GeoDNS

# GeoDNS support for CoreDNS

Table of Contents

[1. Introduction 2](#_Toc474863440)

[2. General Overview and Design Approach 3](#_Toc474863441)

[2.1 External Interfaces and dependency: 3](#_Toc474863442)

[3. Middleware Detailed Design 4](#_Toc474863443)

[3.1 Implementing Middleware in CoreDNS 4](#_Toc474863444)

[3.2 Setup middleware 5](#_Toc474863445)

[3.2.1 Init : 6](#_Toc474863446)

[3.2.2 RegisterMiddleWare: 6](#_Toc474863447)

[3.2.3 LoadGeoipDB 6](#_Toc474863448)

[3.3 Implementing the ServeDNS: 7](#_Toc474863449)

[3.3.1 ServeDNS: 8](#_Toc474863450)

[3.3.2 GetGeoipDestination 8](#_Toc474863451)

[3.3.3 GetGeoipSource 9](#_Toc474863452)

[3.3.4 CalculateShortesDistance 9](#_Toc474863453)

[3.3.5 Distance: 9](#_Toc474863454)

List of Figures

**No table of figures entries found.**

## Introduction

The CoreDNS is a highly flexible DNS server written in golang.

It supports a number of backend (like etcd and kubernetes)

It supports a number of features including the following

Use etcd as a backend, i.e., a 101.5% replacement for SkyDNS (etcd).

Use k8s (kubernetes) as a backend (kubernetes).

It is highly flexible because of its simple approach towards adding new features via middleware. With CoreDNS now available in kubernetes, there is a need for GeoDNS support to extend the capability of the current CoreDNS in kubernetes. With GeoDNS support we can have kubernetes to route request to the nearest geographically located application/server. Also in the future we can have policies based on the geoip ( dns records ). For example: the we can have policy like all the request coming from asia continent to route traffic request to a particular location ( address/ip ), like India.

## General Overview and Design Approach

The scope of this document is to integrate the GeoDNS as a middleware in CoreDNS.

The middleware Geodns will route the request ip information to the geographically closed matching A or AAAA record in the CoreDNS.

CoreDNS has a very eligenat way to add new feature/functionality as a middleware (similar to the decorator pattern). Implementing GeoDNS as a middleware is the best design approach. The CoreDNS has layed the basic blocks on how to implement a middlewear.

The approach is basically chaining the middleware and passing the object to the next middleware. Each middleware when called apply some functionality and then passes it on to the next middleware. Because of this approach it is possible to apply middleware in specific order.

### External Interfaces and dependency:

The GeoIP information has the latitiude,longitude, city and country information. This Geoip database is freely available from MindMax. They have a subscription version of the database which is more accurate than the freely available one. But the formats of both the database are same. There are a number of libraries written to use these databases in applications. A golang version of such a library is available at ‘***github.com/abh/geoip***’ which uses the [libgeoip C library](http://www.maxmind.com/app/c). We consider this library for our current design. A pure go implementation of the geoip is available at ‘***github.com/oschwald/geoip2-golang***’.

## Middleware Detailed Design

CoreDNS defines a corefile , which configure the CoreDNS and in the corefile the middleware are specified.

For GeoDNS support we need to add a directive in the corefile for the CoreDNS to trigger our GeoDNS middleware functions and methods.

Below is an example of a corefile with proposed geodns directive to trigger the same.

|  |
| --- |
| *A sample CoreDNS corefile:*  *some.net:53 {*  *file some.net*  *geodns*  *errors stdout*  *log stdout*  *}* |

### Implementing Middleware in CoreDNS

To implement a new middleware one has to follow the below two steps

1. Setup middleware
2. Implement the ServeDNS function.

### Setup middleware

PopulateGeoIP

RegisterMiddleWare

LoadGeoipDB

Init

In the setup middleware, GeoDNS middleware registration and the loading of the GeoIP information from the GeoIP database ( i.e. MindMax DB ) is done.

GeoDNS is registered to CoreDNS as a middleware.

This is achieved by executing call to the function “caddy.RegisterPlugin”

We also need to load the GeoIP database files.

We use the MindMax GeoIP database.

#### Init :

It initializes the middleware structure. Currently the structure has one field which stores the GeoIP DB object, and the next middleware.

|  |
| --- |
| *type Geodns struct{*  *Next middleware.Handler*  *GeoIPDB \** *GeoIP*  *}* |

Init calls the RegisterMiddleWare function.

#### RegisterMiddleWare:

This function simply calls the below function to register the new Geodns middleware.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  |  | | *caddy.RegisterPlugin("Geodns",*  *caddy.Plugin{*  *ServerType: "dns",*  *Action: setup,*  *})* |  | |  |  | |

Also the order in which the middleware are executed are based on the order in which the middleware are indexed in the directives slice in the ‘directives.go’.

#### LoadGeoipDB

This function loads the geoip from the mindmax db files using the helper function from ‘github.com/abh/geoip’ package. This function updated the ‘GeoIPDB’ fields of the middleware structure.

### Implementing the ServeDNS:

Any middleware in CoreDNS should implement the ‘middleware.Handler’ interface (i.e. need to implement the ServerDNS method )

This function is called by Coredns, when a request arrives and when a ‘middleware’ directive is found in the ‘corefile’ configuration, the respective middleware ServeDNS function is called.

GetGeoipDestination

GetGeoipSource

ServeDNS

Distance

CalculateShortesDistance

#### ServeDNS:

This function will be called by the CoreDNS, having the following parameters

ServeDNS(ctx context.Context, w dns.ResponseWriter, r \*dns.Msg) (int, error)

dns.MSG has the answer section build by the previous middleware.

In the ServeDNS method of GeoDNS middleware, the nearest geographical location of a A or AAAA record is determined and the corresponding answer section is built and passed on. Hence the index in the directive slice of the file ‘directive.go’ plays a vital role.

#### GetGeoipDestination

This function will populate the geoip locations of all the A and the AAAA records and pass it on to the ServerDNS.

|  |
| --- |
| *var destinationGeoLocation map[string]location*  *type location{*  *latitude float64*  *longitude float64*  *}*  *for \_, r := range dns.MSG.Ns {*  *if dns.TypeA== r.Header().Rrtype || dns.TypeAAAA == r.Header().Rrtype {*  *destinationGeoLocation [destinationAddress] = GetGeoIPInfo(destinationAddress)*  *}*  *}* |

dns.Msg will have all the resource records for the NS , A and AAAA records section of the dns record.

We use this information to get the list of A records in case of ipv4 or AAAA for ipv6 address record and store them in a map (i.e. destinationGeoLocation ) .

#### GetGeoipSource

We get the source ip from the dns.MSG and its geoip location.

|  |
| --- |
| *state := request.Request{W: w, Req: r}*  *sourceIP:= state.IP()* |

#### CalculateShortesDistance

This function iterates through the ‘destinationGeoLocation’ map and finds the distance between the source geo ip. It calculates the shortest distance and return the result to ServeDNS.

#### Distance:

This function calculates the distance between two location using Haversine formula.

The function/methods already available in the following package “github.com/golang/geo/s2”