamic rpm or the dynamic tar.gz or the Debian unstable ('Woody') deb. FreeBSD dynamic releases are forthcoming.

Q: Will PDNS drivers work with other PDNS versions than they were compiled for?

A: 'Probably'. We make no guarantees. Efforts have been made to keep the interface between the backend and PDNS as thin as possible. For example, a backend compiled with the 1.99.11 backend development kit works with 1.99.10. But don't count on it. We will notify when we think an incompatible API change has occured but you are best off recompiling your driver for each new PDNS release.

Q: What is in that DNSPacket * pointer passed to lookup!

A: For reasons outlined above, you should treat that pointer as opaque and only access it via the `getRemote()` functions made available and documented above. The DNSPacket class changes a lot and this level of indirection allows for greater changes to be made without changing the API to the backend coder.

Q: Can I release the backend I wrote?

A: Please do! If you tell us about it we will list you on our page.

Q: Can I sell backends I wrote?

A: You can. Again, if you tell us about them we will list your backend on the site. You can keep the source of your backend secret if you want, or you can share it with the world under any license of your chosing.

Q: Will PowerDNS use my code in the PDNS distribution?

A: If your license permits it and we like your backend, we sure will. If your license does not permit it but we like your backend anyway we may contact you.

Q: My backend compiles but when I try to load it, it says 'undefined symbol: _Z13BackendMakersv'

A: Your pdns_server binary is static and cannot load a backend driver at runtime. Get a dynamic version of pdns, or complain to pdns@powerdns.com if one isn't available. To check what kind of binary you have, execute 'file $(which pdns_server)'.

Q: I downloaded a dynamic copy of pdns_server but it doesn't run, even without my backend

A: Run 'ldd' on the pdns_server binary and figure out what libraries you are missing. Most likely you need to install gcc 3.0 libraries, RedHat 7.1 and 7.2 have packages available, Debian installs these by default if you use the 'unstable deb' of PDNS.

Q: What I want can't be done from a backend - I need the whole PDNS source

A: If you require the source, please contact us (pdns@powerdns.com). All commercial licensees receive the source, for others we may grant exceptions.

Q: What is this 'AhuException' I keep reading about?

A: This name has historical reasons and has no significance (http://ds9a.nl).

Q: I need a backend but I can't write it, can you help?

A: Yes, we also do custom development. Contact us at pdns@powerdns.com.