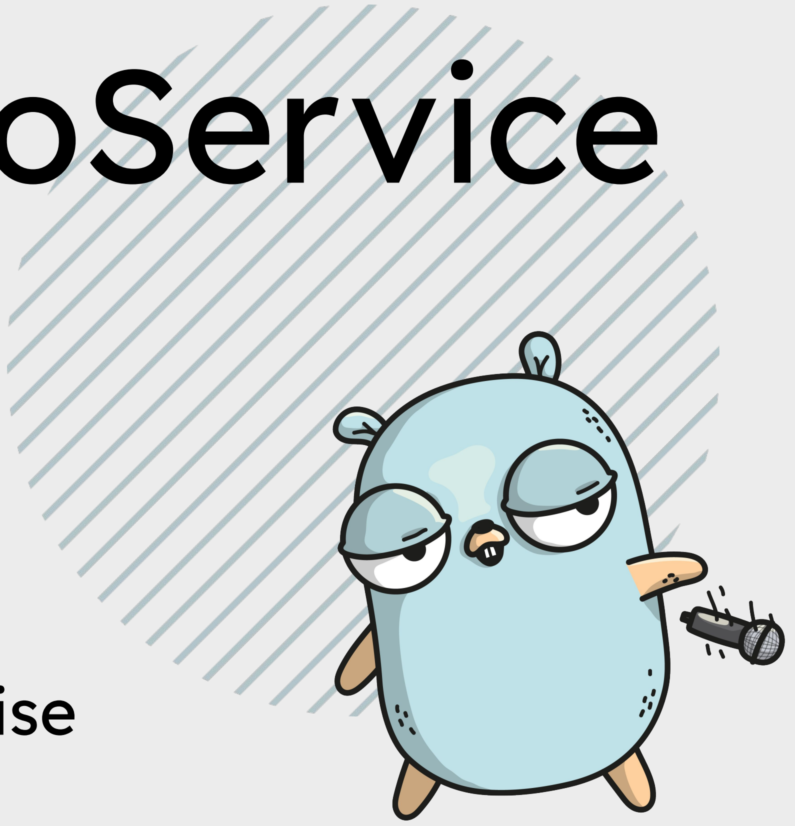
Building Minimalistic Backend MicroService in Go



박래철 (@raeperd) / Kakao Enterprise

박래철

@raeperd on [Github](#) | [Linkedin](#) | [Brunch](#)

- Kakao Enterprise, Daum Search Service
- 2 years in Go
- Contributor of golangci-lint

Microservice in Go needs...

- Reading config, Graceful shutdown, ...
- Testability
- API documents
- Logging
- Profiling, Error monitoring, Metic, Tracing, ...

Solve it with minimal code

- Single main.go file
- Under 200 lines of code
- Standard package only
- In a **scalable** ways

How to use this talk

- Code on Github
- Review, then use
- **You can change it!**

I choose …

- **Keep code minimal**
- Testability
- Conciseness
- There is trade-off

https://raeperd.dev/go2024

Session Slide

Let’s Start Coding …

👨‍💻

Tiny main abstraction

```go
func main() {
  ctx := context.Background()
  if err := run(ctx, os.Stdout, os.Args); err != nil {
    fmt.Fprintf(os.Stderr, "%s\n", err)
    os.Exit(1)
  }
}


func run(ctx context.Context, w io.Writer, args []string) error {
  var port uint
  fs := flag.NewFlagSet(args[0], flag.ExitOnError)
  fs.SetOutput(w)
  fs.UintVar(&port, "port", 8080, "port for http api")
  if err := fs.Parse(args[1:]); err != nil {
    return err
  }
  // ...
}
```

ref: How I write HTTP services in Go after 13 years - Mat Ryer

http.Server with Graceful shutdown

```go
func run(ctx context.Context, w io.Writer, args []string) error {
  // ...
  server := &http.Server{/*...*/}

  ctx, cancel := signal.NotifyContext(ctx, syscall.SIGINT, syscall.SIGTERM)
  defer cancel()
  go func() {
    if err := server.ListenAndServe(); err != http.ErrServerClosed {
      // ...
    }
  }()
  <-ctx.Done()

  ctx, cancel = context.WithTimeout(ctx, 5*time.Second)
  defer cancel()
  if err := server.Shutdown(ctx); err != nil {
    return err
  }
  return nil
}
```

Test http layer in control

```go
func TestHandler(t *testing.T) {
  port := getFreePort()
  go func() {
    var buf bytes.Buffer
    err := run(ctx, &buf, []string{"app", "--port", port})
    // ...
  }()
  address := "http://localhost:" + port + "/health"

  ctx = context.WithTimeout(ctx, 2*time.Second)
  err := waitForHealthy(ctx, address)
  testNil(t, err)

  res, err := http.Get(address)
  testNil(t, err)
  testEqual(t, http.StatusOK, res.StatusCode)
}
```

Test http layer in control

```go
func TestHandler(t *testing.T) {
  port := getFreePort()
  go func() {
    var buf bytes.Buffer
    err := run(ctx, &buf, []string{"app", "--port", port})
    // ...
  }()
  address := "http://localhost:" + port + "/health"

  ctx = context.WithTimeout(ctx, 2*time.Second)
  err := waitForHealthy(ctx, address)
  testNil(t, err)

  res, err := http.Get(address)
  testNil(t, err)
  testEqual(t, http.StatusOK, res.StatusCode)
}
```

```go
func waitForHealthy(ctx context.Context, endpoint string) error {
  for {
    select {
    case <-ctx.Done():
      return errors.New("context done before healthy")
    default:
      res, err := http.Get(endpoint)
      if err == nil && res.StatusCode == http.StatusOK {
        return nil
      }
      time.Sleep(250 * time.Millisecond)
    }
  }
}
```

- Use run() and waitForHealthy() every time?

```go
func TestMain(m *testing.M) {
  flag.Parse() // NOTE: this is needed

  // ...
  go func() {
    err := run(ctx, os.Stdout, []string{"testapp", "--port", port()})
    // ...
  }()

  ctx = context.WithTimeout(ctx, 2*time.Second)
  err := waitForHealthy(ctx, endpoint()+"/health")
  // ...

  os.Exit(m.Run())
}
```

- TestMain to setup & tear-down test
- **Setup database, test-containers etc...**

```go
func TestGetHealth(t *testing.T) {
  res, err := http.Get(endpoint() + "/health")
  testNil(t, err)
  testEqual(t, http.StatusOK, res.StatusCode)
}

func TestGetOpenapi(t *testing.T) {
  res, err := http.Get(endpoint() + "/openapi.yaml")
  testNil(t, err)
  testEqual(t, http.StatusOK, res.StatusCode)
}

// ...
```

- Test without httptest.NewRecorder(), httptest.NewServer() ...

health check with information

```
  $ curl http://localhost:8080/health
{
  "Version": "v1.0.0",
  "Uptime": "52.671218125s",
  "LastCommitHash": "8d9e2b79ce85",
  "LastCommitTime": "2024-10-03T12:51:17Z",
  "DirtyBuild": false
}
```

Embed Version

```go
var Version string
```

```
$ go build -o app -ldflags '-X main.Version=$(VERSION)' .
```

main, v1.0.0, …

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      ...
      - run: go build -o app -ldflags '-w -X main.Version=${{ github.ref_name }}'
```

Concise http.Handler

```go
func handleGetHealth(version string) http.HandlerFunc {
  type responseBody struct {
    Version        string    `json:"Version"`
    Uptime         string    `json:"Uptime"`
    LastCommitHash string    `json:"LastCommitHash"`
    LastCommitTime time.Time `json:"LastCommitTime"`
    DirtyBuild     bool      `json:"DirtyBuild"`
  }

  // ...

  return func(w http.ResponseWriter, r *http.Request) {
    // ...
  }
}
```

buildInfo with debug package

```go
func handleGetHealth(version string) http.HandlerFunc {
  // ...

  buildInfo, _ := debug.ReadBuildInfo()
  for _, kv := range buildInfo.Settings {
    switch kv.Key {
    case "vcs.revision":
      res.Revision = kv.Value
    case "vcs.time":
      res.Time, _ = time.Parse(time.RFC3339, kv.Value)
    case "vcs.modified":
      res.Modified = kv.Value == "true"
    }
  }

  return func(w http.ResponseWriter, r *http.Request) {
    // ...
}
```

http.Handler with closure

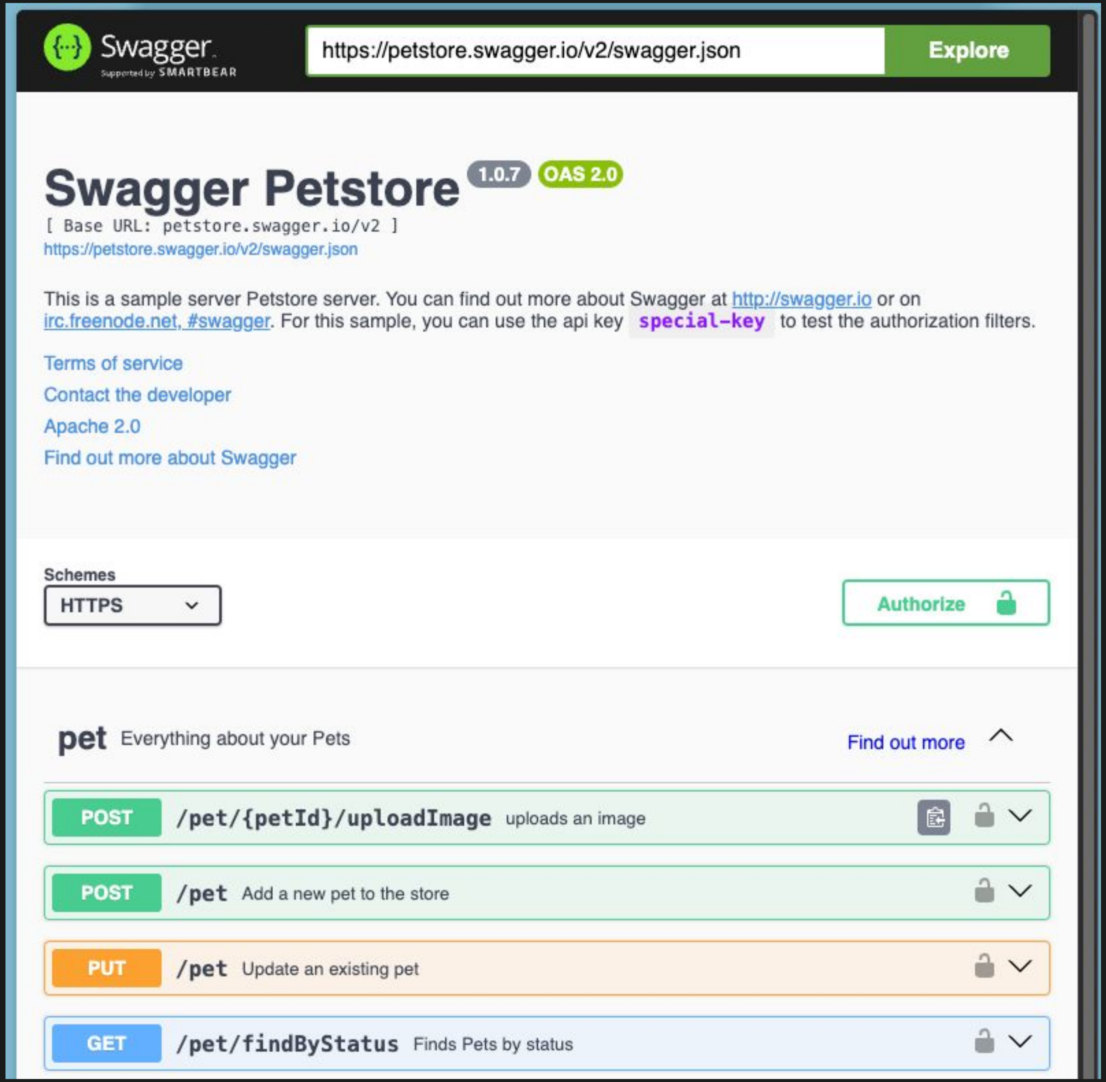
```go
func handleGetHealth(version string) http.HandlerFunc {
  // ...

  up := time.Now()
  return func(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(200)

    res.Uptime = time.Since(up).String()
    _ = json.NewEncoder(w).Encode(res)
  }
}
```

Doc is a Must

- For effective communications

OpenAPI3

```yaml
openapi: 3.0.0
info:
  title: title
  description: description
  version: {{ $VERSION }}
servers:
  - url: http://api.example.com/v1
    description: production
  - url: http://staging-api.example.com
    description: staging
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: description
      responses:
        200:    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```



Swagger Petstore 1.0.7 OAS 2.0

[Base URL: petstore.swagger.io/v2]
https://petstore.swagger.io/v2/swagger.json

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

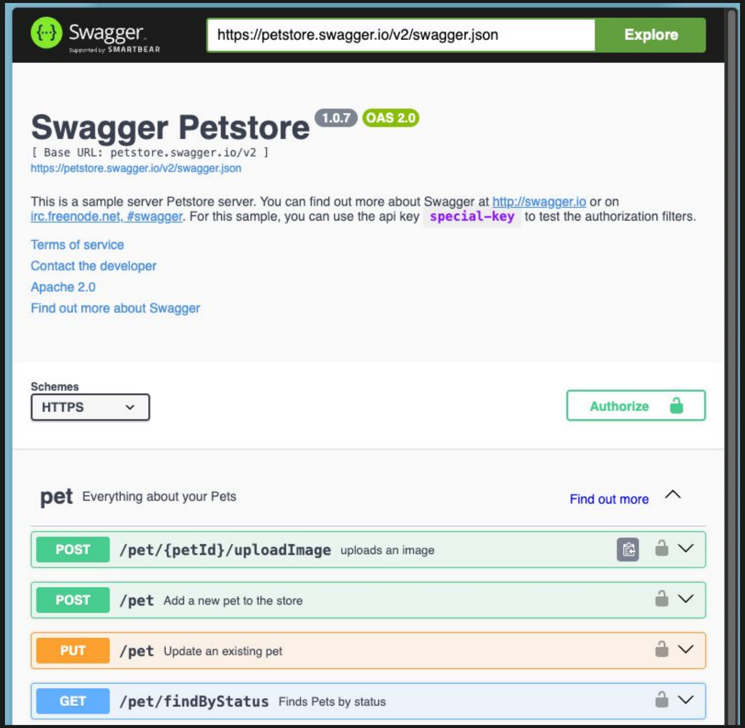
Schemes
HTTPS

Authorize

pet Everything about your Pets — Find out more

POST /pet/{petId}/uploadImage uploads an image
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status

OpenAPI3 - Serve yaml



Swagger
Supported by SMARTBEAR
https://petstore.swagger.io/v2/swagger.json
Explore

Swagger Petstore 1.0.7 OAS 2.0
[Base URL: petstore.swagger.io/v2]
https://petstore.swagger.io/v2/swagger.json

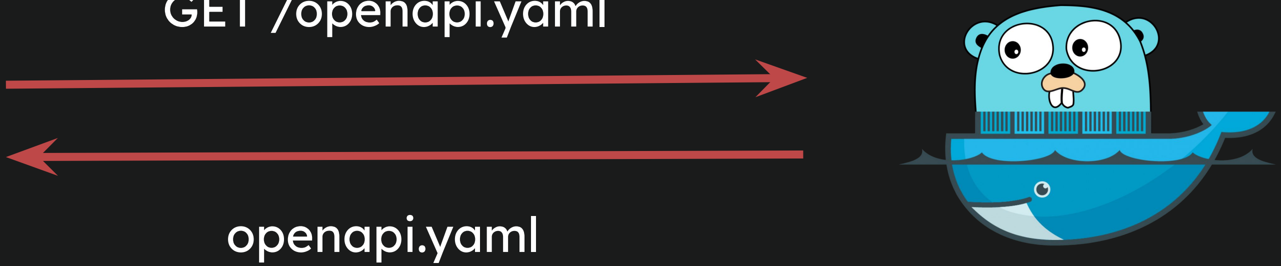
This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Schemes
HTTPS
Authorize

pet Everything about your Pets
Find out more

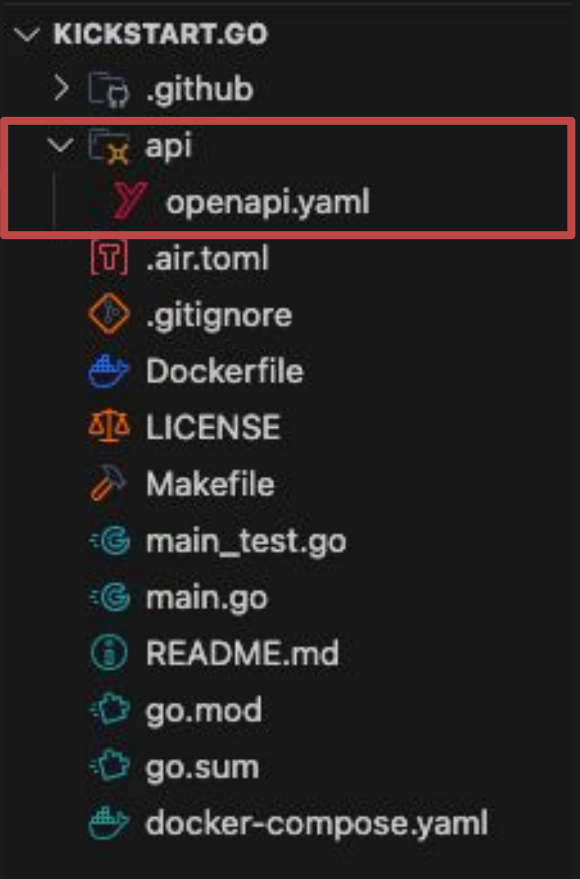
POST /pet/{petId}/uploadImage uploads an image
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status



handleGetOpenAPI with embed package

```go
func handleGetOpenAPI(version string) http.HandlerFunc {
  body := bytes.Replace(openapi, []byte("${{ VERSION }}"), []byte(version), 1)

  return func(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "text/plain")
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.WriteHeader(200)
    _, _ = w.Write(body)
  }
}
```

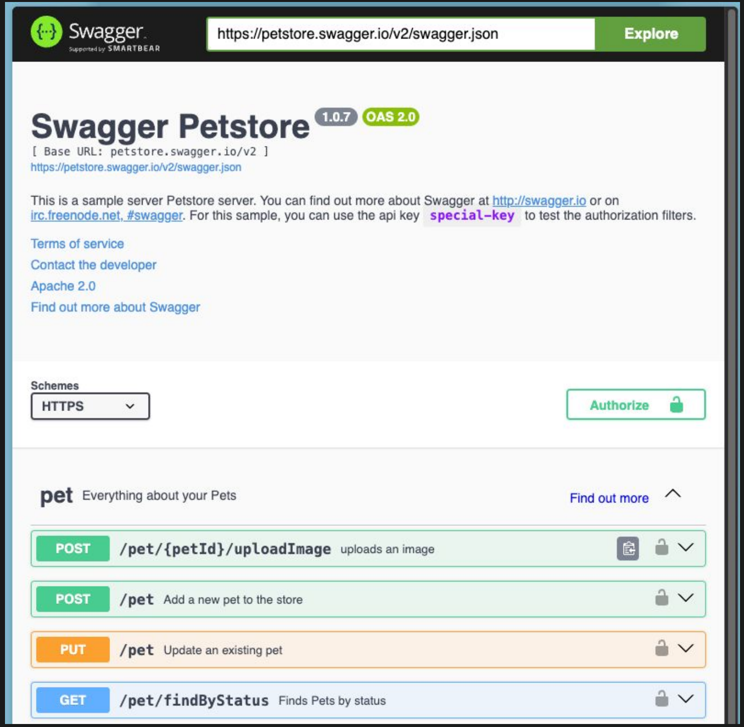
```go
//go:embed api/openapi.yaml
var openapi []byte
```



```
KICKSTART.GO
  .github
  api
    openapi.yaml
  .air.toml
  .gitignore
  Dockerfile
  LICENSE
  Makefile
  main_test.go
  main.go
  README.md
  go.mod
  go.sum
  docker-compose.yaml
```

- Binary embeds openapi []bytes
- **Deploy one executables without extra**
- ⚠️ CORS ERROR ⚠️

handleGetOpenAPI



Swagger
Supported by SMARTBEAR
https://petstore.swagger.io/v2/swagger.json
Explore

Swagger Petstore 1.0.7 OAS 2.0

[Base URL: petstore.swagger.io/v2]
https://petstore.swagger.io/v2/swagger.json

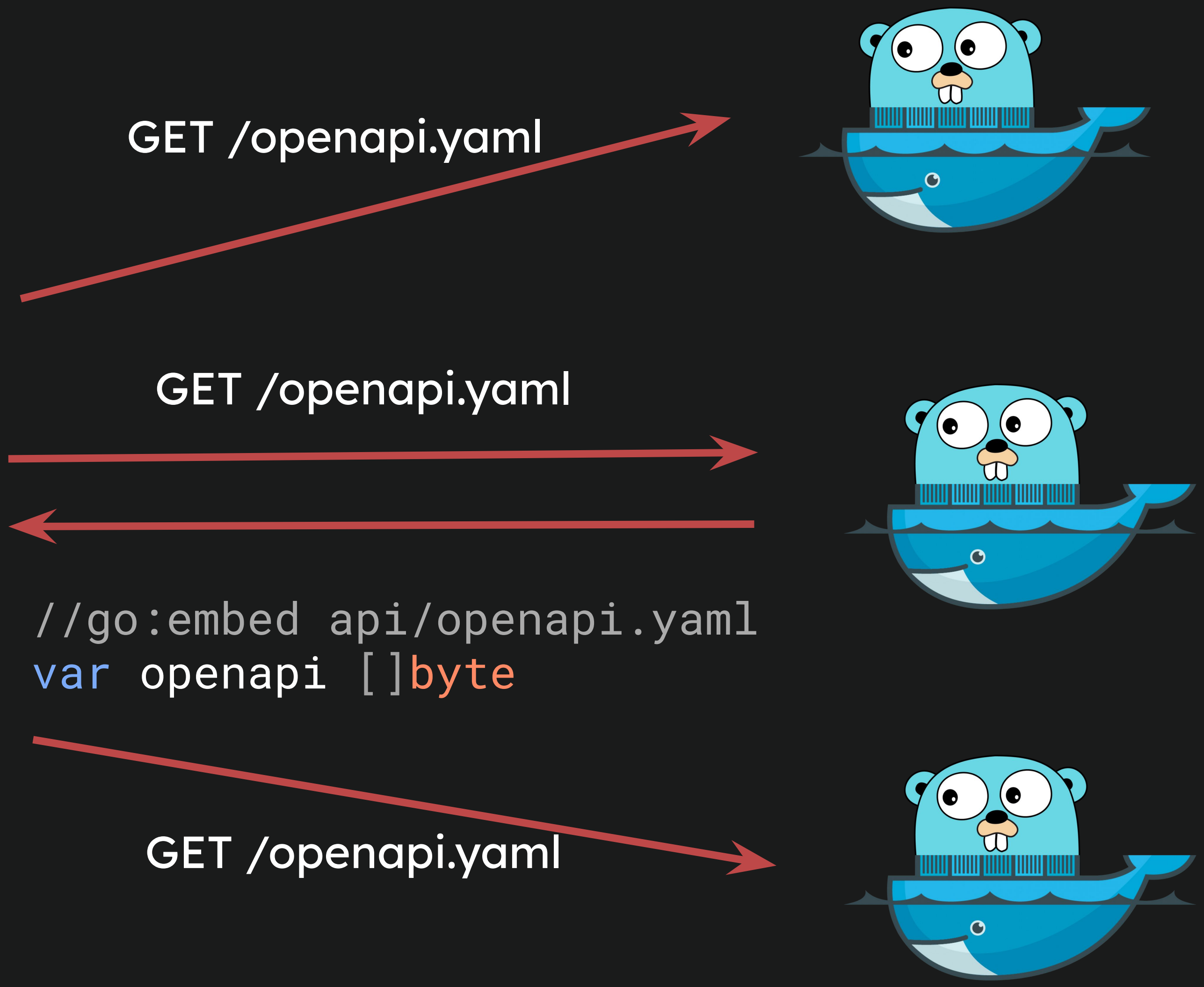
This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Schemes: HTTPS
Authorize

pet Everything about your Pets — Find out more

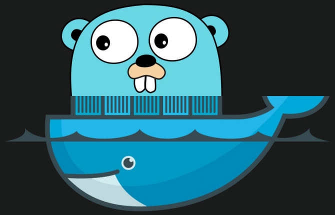
POST	/pet/{petId}/uploadImage	uploads an image
POST	/pet	Add a new pet to the store
PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status



```go
//go:embed api/openapi.yaml
var openapi []byte
```

Logging is a Must

Logging is a Must

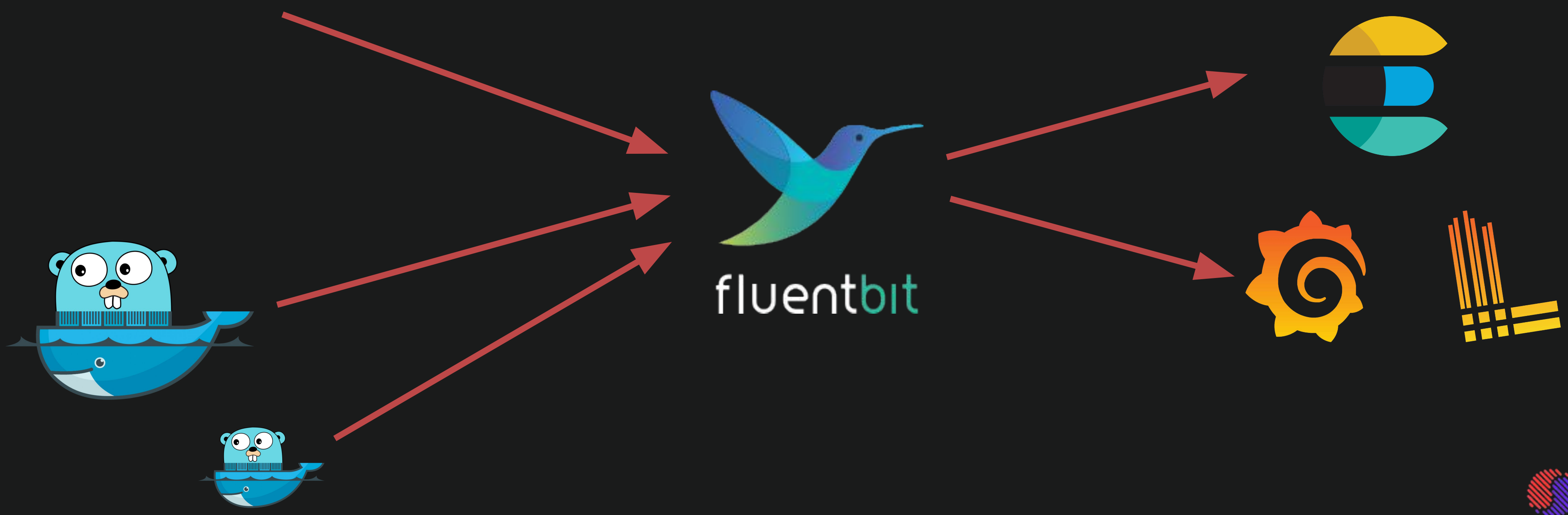


```
{"time":"2024-10-04T00:21:24","level":"INFO","latency":"94.125µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"53.542µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"41.125µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"29.542µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:48","level":"INFO","latency":"35.291µs","method":"GET","path":"/health"}
```

- Structured logging in slog
- Stdout log is enough! - ref: 12 factor app

Scale with logging processor

```
{"time":"2024-10-04T00:21:24","level":"INFO","latency":"94.125µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"53.542µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"41.125µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:47","level":"INFO","latency":"29.542µs","method":"GET","path":"/health"}
{"time":"2024-10-04T00:21:48","level":"INFO","latency":"35.291µs","method":"GET","path":"/health"}
```



Scale with logging processor 2

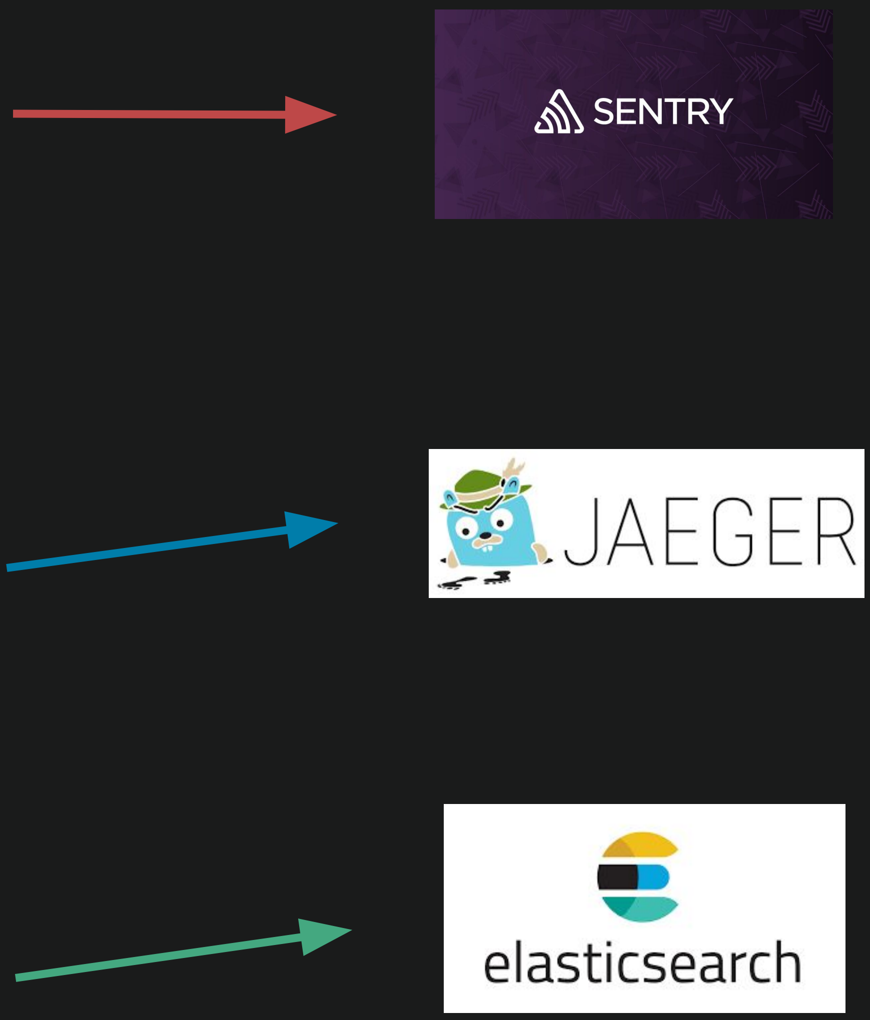
⚠️ need more field to work ⚠️

check fluentbit.io

```
[FILTER]
    Name grep
    Match *
    Regex key error
    Tag has_error

[OUTPUT]
    Name http
    Match has_error
    Host <sentry-host>
```

```
[FILTER]
    Name grep
    Match *
    Regex key traceId
    Tag has_traceId

[OUTPUT]
    Name http
    Match has_traceId
    Host <jaeger-collector-host>
```

```
[OUTPUT]
    Name es
    Match *
    Host <elasticsearch-host>
```



accesslog Middleware

```go
func accesslog(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    wr := responseRecorder{ResponseWriter: w}

    next.ServeHTTP(&wr, r)

    slog.InfoContext(r.Context(), "accessed",
      slog.Int("status", wr.status),
      slog.Int("bytes", wr.numBytes))
      slog.String("latency", time.Since(start).String()),
      // ...
  })
}
```

Decorate http.ResponseWriter

```go
type responseRecorder struct {
  http.ResponseWriter
  status   int
  numBytes int
}

func (re *responseRecorder) Header() http.Header {
  return re.ResponseWriter.Header()
}

func (re *responseRecorder) Write(b []byte) (int, error) {
  re.numBytes += len(b)
  return re.ResponseWriter.Write(b)
}

func (re *responseRecorder) WriteHeader(statusCode int) {
  re.status = statusCode
  re.ResponseWriter.WriteHeader(statusCode)
}
```

Decorate http.ResponseWriter

```go
type responseRecorder struct {
  http.ResponseWriter
  status   int
  numBytes int
}

func (re *responseRecorder) Header() http.Header {
  return re.ResponseWriter.Header()
}

func (re *responseRecorder) Write(b []byte) (int, error) {
  re.numBytes += len(b)
  return re.ResponseWriter.Write(b)
}

func (re *responseRecorder) WriteHeader(statusCode int) {
  re.status = statusCode
  re.ResponseWriter.WriteHeader(statusCode)
}
```

recover Middleware

```go
func recovery(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        defer func() {
            if err := recover(); err != nil && err != http.ErrAbortHandler {
                stack := make([]byte, 1024)
                n := runtime.Stack(stack, true)

                slog.ErrorContext(r.Context(), "panic!",
                    slog.String("error", err.Error()),
                    slog.String("stack", string(stack[:n])),
                    slog.String("method", r.Method),
                    //...
            }
        }()

        wr := responseRecorder{ResponseWriter: w}
        next.ServeHTTP(&wr, r)
    })
}
```

log example

```
{
  "time": "2024-10-04T00:21:48",
  "level": "INFO",
  "msg": "accessed",
  "latency": "35.291µs",
  "method": "GET",
  "path": "/health",
  "query": "",
  "ip": "[::1]:50368",
  "status": 200,
  "bytes": 167
}
```

```
{
  "time": "2024-10-04T00:22:39",
  "level": "ERROR",
  "error": "out of index"
  "msg": "panic from",
  "method": "GET",
  "path": "/users",
  "query": "",
  "ip": "[::1]:50368",
  "status": 500,
  "bytes": 167
}
```

SENTRY

slack

There's more,

But I'll sum up. ⏰

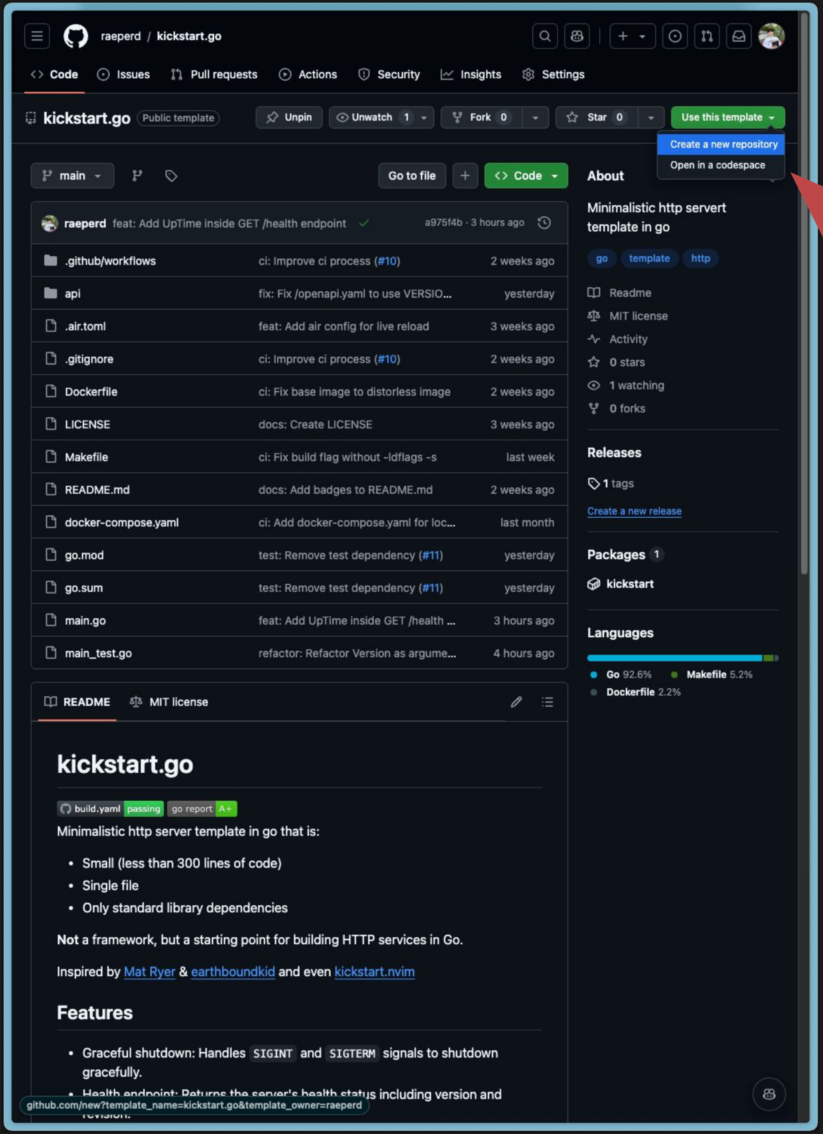
Sum up

- Simple endpoint test with run()
- Healthcheck with version & debug info
- API Docs with version and openapi.yaml
- std logging, error notification with `slog` and fluentbit
- **with one main.go, under 200 loc, no 3rd party**

You missed...

- **Makefile & Dockerfile**
- **JenkinsFile, github-actions, ...**
- lint using golangci-lint
- live reload
- **pprof**
- ...

github.com/raeperd/kickstart.go



raeperd feat: Add UpTime inside GET /health endpoint ✓ a975f4b · 3 hours ago

.github/workflows	ci: Improve ci process (#10)	2 weeks ago
api	fix: Fix /openapi.yaml to use VERSIO...	yesterday
.air.toml	feat: Add air config for live reload	3 weeks ago
.gitignore	ci: Improve ci process (#10)	2 weeks ago
Dockerfile	ci: Fix base image to distorless image	2 weeks ago
LICENSE	docs: Create LICENSE	3 weeks ago
Makefile	ci: Fix build flag without -ldflags -s	last week
README.md	docs: Add badges to README.md	2 weeks ago
docker-compose.yaml	ci: Add docker-compose.yaml for loc...	last month
go.mod	test: Remove test dependency (#11)	yesterday
go.sum	test: Remove test dependency (#11)	yesterday
main.go	feat: Add UpTime inside GET /health ...	3 hours ago
main_test.go	refactor: Refactor Version as argume...	4 hours ago

About

Minimalistic http servert template in go

go template http

Readme
MIT license
Activity
0 stars
1 watching
0 forks

Releases

1 tags

Create a new release

Packages 1

kickstart

Languages

Go	92.6%
Makefile	5.2%
Dockerfile	2.2%

README MIT license

kickstart.go

build.yaml passing go report A+

Minimalistic http server template in go that is:

- Small (less than 300 lines of code)
- Single file
- Only standard library dependencies

Not a framework, but a starting point for building HTTP services in Go.

Inspired by Mat Ryer & earthboundkid and even kickstart.nvim

Features

- Graceful shutdown: Handles `SIGINT` and `SIGTERM` signals to shutdown gracefully.
- Health endpoint: Returns the server's health status including version and revision.

github.com/new?template_name=kickstart.go&template_owner=raeperd

Full code with docs

Use this template -> Create a new repo

Star ⭐ to comeback!

Fork 🔗 and change

Work in progress to be on awesome

Thank You!

🙏

Welcome feedback on

or @raeperd linkedin