WHO GOES THERE?

Developing a Github user audit tool with Go, event driven lambda, graphQL and serverless

WHO AM I?

John Gregory

Senior Software Engineer at Admiral Financial Services (AFSL), artist, and proud Gopher

SO WHO GOES WHERE NOW?

1. How did we get here?
2. Key technologies
3. Bringing it all together

HOW DID WE GET HERE?

Some background and motivations

“

The **Office for National Statistics (ONS)** was formed on **1 April 1996** by the merger of the Central Statistical Office (CSO) and the Office of Population Censuses and Surveys (OPCS).

https://en.wikipedia.org/wiki/Office_for_National_Statistics

A WIDE RANGE OF REVISION CONTROL

OPEN SOURCE

https://github.com/onsdigital

300+ USERS
And growing!

100+ TEAMS
With a lot of mobility

1.1K REPOSITORIES
That's quite a few!

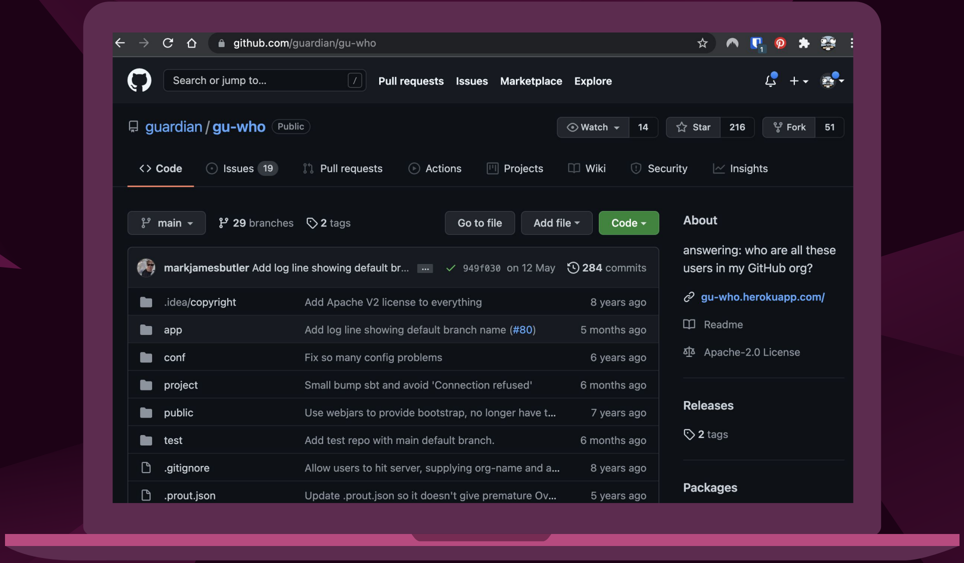
Office for National Statistics github usage*

**Approx. user, team and repository numbers circa start 2020*

INSPIRATION: GU-WHO

A github user auditing tool written in **Scala**.

ONS had few staff with suitable **Scala** experience to install and maintain as a mission critical tool.



github.com/guardian/gu-who

guardian / gu-who Public

29 branches 2 tags

markjamesbutler Add log line showing default br... 949f030 on 12 May 284 commits

.idea/copyright	Add Apache V2 license to everything	8 years ago
app	Add log line showing default branch name (#80)	5 months ago
conf	Fix so many config problems	6 years ago
project	Small bump sbt and avoid 'Connection refused'	6 months ago
public	Use webjars to provide bootstrap, no longer have t...	7 years ago
test	Add test repo with main default branch.	6 months ago
.gitignore	Allow users to hit server, supplying org-name and a...	8 years ago
.prout.json	Update .prout.json so it doesn't give premature Ov...	5 years ago

About

answering: who are all these users in my GitHub org?

gu-who.herokuapp.com/

Readme

Apache-2.0 License

Releases

2 tags

Packages

https://github.com/guardian/gu-who

USING GO!

Go was a great fit as it has:

- ➢ **fast** start up and execution
- ➢ **simplicity** for learning and support
- ➢ good **support** across the services I wished to use.

DESIGN CHOICES

Whys, whens, and wherefores

KEY TECHNOLOGIES

Built serverless with **aws lambda**

Using **lambda** for flexibility and cost.

Deployed with **serverless framework**

Simplifying deployment using the **serverless framework**.

Query Github using **GraphQL API**

Communicating with Github using their **GraphQL API** for simple querying.

Event driven using **aws sqs queues**

Using **sqs** as an event queue for scalability and extensibility of the service.

λ WHY LAMBDA?

- ➢ **Cost effective** for infrequent runs - only pay for what you use.
- ➢ Can be **triggered** by a timed event.
- ➢ No **servers** to manage.
- ➢ Easy routing with **destinations**.

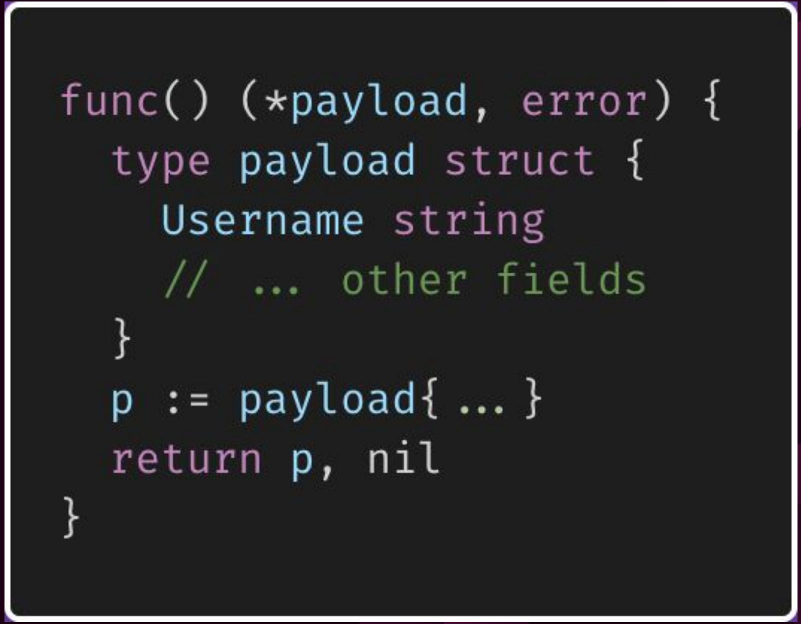
***AWS Lambda** is a compute service that lets you run code without provisioning or managing servers.*

Maintenance, provisioning, scaling and logging are all built in.

ASIDE: LAMBDA DESTINATIONS

A **destination** allows you to specify where to route your response if the **lambda** succeeds or fails.

For **Go** we can easily use this to automatically marshal a struct payload.



```
func() (*payload, error) {
  type payload struct {
    Username string
    // ... other fields
  }
  p := payload{...}
  return p, nil
}
```

WHY SERVERLESS?

- ➢ **Replicable builds** with infrastructure as code.
- ➢ Built for **serverless** applications such as **lambda**.
- ➢ **Cross cloud** support..
- ➢ Easy to **package** whole application.

*The **Serverless Framework** is a free* and open-source framework for deploying serverless code.*

Offers from simple deploy to full-lifecycle and monitoring services.

**base offering is free, but additional support tiers and features are paid.*

WHY GRAPHQL?

- **Single API call** instead of potentially many with REST.
- No **over-** or **under-fetching**.
- Built in **validation** and **type checking**.

***GraphQL** is an API query language, originally created by Facebook.*

Schema-based, rather than endpoint based.

ASIDE: GRAPHQL VS REST

	GraphQL	**REST**
Architecture	client driven	server driven
Organisation	schema & types	endpoints
Performance	fast	more calls can take more time
Operations	Query, Mutation, Subscription	Create, Read, Update, Delete
Data Fetching	specific data in single call	fixed data in multiple calls
Stability	less error prone; automatic validation and type checking	better choice for complex queries

Taken from https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/

WHY EVENT DRIVEN?

- ➢ **Events** can be anything that something may be interested in
- ➢ Easy **decoupling** of services.
- ➢ Enables **extensibility**.

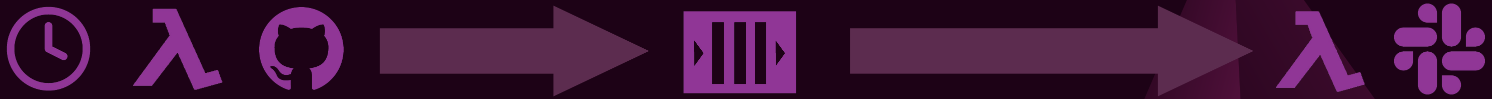
*An **event driven** architecture uses events to trigger and communicate between decoupled services.*

*They comprise **producers, routers** and **consumers.***

BRINGING IT TOGETHER

Architecture and code dive

HIGH LEVEL FLOW



Call Github GraphQL API → Publish report to event queue → Clients receive report

The **checker lambda** is activated by a *Cloudwatch scheduled event* and calls the Github GraphQL API to create a report.

The report is published via an **on_success** lambda action to the **event queue**

Clients subscribe to the queue and receive the report. They can then take action such as firing notifications / alerts to a variety of channels.

CODE DIVE

https://github.com/necrophonic/who-goes-there

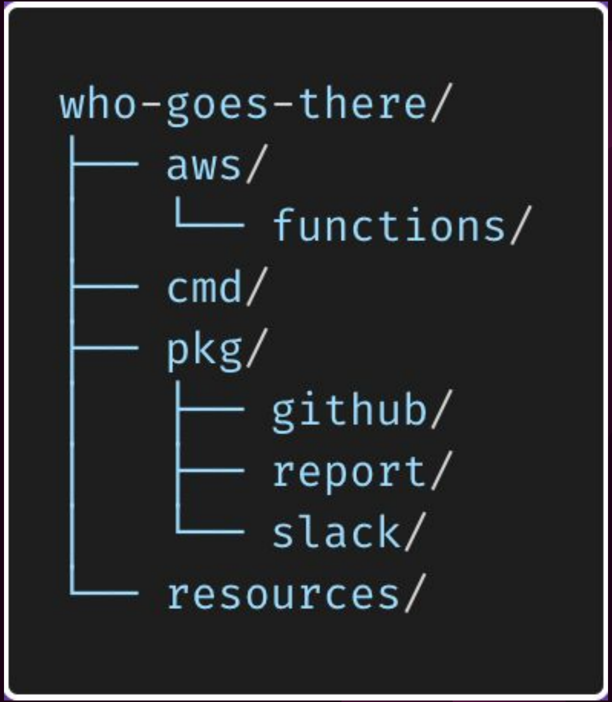
REPOSITORY LAYOUT

aws/ - contains the lambda function code and serverless configuration

cmd/ - an example command line runner

pkg/ - standard folder for local packages

resources/ - extra fluff like slackbot avatar images



```
who-goes-there/
├── aws/
│   └── functions/
├── cmd/
├── pkg/
│   ├── github/
│   ├── report/
│   └── slack/
└── resources/
```

THE CHECKER LAMBDA

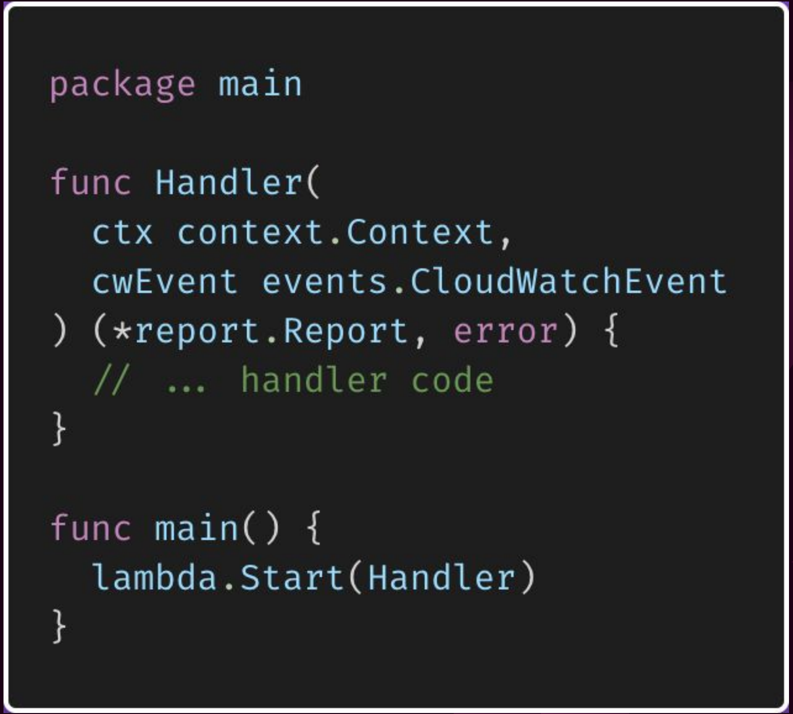
/aws/functions/checker/

CHECKER LAMBDA

#1/4 - STRUCTURE

Structured as a typical **go lambda**

Entry point via **main()** and invoking the **aws sdk** function: *lambda.Start()*



```go
package main

func Handler(
  ctx context.Context,
  cwEvent events.CloudWatchEvent
) (*report.Report, error) {
  // ... handler code
}

func main() {
  lambda.Start(Handler)
}
```

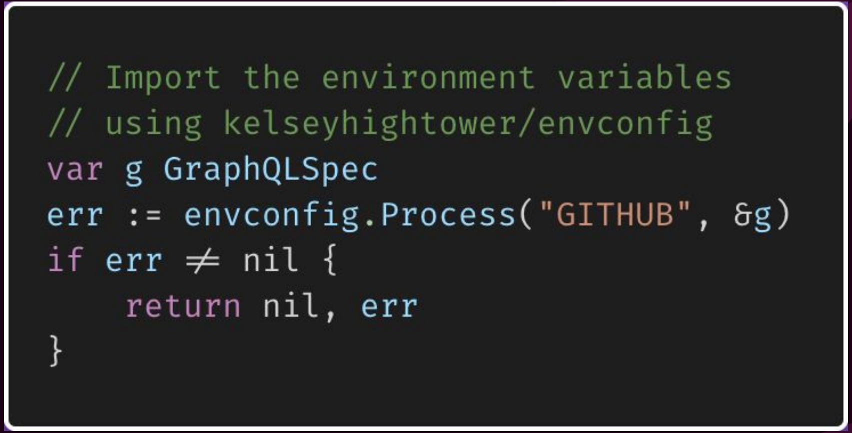
/aws/functions/checker/checker.go

CHECKER LAMBDA

#2/4 - IMPORT ENVIRONMENT VARIABLES

Import **environment variables** such as *organisation name* and the github *access token*.

Should ideally use more secure storage such as (on **aws**) an encrypted **ssm** value or **kms**.



```
// Import the environment variables
// using kelseyhightower/envconfig
var g GraphQLSpec
err := envconfig.Process("GITHUB", &g)
if err ≠ nil {
    return nil, err
}
```

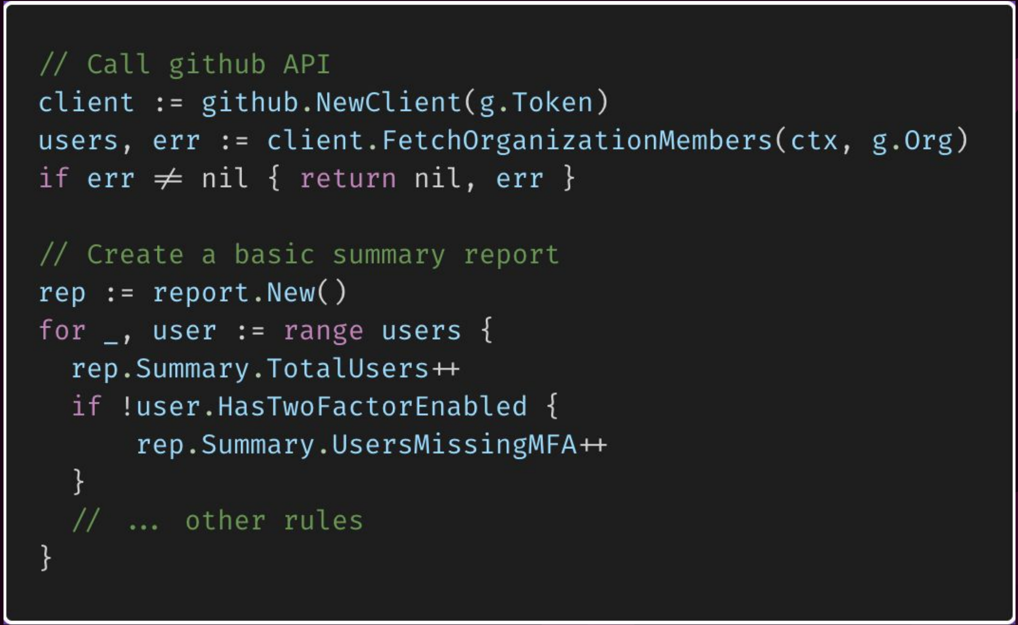
/aws/functions/checker/checker.go

CHECKER LAMBDA

#3/4 - PERFORM THE QUERY

Establish a connection to the **github graphQL API** and fetch all the members.

Run rules on the result and compile the report.



```go
// Call github API
client := github.NewClient(g.Token)
users, err := client.FetchOrganizationMembers(ctx, g.Org)
if err != nil { return nil, err }

// Create a basic summary report
rep := report.New()
for _, user := range users {
  rep.Summary.TotalUsers++
  if !user.HasTwoFactorEnabled {
      rep.Summary.UsersMissingMFA++
  }
  // ... other rules
}
```

/aws/functions/checker/checker.go

CHECKER LAMBDA

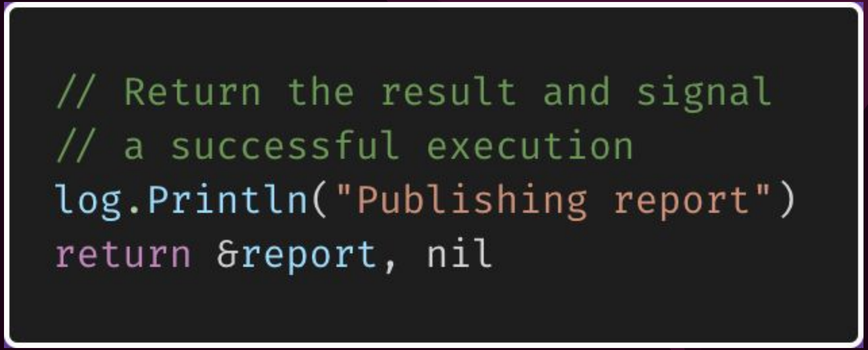
#4/4 - RETURN THE RESULT

Last thing is to return the **report**.

The **lambda destination** will automatically marshal our *struct*.

Return **error** as **nil** to signify success.

/aws/functions/checker/checker.go



```go
// Return the result and signal
// a successful execution
log.Println("Publishing report")
return &report, nil
```

THE GRAPHQL QUERY

/pkg/github/

GRAPHQL

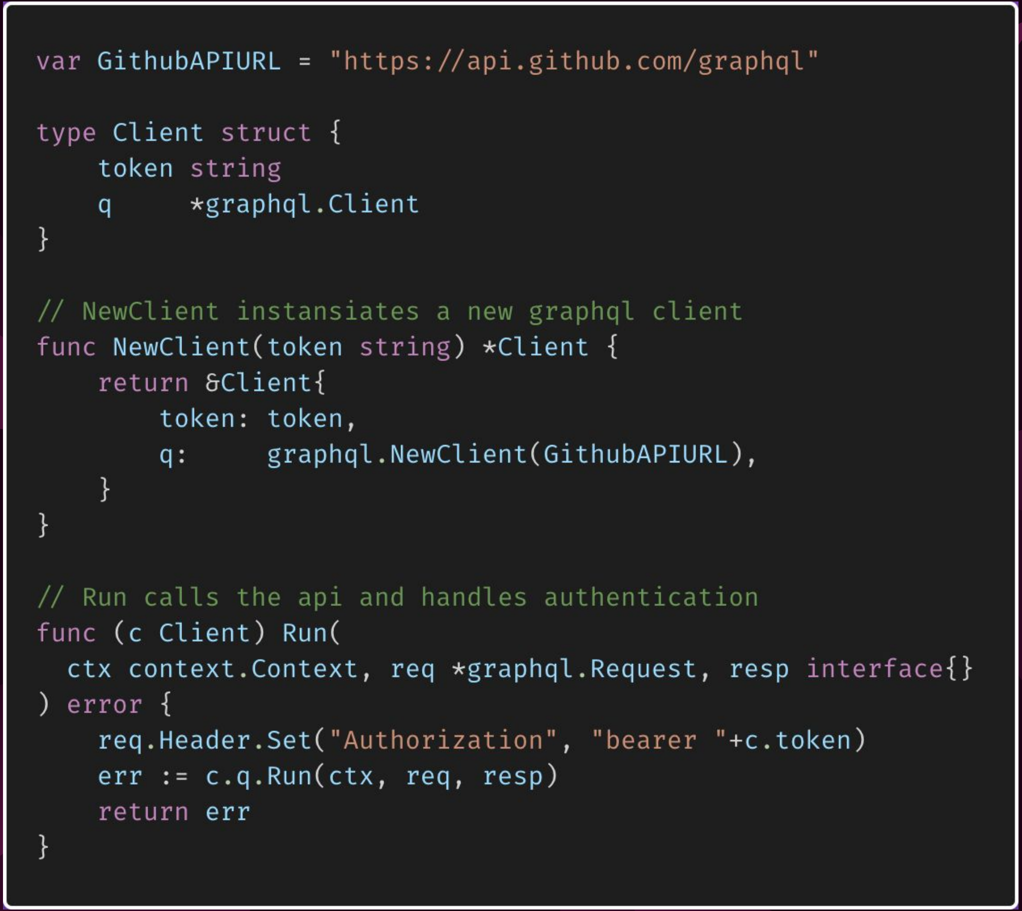
#1/2 - CONNECT

Using module **machinebox/graphql**.

Connect to single **api endpoint**.

Authenticate using an **access token** header.

/pkg/github/github.go



```go
var GithubAPIURL = "https://api.github.com/graphql"

type Client struct {
    token string
    q     *graphql.Client
}

// NewClient instansiates a new graphql client
func NewClient(token string) *Client {
    return &Client{
        token: token,
        q:     graphql.NewClient(GithubAPIURL),
    }
}

// Run calls the api and handles authentication
func (c Client) Run(
  ctx context.Context, req *graphql.Request, resp interface{}
) error {
    req.Header.Set("Authorization", "bearer "+c.token)
    err := c.q.Run(ctx, req, resp)
    return err
}
```

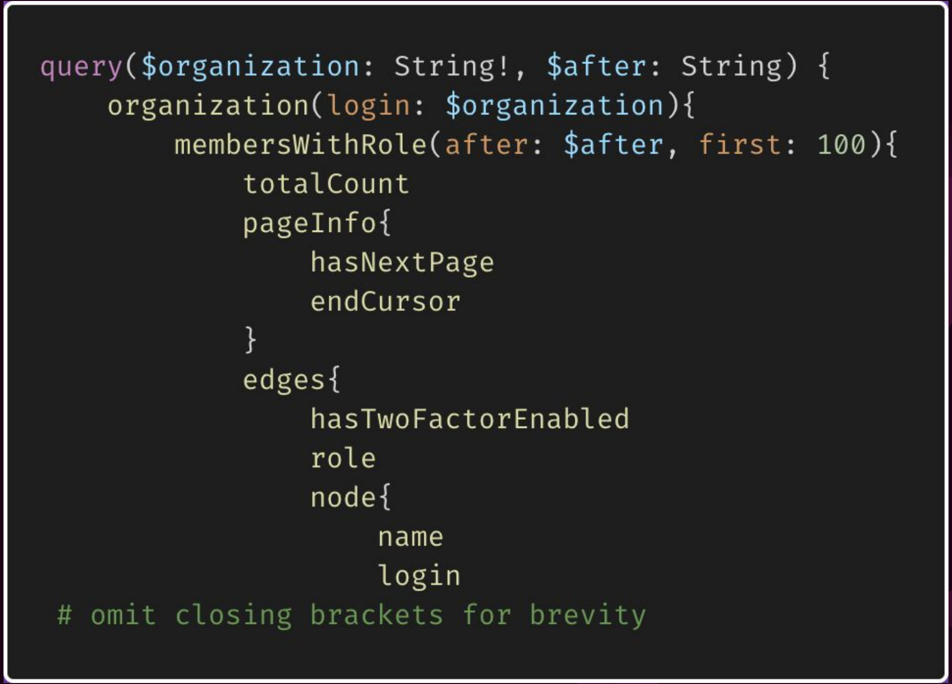
GRAPHQL

#2/2 - THE QUERY

Get organisation level details (**total users**) then pages through all the **members**.

Can pass variables in when performing the query call.

/pkg/github/members.go



```
query($organization: String!, $after: String) {
    organization(login: $organization){
        membersWithRole(after: $after, first: 100){
            totalCount
            pageInfo{
                hasNextPage
                endCursor
            }
            edges{
                hasTwoFactorEnabled
                role
                node{
                    name
                    login
 # omit closing brackets for brevity
```

SLACK NOTIFIER

/aws/functions/notifier-slack/

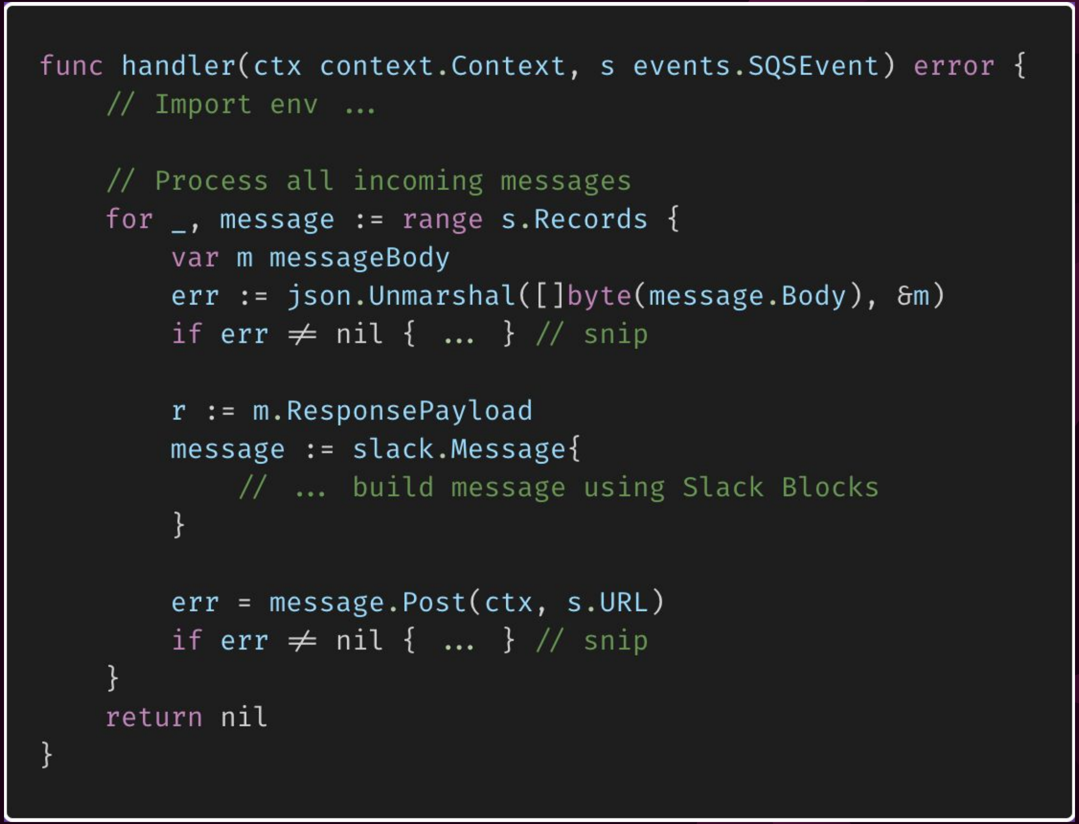
NOTIFIER

#1/2 - STRUCTURE

Structured very similarly to the **checker lambda**.

Process all **events** in loop.

Build message using **slack blocks**.



```go
func handler(ctx context.Context, s events.SQSEvent) error {
    // Import env ...

    // Process all incoming messages
    for _, message := range s.Records {
        var m messageBody
        err := json.Unmarshal([]byte(message.Body), &m)
        if err ≠ nil { ... } // snip

        r := m.ResponsePayload
        message := slack.Message{
            // ... build message using Slack Blocks
        }

        err = message.Post(ctx, s.URL)
        if err ≠ nil { ... } // snip
    }
    return nil
}
```

/aws/functions/notifier-slack/notifier-slack.go

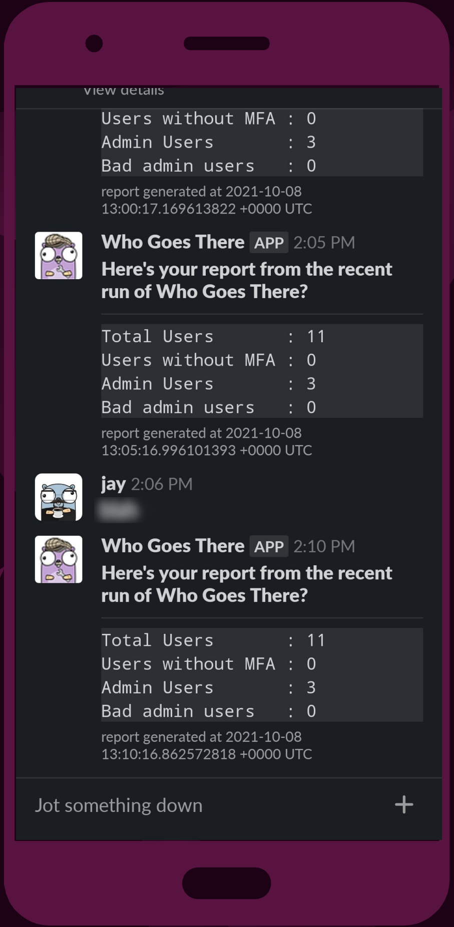
```go
message := slack.Message{
  Text: "New report from Who Goes There",
  Blocks: []*slack.MessageBlock{
    {
      Type: slack.HeaderBlock,
      Text: &slack.MessageBlockText{
        Type: slack.FormatPlainText,
        Text: "Here's your report ... ",
    }},
    { Type: slack.DividerBlock },
    {
      Type: slack.SectionBlock,
      Text: &slack.MessageBlockText{
        Type: slack.FormatMarkdown,
        Text: r.SummaryTableMarkdown(),
    }},
    {
      Type: slack.ContextBlock,
      Elements: []*slack.MessageBlockText{{
        Type: slack.FormatMarkdown,
        Text: fmt.Sprintf(" ... at %s", r.Generated),
      }},
    },
  },
}
```

```json
{
  "text": "New report from Who Goes There",
  "blocks": [
    {
      "type": "header",
      "text": {
          "type": "plain_text",
          "text": "Here's your report ... "
      }
    },
    { "type": "divider" },
    {
      "type": "section",
      "text": {
          "type": "mrkdwn",
          "text": "<code block>"
      }
    },
    {
      "type": "context",
      "elements": [{
          "type": "mrkdwn",
          "text": "generated at <time>"
      }]
    }
  ]
}
```

/aws/functions/notifier-slack/notifier-slack.go

SLACK EXAMPLE

Showing the basic report as written by the **notifier-slack** service



View details

```
Users without MFA : 0
Admin Users       : 3
Bad admin users   : 0
```

report generated at 2021-10-08 13:00:17.169613822 +0000 UTC

Who Goes There APP 2:05 PM

Here's your report from the recent run of Who Goes There?

```
Total Users       : 11
Users without MFA : 0
Admin Users       : 3
Bad admin users   : 0
```

report generated at 2021-10-08 13:05:16.996101393 +0000 UTC

jay 2:06 PM

[redacted]

Who Goes There APP 2:10 PM

Here's your report from the recent run of Who Goes There?

```
Total Users       : 11
Users without MFA : 0
Admin Users       : 3
Bad admin users   : 0
```

report generated at 2021-10-08 13:10:16.862572818 +0000 UTC

Jot something down +

DEPLOY DEMO
(IF TIME ALLOWS!)

THANKS!

You can find me at:

- **Twitter**: @n3crophonic
- **Github**: necrophonic
- **Instagram**: cafpanda

Slides: github.com/necrophonic/talks/who-goes-there

Code: github.com/necrophonic/who-goes-there

ATTRIBUTIONS

Diagrams: draw.io

Gopher avatars: gopherize.me

Theme: slidescarnival.com

Code render: carbon.now.sh

Stock images: pixabay.com

REFERENCES

GraphQL: graphql.org/

Github API: docs.github.com/en/graphql

Gu-Who: github.com/guardian/gu-who

ONS Github: github.com/onsdigital

Serverless framework: serverless.com

Altexsoft Blog: altexsoft.com/blog