
GeoDNS servers

This is the DNS server powering the NTP Pool system and other similar services.

openssf best practices in progress 97%

Questions or suggestions?

For bug reports or feature requests, please create an issue. For questions or discussion, you can post to the GeoDNS category on the NTP Pool forum.

Installation

Release builds are available in a yum repository at <https://pkgs.ntppool.org/yum/> and apt (debian, ubuntu) packages at <https://pkgs.ntppool.org/apt/>.

From source

If you don't have Go installed the easiest way to build geodns from source is to download and install Go from <https://golang.org/dl/>.

GeoDNS generally requires a recent version of Go (one of the last few major versions)

```
1 git clone https://github.com/abh/geodns.git
2 cd geodns
3 go build
4 ./geodns -h
```

You can also build with goreleaser.

Sample configuration

There's a sample configuration file in [dns/example.com.json](#). This is currently derived from the [test.example.com](#) data used for unit tests and not an example of a "best practices" configuration.

For testing there's also a bigger test file at:

```
1 mkdir -p dns
2 curl -o dns/test.ntppool.org.json http://tmp.askask.com/2012/08/dns/ntppool.org.json.big
```

Run it

After building the server you can run it with:

```
./geodns -log -interface 127.1 -port 5053
```

To test the responses run

```
dig -t a test.example.com @127.1 -p 5053
```

or

```
dig -t ptr 2.1.168.192.IN-ADDR.ARPA. @127.1 -p 5053
```

or more simply put

```
dig -x 192.168.1.2 @127.1 -p 5053
```

The binary can be moved to /usr/local/bin, /opt/geodns/ or wherever you find appropriate.

Configuration

See the sample configuration file.

Notable command line parameters (and their defaults)

- -config="/dns/"

Directory of zone files (and configuration named `geodns.conf`).

- -checkconfig=false

Check configuration file, parse zone files and exit

- -interface=""

Comma separated IPs to listen on for DNS requests.

- -port="53"

Port number for DNS requests (UDP and TCP)

- -http=":8053"

Listen address for HTTP interface. Specify as `127.0.0.1:8053` to only listen on localhost.

- -identifier=""

Identifier for this instance (hostname, pop name or similar).

It can also be a comma separated list of identifiers where the first is the “server id” and subsequent ones are “group names”, for example region of the server, name of anycast cluster the server is part of, etc. This is used in (future) reporting/statistics features.

- `-log=false`

Enable to get lots of extra logging, only useful for testing and debugging. Absolutely not recommended in production unless you get very few queries (less than 1-200/second).

- `-cpus=4`

Maximum number of CPUs to use. Set to 0 to match the number of CPUs available on the system (also the default).

Logging

GeoDNS supports query logging to JSON or Avro files (see the sample configuration file for options).

Prometheus metrics

`/metrics` on the http port provides a number of metrics in Prometheus format.

Runtime status page, Websocket metrics & StatHat integration

The runtime status page, websocket feature and StatHat integration have been replaced with Prometheus metrics.

Country and continent lookups

See zone targeting options below.

Weighted records

Most records can have a ‘weight’ assigned. If any records of a particular type for a particular name have a weight, the system will return `max_hosts` records (default 2).

If the weight for all records is 0, all matching records will be returned. The weight for a label can be any integer as long as the weights for a label and record type is less than 2 billion.

As an example, if you configure

```
1 10.0.0.1, weight 10
2 10.0.0.2, weight 20
3 10.0.0.3, weight 30
4 10.0.0.4, weight 40
```

with `max_hosts` 2 then .4 will be returned about 4 times more often than .1.

Configuration file

The `geodns.conf` file allows you to specify a specific directory for the GeoIP data files and other options. See the `geodns.conf.sample` file for example configuration.

The global configuration file is not reloaded at runtime.

Most of the configuration is “per zone” and done in the zone `.json` files. The zone configuration files are automatically reloaded when they change.

Zone format

In the zone configuration file the whole zone is a big hash (associative array). At the top level you can (optionally) set some options with the keys `serial`, `ttd` and `max_hosts`.

The actual zone data (dns records) is in a hash under the key “data”. The keys in the hash are hostnames and the value for each hostname is yet another hash where the keys are record types (lower-case) and the values an array of records.

For example to setup an MX record at the zone apex and then have a different A record for users in Europe than anywhere else, use:

```
1 {
2     "serial": 1,
3     "data": {
4         "": {
5             "ns": [ "ns.example.net", "ns2.example.net" ],
6             "txt": "Example zone",
7             "spf": [ { "spf": "v=spf1 ~all", "weight": 1 } ],
8             "mx": { "mx": "mail.example.com", "preference": 10 }
9         },
10        "mail": { "a": [ ["192.168.0.1", 100], ["192.168.10.1", 50] ] },
11        "mail.europe": { "a": [ ["192.168.255.1", 0] ] },
12        "smtp": { "alias": "mail" }
13    }
14 }
```

The configuration files are automatically reloaded when they're updated. If a file can't be read (invalid JSON, for example) the previous configuration for that zone will be kept.

Zone options

- serial

GeoDNS doesn't support zone transfers (AXFR), so the serial number is only used for debugging and monitoring. The default is the 'last modified' timestamp of the zone file.

- ttl

Set the default TTL for the zone (default 120).

- targeting
- max_hosts
- contact

Set the soa 'contact' field (default is "hostmaster.\$domain").

Zone targeting options

@

country continent

region and regiongroup

Supported record types

Each label has a hash (object/associative array) of record data, the keys are the type. The supported types and their options are listed below.

Adding support for more record types is relatively straight forward, please open a ticket in the issue tracker with what you are missing.

A

Each record has the format of a short array with the first element being the IP address and the second the weight.

```
1 [ [ "192.168.0.1", 10], [ "192.168.2.1", 5] ]
```

See above for how the weights work.

AAAA

Same format as A records (except the record type is “aaaa”).

Alias

Internally resolved cname, of sorts. Only works internally in a zone.

```
1 "foo"
```

CNAME

```
1 "target.example.com."  
2 "www"
```

The target will have the current zone name appended if it’s not a FQDN (since v2.2.0).

MX

MX records support a **weight** similar to A records to indicate how often the particular record should be returned.

The **preference** is the MX record preference returned to the client.

```
1 { "mx": "foo.example.com" }  
2 { "mx": "foo.example.com", "weight": 100 }  
3 { "mx": "foo.example.com", "weight": 100, "preference": 10 }
```

weight and **preference** are optional.

NS

NS records for the label, use it on the top level empty label ("") to specify the nameservers for the domain.

```
1 [ "ns1.example.com", "ns2.example.com" ]
```

There's an alternate legacy syntax that has space for glue records (IPv4 addresses), but in GeoDNS the values in the object are ignored so the list syntax above is recommended.

```
1 { "ns1.example.net.": null, "ns2.example.net.": null }
```

TXT

Simple syntax

```
1 "Some text"
```

Or with weights

```
1 { "txt": "Some text", "weight": 10 }
```

SPF

An SPF record is semantically identical to a TXT record with the exception that the label is set to 'spf'. An example of an spf record with weights:

```
1 { "spf": "v=spf1 ~all", "weight": 1 }
```

An spf record is typically at the root of a zone, and a label can have an array of SPF records, e.g

```
1 "spf": [ { "spf": "v=spf1 ~all", "weight": 1 } , "spf": "v=spf1  
10.0.0.1", "weight": 100]
```

SRV

An SRV record has four components: the weight, priority, port and target. The keys for these are "srv_weight", "priority", "target" and "port". Note the difference between srv_weight (the weight key for the SRV qtype) and "weight".

An example srv record definition for the _sip._tcp service:

```
1 "_sip._tcp": {  
2   "srv": [ { "port": 5060, "srv_weight": 100, "priority": 10, "target  
           ": "sipserver.example.com."} ]  
3 },
```

Much like MX records, SRV records can have multiple targets, eg:

```
1  "_http_tcp": {
2      "srv": [
3          { "port": 80, "srv_weight": 10, "priority": 10, "target": "www.
4              example.com."},
5          { "port": 8080, "srv_weight": 10, "priority": 20, "target": "
6              www2.example.com."}
7      ]
8  },
```

License and Copyright

This software is Copyright 2012-2015 Ask Bjørn Hansen. For licensing information please see the file called LICENSE.