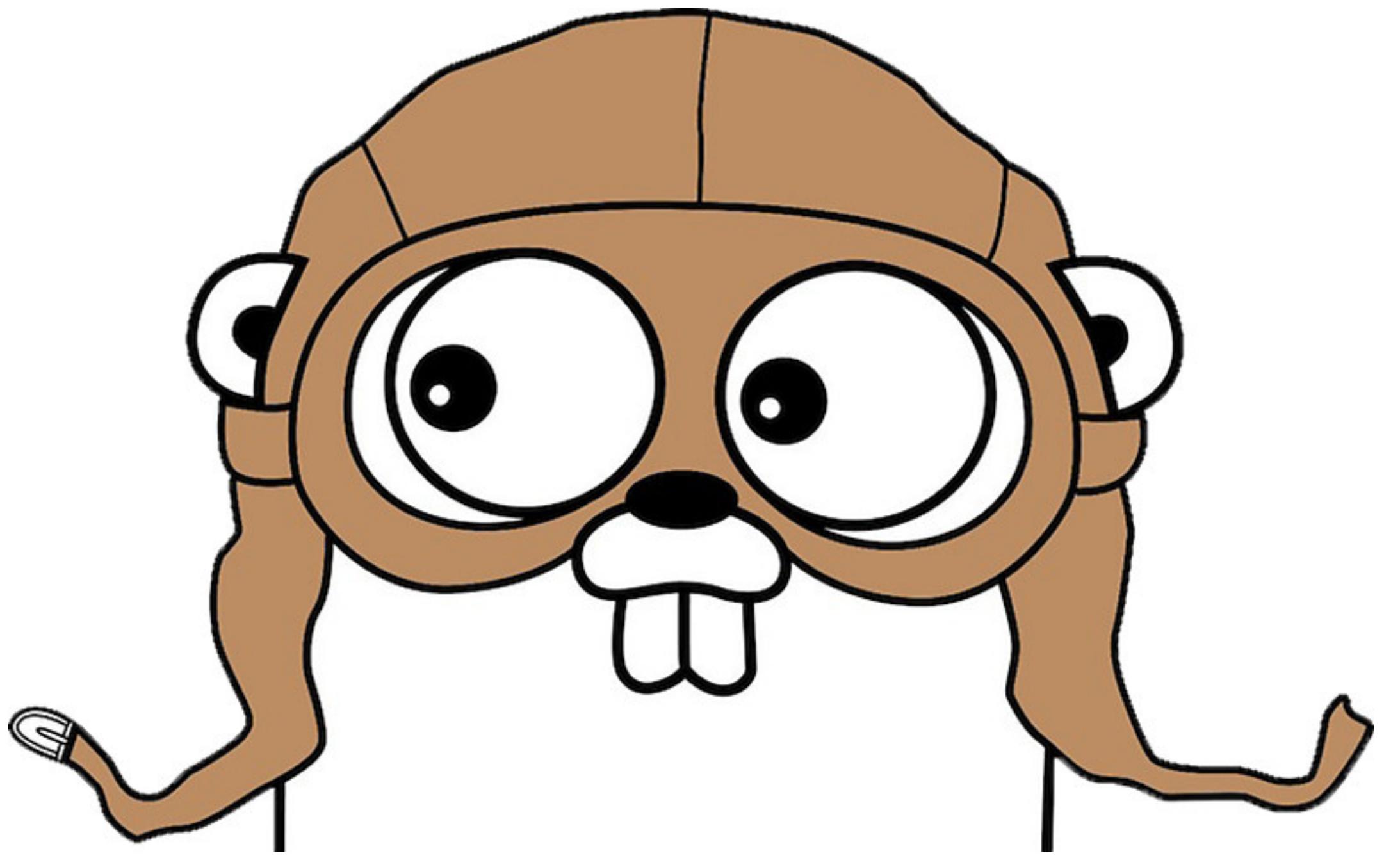


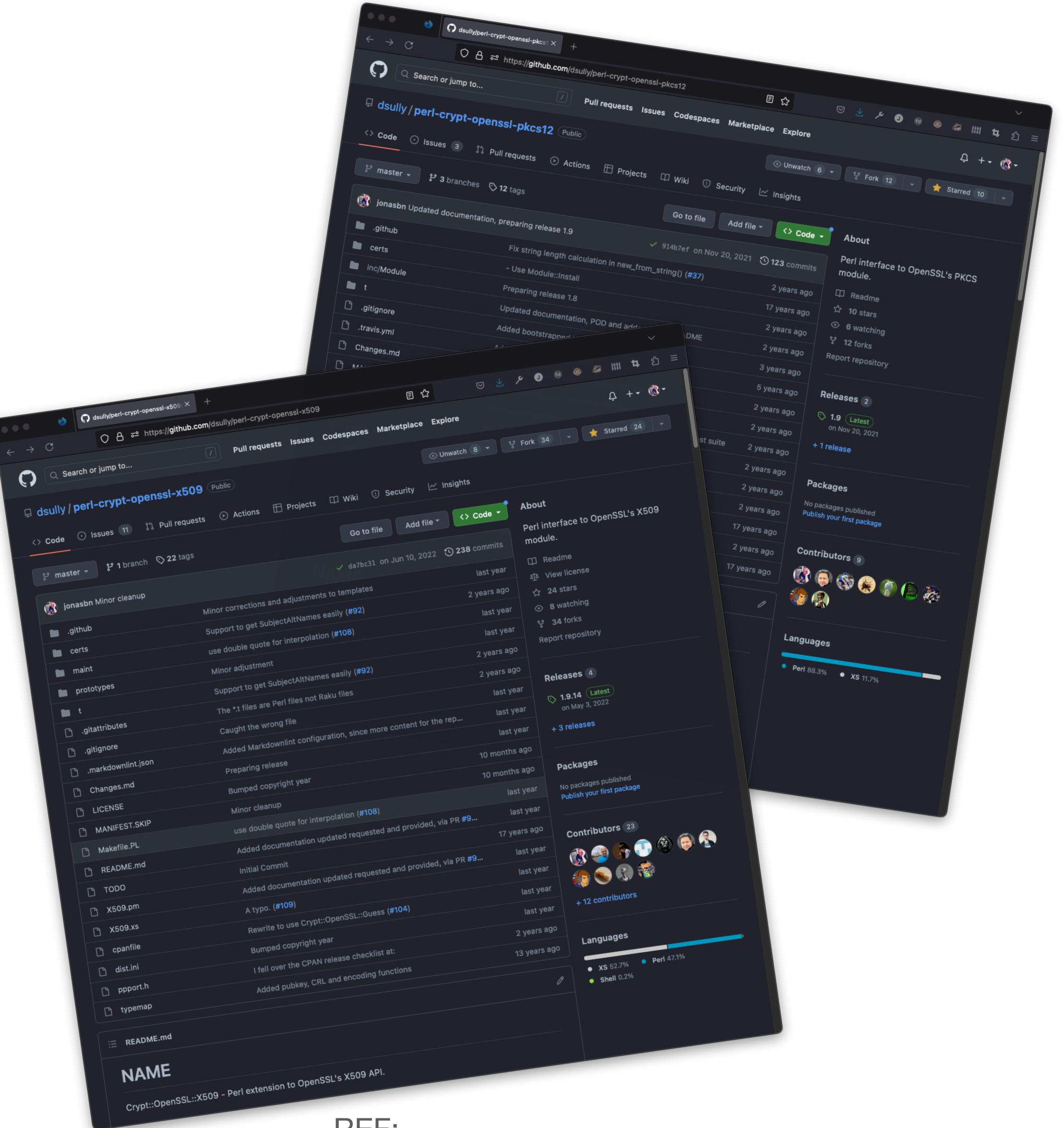
Going serverless with Go on DigitalOcean

An overview of a serverless implementation
and some learning points

- A disclaimer
- I am still learning Go
- I am not a professional programmer
 - I am actually a Product Manager
- I am not a lawyer



- I *maintain* two Perl packages with C-bindings
- Often I have to navigate in all of the available command line flags for Clang
- Not being a C-programmer and working with C on a daily basis, means that this does not come easy



REF:

- <https://github.com/dsully/perl-crypt-openssl-pkcs12>
- <https://github.com/dsully/perl-crypt-openssl-x509>

- The documentation is fairly easy to navigate and I found out that I was able to jump between the different versions easily
- And the versions 4.0.0 to 14.0.0 was pretty uniform
- But instead of opening tabs and comparing the different versions, I decided to make a matrix of the command line flags and versions



REF:

- <https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html>
- <https://releases.llvm.org/N.0.0/tools/clang/docs/DiagnosticsReference.html>

- I implemented a small web scraper in Perl and Mojolicious* to parse the pages and generate the matrix
- The matrix is in Markdown and is served via GitHub pages
- And everything looked good, until I noticed that the rendering **broke** and the Matrix was not completed
- So I reached out to GitHub support

<https://github.com/jonasbn/clang-diagnostic-flags-matrix>

The screenshot shows a GitHub page for the repository `jonasbn/clang-diagnostic-flags-matrix`. The README.md file contains the following content:

```

As part of the data collection fase.  

The proxy reverts this translation and redirects to the proper page. I have published a blog post, giving some more details.

Usage  

carton exec -- perl diag.pl > matrix.md

Installation  

All requirements are listed in the cpanfile, so installation can be performed easily:  

carton

Resources and References  


- My TIL collection: clang diagnostic flags \(GitHub\)
- My TIL collection: clang diagnostic flags \(website\)
- pxy-redirect
- llvm releases documentation site
- pxy.fi



clang command line flags

```

The table below lists various clang command-line flags and their availability across different versions of clang (4.0.0 to 12.0.0). The columns represent clang versions, and the rows represent flags. An 'X' indicates that the flag is supported, while '-' indicates it is not.

	4.0.0	5.0.0	6.0.0	7.0.0	8.0.0	9.0.0	10.0.0	11.0.0	12.0.0
<code>-Rmodule-build</code>	X	X	X	X	X	X	X	X	X
<code>-Rmodule-import</code>	-	-	-	-	-	X	X	X	X
<code>-Rmodule-include-translation</code>	-	-	-	-	-	-	-	-	-
<code>-Rmodule-lock</code>	-	-	-	-	-	-	-	-	-
<code>-Rpass</code>	X	X	X	X	X	X	X	X	X
<code>-Rpass-analysis</code>	X	X	X	X	X	X	X	X	X
<code>-Rpass-missed</code>	X	X	X	X	X	X	X	X	X
<code>-Remark-backend-plugin</code>	X	X	X	X	X	X	X	X	X
<code>-Round-trip-cc1-args</code>	-	-	-	-	-	-	-	-	-
<code>-Rsanitize-address</code>	X	X	X	X	X	X	X	X	X
<code>-Search-path</code>	-	-	-	-	-	-	-	-	-

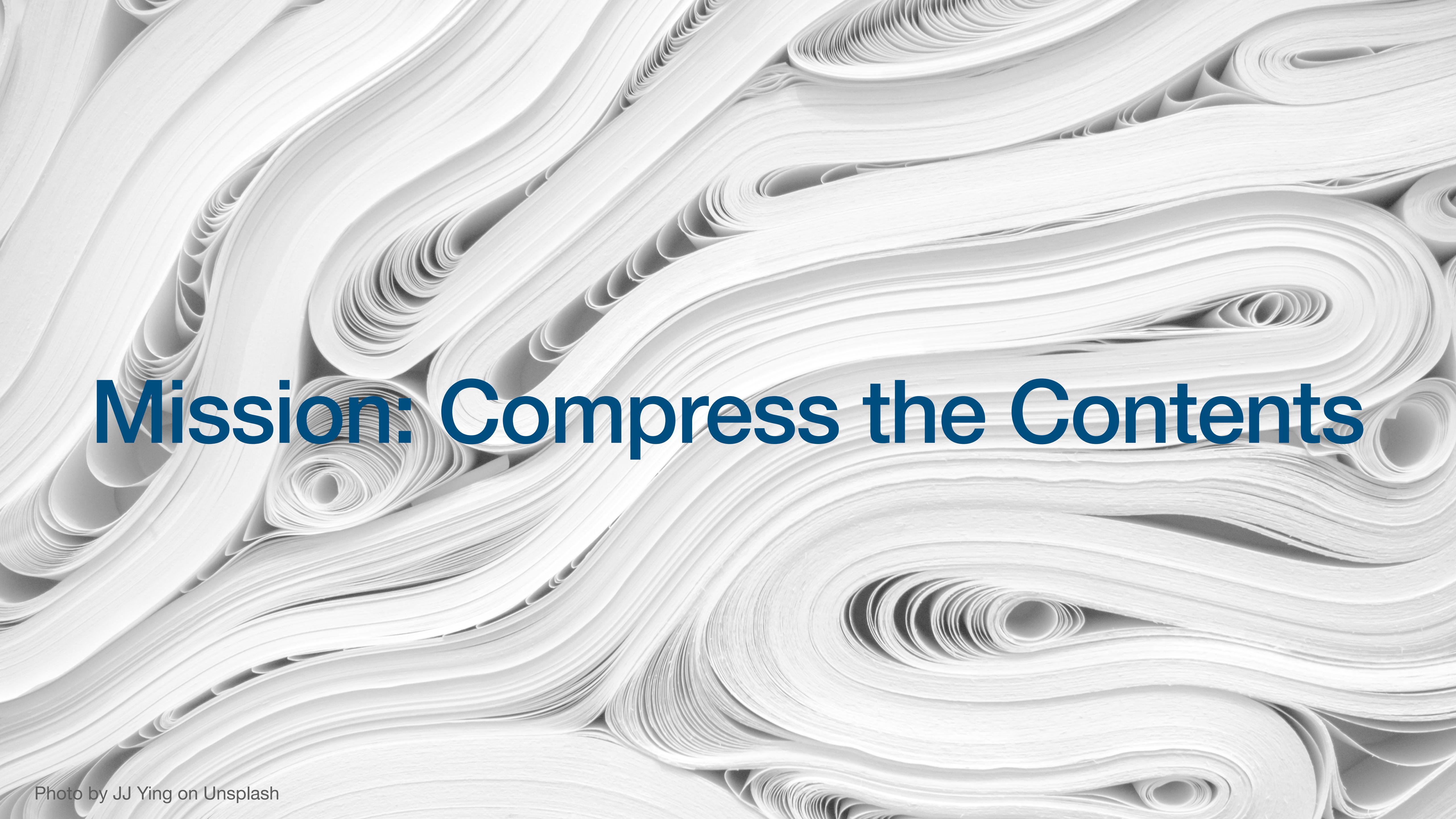
* this is my boring stack

REF: <https://github.com/jonasbn/clang-diagnostic-flags-matrix>

- *Text files over 512 KB are always displayed as plain text. Code is not syntax highlighted, and prose files are not converted to HTML (such as Markdown, AsciiDoc, etc.).*

REF: <https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories#text-limits>





Mission: Compress the Contents

- Most of the contents are links, so I did a comparison

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

<https://releases.llvm.org/5.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

<https://releases.llvm.org/6.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

<https://releases.llvm.org/12.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

<https://releases.llvm.org/14.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

- As you can see the version part of links are uniform
- And only major versions have documentation

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

- And I did a second comparison

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wformat>

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#id200>

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wimplicit-int>

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#id265>

- Only variants are the anchors
- And the major versions

<https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wall>

- This mean I can boil this down to

`https://<domain name>/<major version>/#<anchor>`

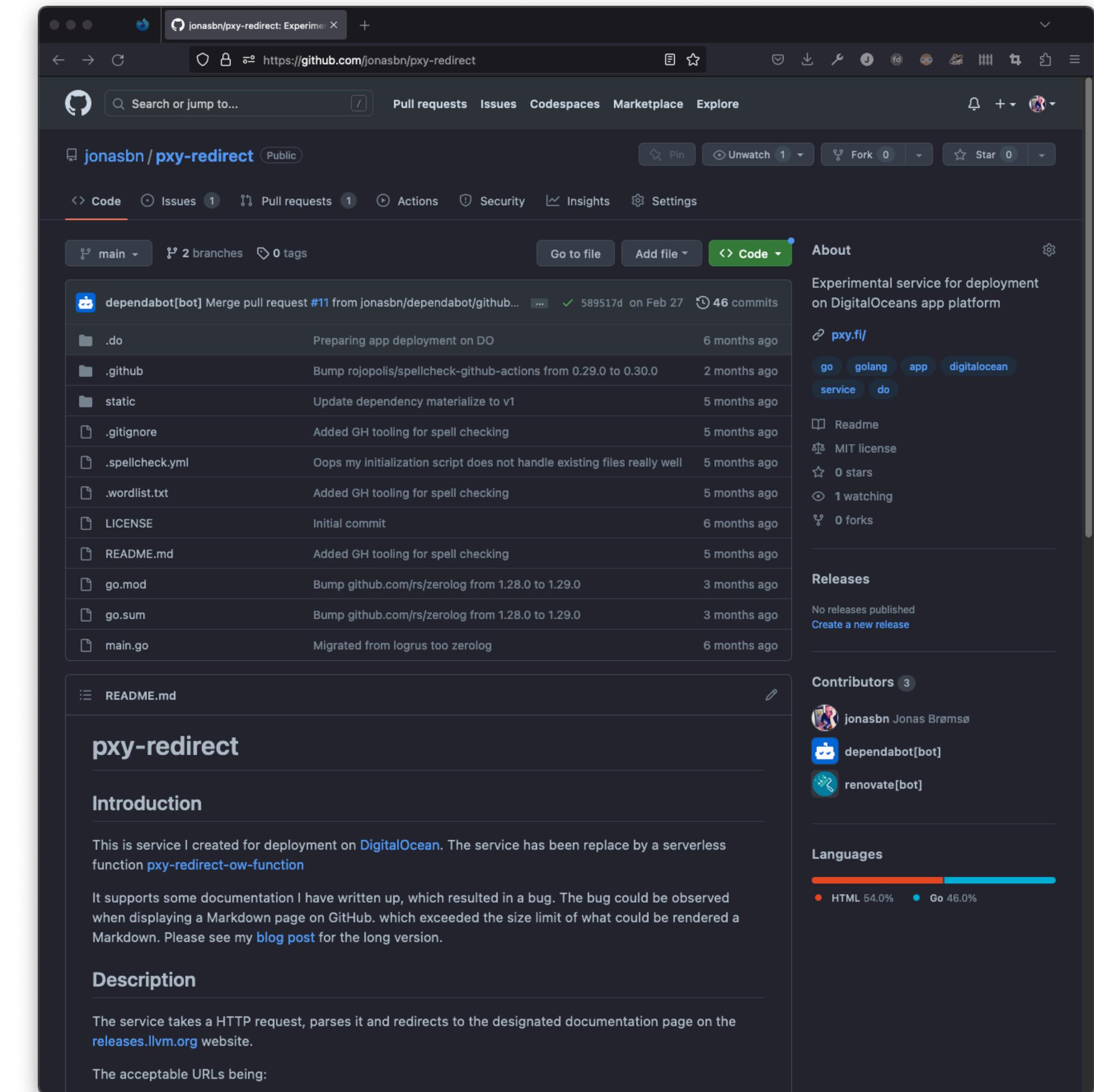
- With a *short* domain name and a single digit major version
- I could redirect from:

`https://<domain name>/4/#wall`

- To:

`https://releases.llvm.org/4.0.0/tools/clang/docs/DiagnosticsReference.html#wall`

- First up I implemented a service and deployed on DigitalOcean - it then struck me
 - *Nobody* is going to use this service, I do not want to pay to have a service running continuously
 - What if we could just pay for used compute and not idle compute?



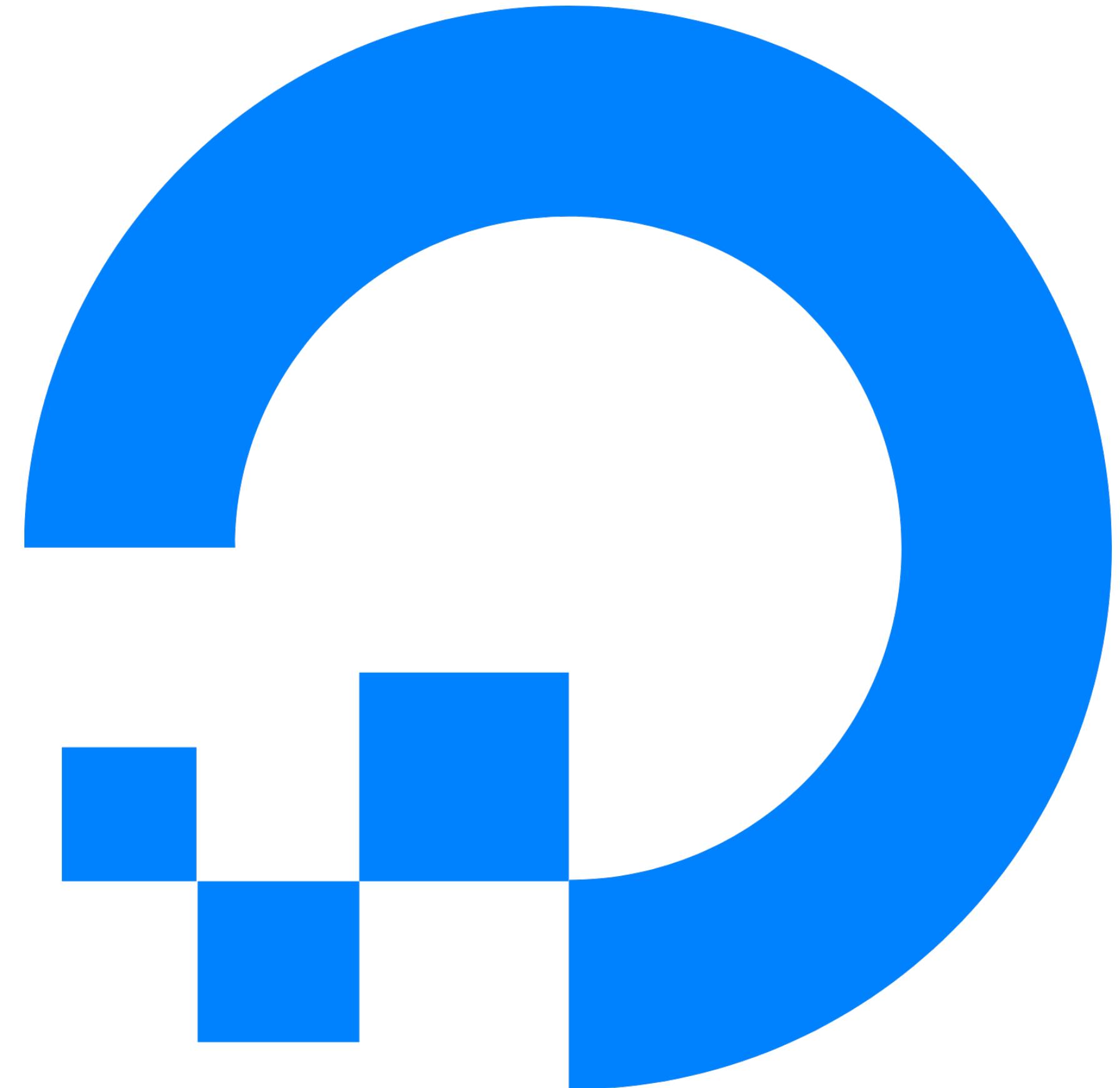
REF: <https://github.com/jonasbn/pxy-redirect>



Mission: Evaluate serverless

- I was already on DigitalOcean (DO), so I decided to check out their serverless function offering
- Requirements:
 - A DigitalOcean account
 - The CLI tool: “doctl” installed

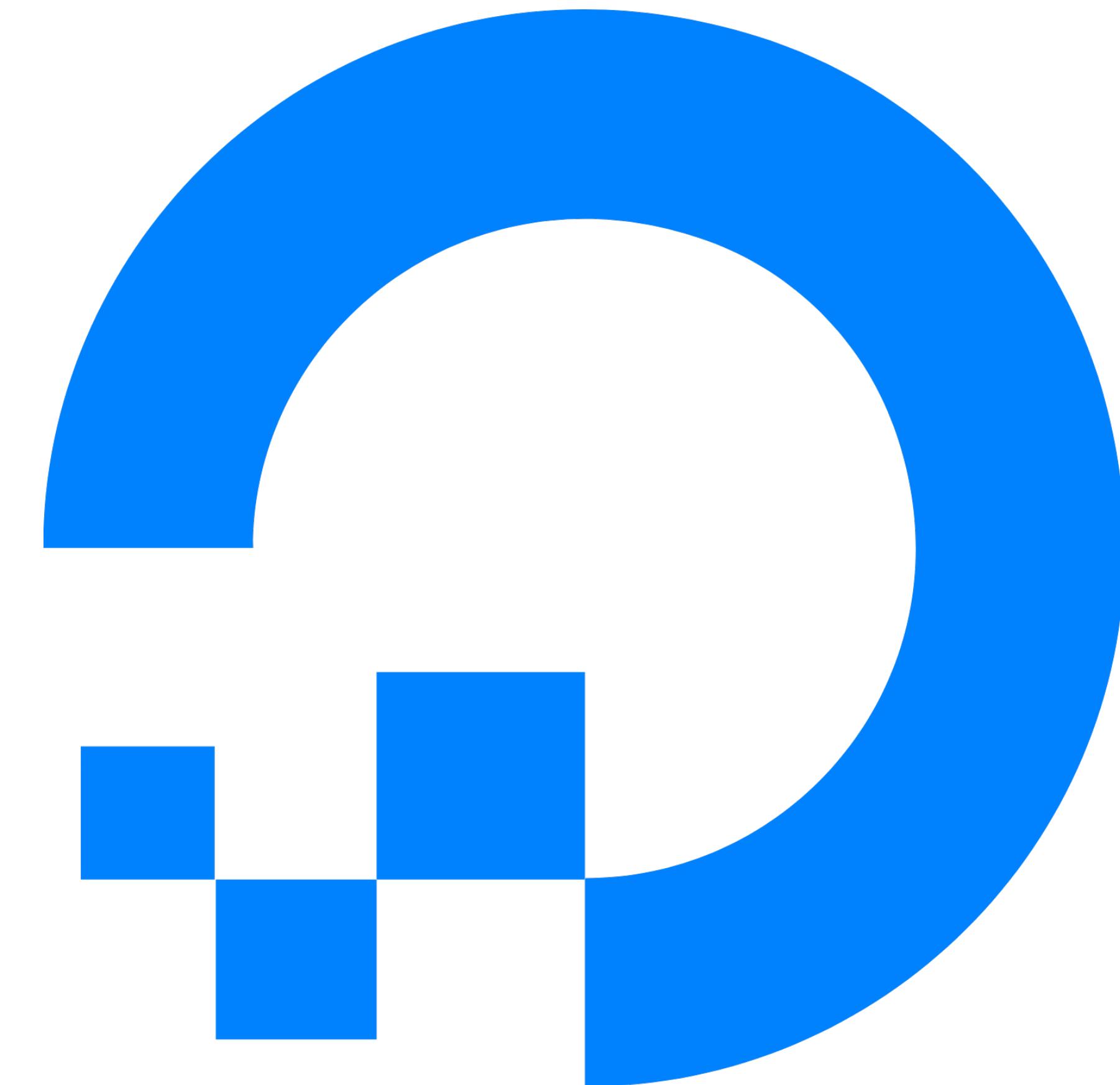
REF: <https://docs.digitalocean.com/products/functions/>



- Repository structure

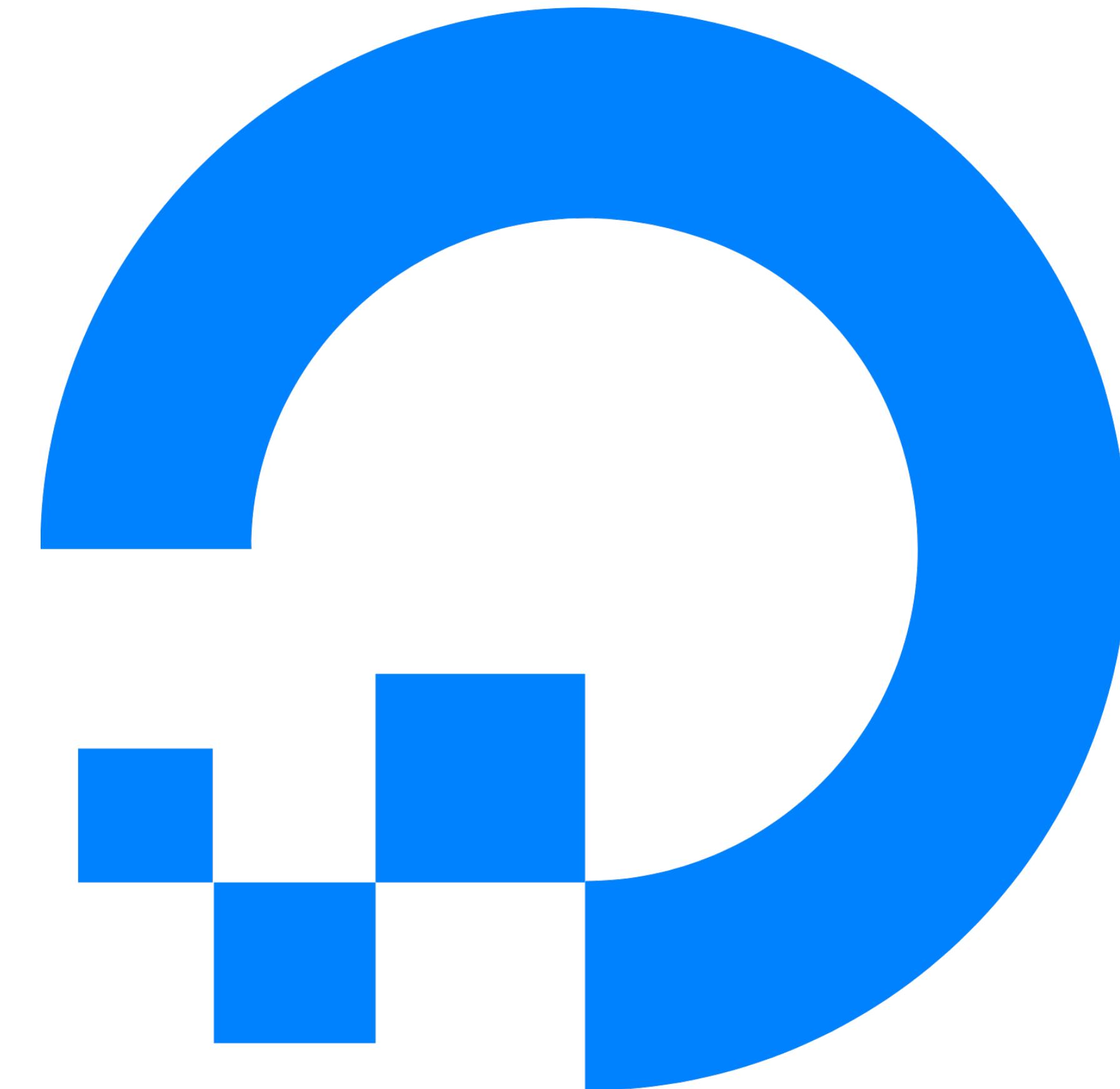
```
sample-functions-golang-helloworld
└── project.yml
    └── packages
        └── sample
            └── hello.go
```

REF: <https://github.com/digitalocean/sample-functions-golang-helloworld>



- Configuration file (project.yml)

```
parameters: {}
environment: {}
packages:
- name: sample
  environment: {}
  parameters: {}
actions:
- name: hello
  main: ''
  runtime: 'go:default'
  environment: {}
  parameters: {}
```



```
package main

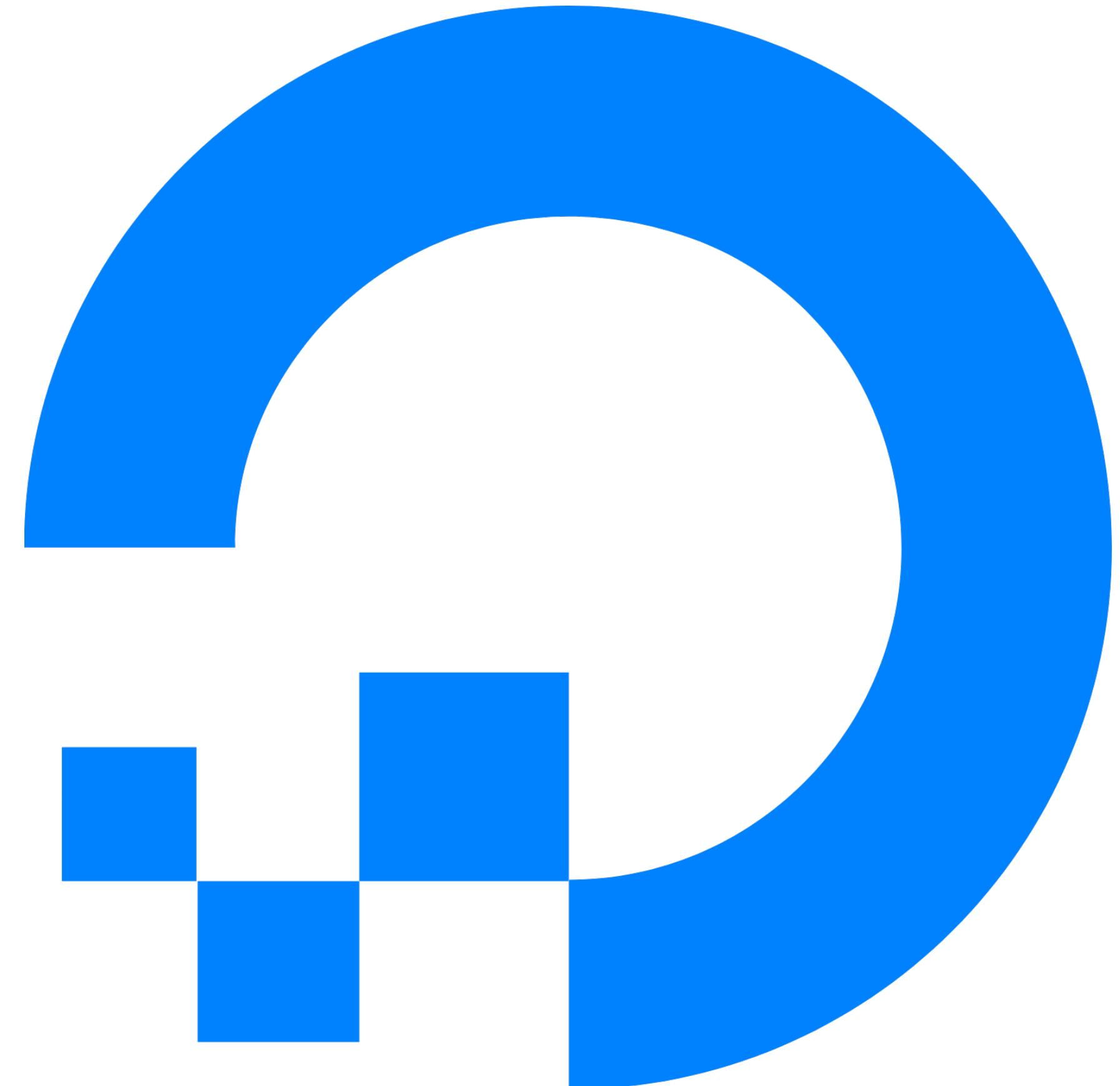
import (
    "fmt"
)

type Request struct {
    Name string `json:"name"`
}

type Response struct {
    StatusCode int           `json:"statusCode,omitempty"`
    Headers    map[string]string `json:"headers,omitempty"`
    Body       string          `json:"body,omitempty"`
}

func Main(in Request) (*Response, error) {
    if in.Name == "" {
        in.Name = "stranger"
    }

    return &Response{
        Body: fmt.Sprintf("Hello %s", in.Name),
    }, nil
}
```



```
$ doctl serverless init --language go sample-hello
```

```
package main

import "fmt"

func Main(args map[string]interface{}) map[string]interface{}
{
    name, ok := args["name"].(string)
    if !ok {
        name = "stranger"
    }
    msg := make(map[string]interface{})
    msg["body"] = "Hello " + name + "!"
    return msg
}
```



```
$ go run sample/hello/hello.go

package main

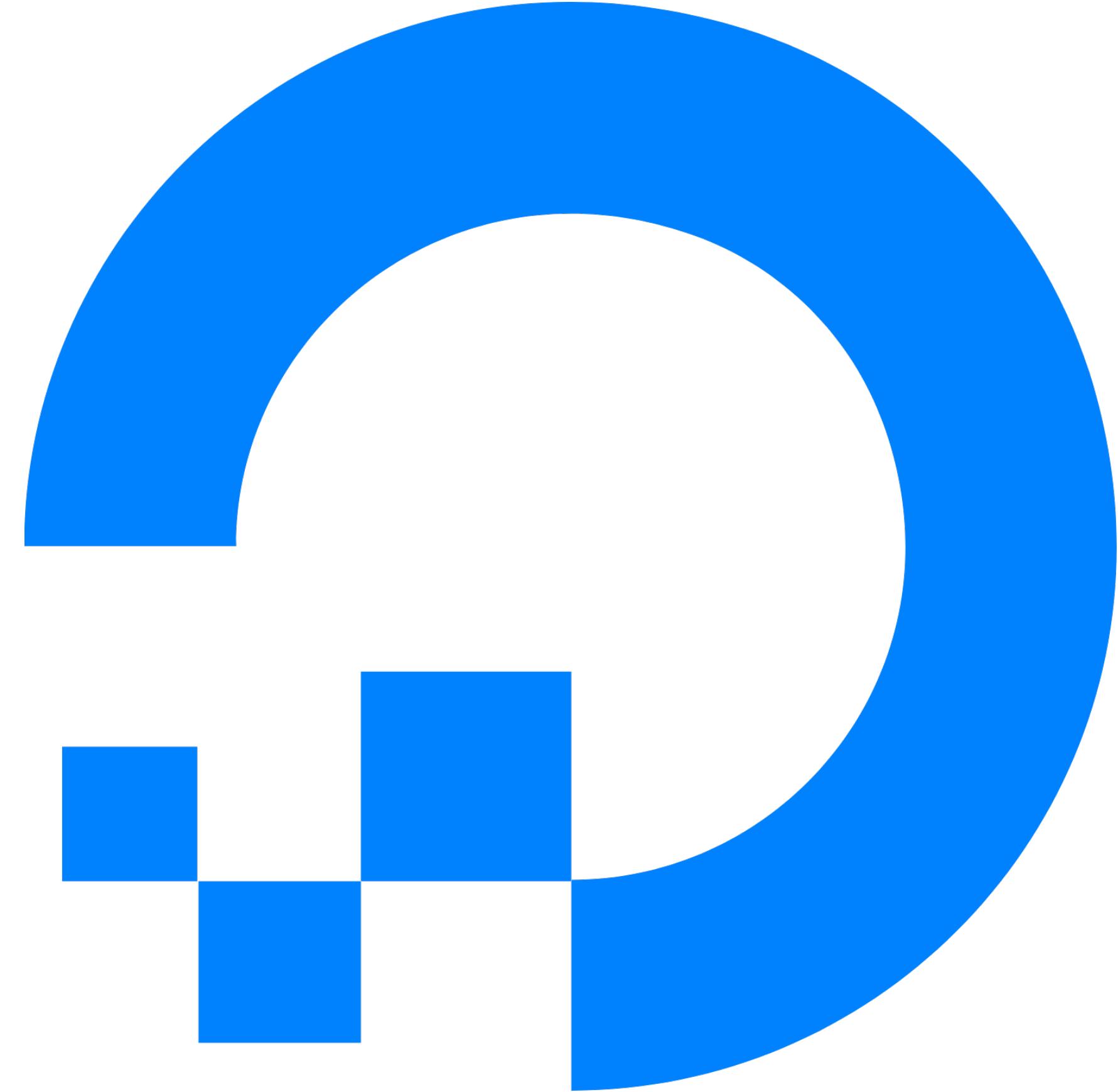
import "fmt"

func Main(args map[string]interface{}) map[string]interface{}
{
    name, ok := args["name"].(string)
    if !ok {
        name = "stranger"
    }
    msg := make(map[string]interface{})
    msg["body"] = "Hello " + name + "!"
    return msg
}

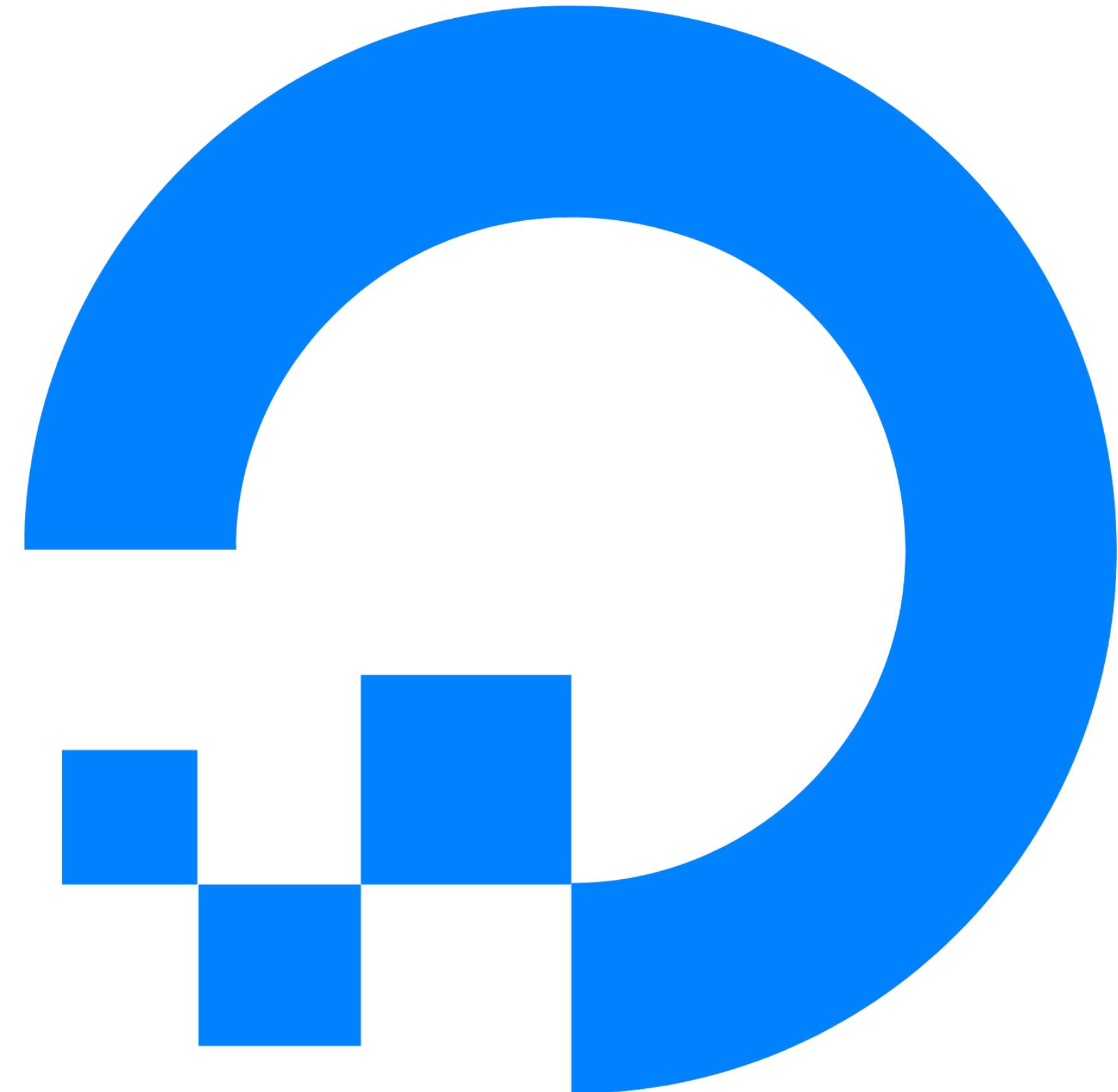
func main() {
    args := make(map[string]interface{})
    r := Main(args)
    fmt.Printf("%+v", r)
}
```



- Demo



- DO functions seemed like a good fit for solving my problem
 - It does not require state
 - It only has to parse and transform received data
 - And finally inform the client of the new location, this by setting the headers to redirect the request
 - All can be lifted from the service



The background of the slide is a blurred photograph of a computer keyboard. The keys are illuminated with a soft blue light, creating a cool, tech-oriented atmosphere. The focus is on the center of the keyboard, with the edges appearing darker and more out of focus.

Mission: Migrate to serverless

- After I did a lot of reading and experimenting - I found out that the DO functions platform is based on Apache Open Whisk
- It is so much easier to find resources on the Internet, if you know what to look for

REF: <https://openwhisk.apache.org/>

Create Your Local Playground

Deploys anywhere

Since Apache OpenWhisk builds its components using containers it easily supports many deployment options both locally and within Cloud infrastructures. Options include many of today's popular Container frameworks such as **Kubernetes** and **OpenShift**, and **Compose**. In general, the community endorses deployment on Kubernetes using **Helm** charts since it provides many easy and convenient implementations for both Developers and Operators alike.

Write functions in any language

Work with what you know and love. OpenWhisk supports a growing list of your favorite languages such as **Go**, **Java**, **NodeJS**, **.NET**, **PHP**, **Python**, **Ruby**, **Rust**, **Scala**, **Swift**. There is also an experimental runtime for **Deno** in-development.

If you need languages or libraries the current "out-of-the-box" runtimes do not support, you can create and customize your own executables as Zip Actions which run on the **Docker** runtime by using the **Docker SDK**. Some examples of how to support other languages using Docker Actions include a tutorial for **Rust** and a completed project for **Haskell**.

Once you have your function written, use the **wsk CLI**, to target your Apache OpenWhisk instance, and run your first action in seconds.

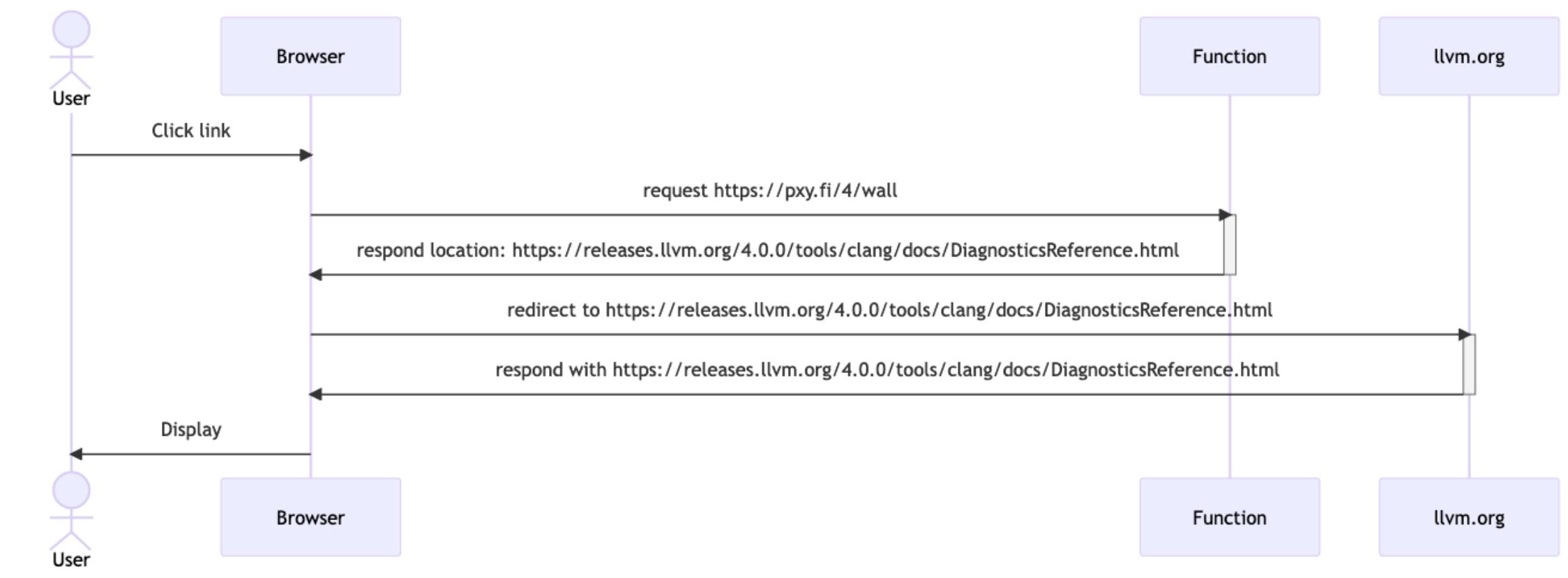
[Create Your First Action](#)

Integrate easily with many popular services

OpenWhisk makes it simple for developers to integrate their Actions with many popular services via **Webhooks** that are provided either as independently developed projects under

- The DO functions solution seem to be aimed at receiving and returning JSON, which is fine, but not for my problem

1. I wanted to parse the path
2. I wanted to return HTTP headers, so I could redirect the request



```
if args["__ow_path"].(string) ≠ "" {
    path = args["__ow_path"].(string)
}

requestHeaders := args["__ow_headers"]
val, _ := requestHeaders.(map[string]interface{})

//  

// path deconstruction and  

// location construction happening here  

//  

//  

headers := make(map[string]string)
headers["location"] = targetURL

return &Response{
    Headers:    headers,
    StatusCode: http.StatusTemporaryRedirect, // 307
}
```

REF: <https://github.com/jonasbn/pxy-redirect-ow-function>



```
func parseRedirectURL(path) (*url.URL, error) {  
  
    redirectURL, parseErr := url.Parse(path)  
  
    if parseErr != nil {  
        return nil, fmt.Errorf("Unable to parse received URL: >%s<", path)  
    }  
  
    return redirectURL, nil  
}
```

REF: <https://github.com/jonasbn/pxy-redirect-ow-function>



```

func assembleTargetURL(url *url.URL) (string, error) {

    s := strings.SplitN(url.Path, "/", 3)

    // 0 is empty because we split on "/" and the URL begins with "/"
    // 1 = version
    // 2 = fragment

    if len(s) < 3 {
        err := fmt.Errorf("insufficient parts in provided url: >%s<", url.String())
        return "", err
    }

    _, err := strconv.Atoi(s[1])
    if err != nil {
        err := fmt.Errorf("first part of url: >%s< is not a number: %q", url.String(), s)
        return "", err
    }

    if s[2] == "" {
        err := fmt.Errorf("second part of url: >%s< is not a string: %q", url.String(), s)
        return "", err
    }

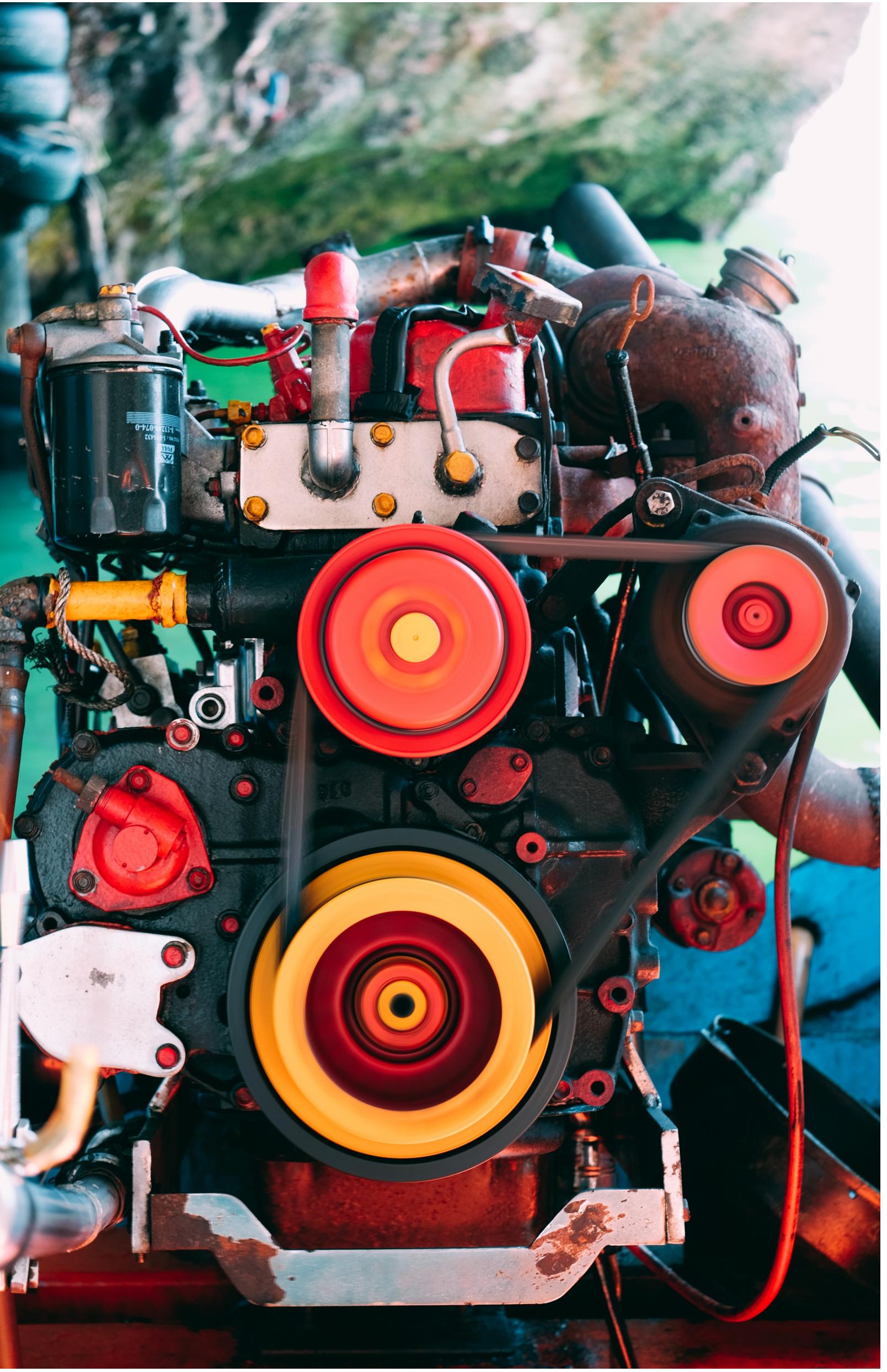
    return fmt.Sprintf("https://releases.llvm.org/%s.0.0/tools/clang/docs/DiagnosticsReference.html#%s", s[1], s[2]), nil
}

```

REF: <https://github.com/jonasbn/pxy-redirect-ow-function>



- Demo



The background of the image is a blurred photograph of a computer keyboard. The keys are illuminated with a soft blue light, creating a glowing effect against a dark background. The keys are slightly out of focus, giving the image a dreamlike or futuristic feel.

Mission: Take it to the next level

- I attempted to do some establish some remote logging
- Due to the way the functions are arranged I felt I was loosing out on some information, so I decided to put a reverse proxy in front of the function platform
- DO has a load-balancer offering, but it was above my budget
- So I spun up a droplet and installed NGINX
- Now I have standardised remote logging using Logtail, alternatives are:
 - Papertrail
 - Datadog

- Next up is getting the error handling/communication to work better
- And I want a *cool* log Dashboard
- Implementing a reverse proxy mad me think this can actually be solved using NGINX
- The current compression might not be suffice over time
- Adding links to the matrix for a new version cost about 44.000 KB
- With the addition of version 16 the matrix has a size of: 507 KB