



# Internship Druva

## Realization report

**Bachelor in Electronics-ICT**

Graduation subject: Cloud & Cybersecurity

Year 2022-2023

Campus: Thomas More Geel

**Milan Sterkens**

# PREFACE

My internship was at the software company Druva in Letterkenny, Ireland. Formerly called Cloudranger. It was acquired by Druva in 2018. The Cloudranger project offers simple and scalable AWS backup & recovery. It is the main product under development at the Letterkenny site.

The first weeks of the internship taught me a lot about how a modern software-company operates and, specific to Cloudranger, how their cloud-environment works.

As a Cloud & Cybersecurity student, I made it my mission to tighten the gap between developers and operations. Keeping this in mind, I did a couple of things to improve the company's pipeline and keeping their secrets from leaking.

## TABLE OF CONTENTS

<a href="#">Preface.....</a>	<a href="#">2</a>
<a href="#">Table of contents.....</a>	<a href="#">2</a>
<a href="#">List of abbreviations.....</a>	<a href="#">3</a>
<a href="#">Jest pipeline integration + Grafana dashboard.....</a>	<a href="#">3</a>
<a href="#">Plan of approach.....</a>	<a href="#">3</a>
<a href="#">The (Bash) script.....</a>	<a href="#">4</a>
<a href="#">Slack bot.....</a>	<a href="#">4</a>
<a href="#">Grafana.....</a>	<a href="#">4</a>
<a href="#">Implementation.....</a>	<a href="#">5</a>
<a href="#">The script.....</a>	<a href="#">5</a>
<a href="#">Slack bot.....</a>	<a href="#">8</a>
<a href="#">Grafana.....</a>	<a href="#">9</a>
<a href="#">Setting up.....</a>	<a href="#">9</a>
<a href="#">Creating a dashboard.....</a>	<a href="#">10</a>
<a href="#">Lambda function: Secrets &amp; PII Obfuscation.....</a>	<a href="#">12</a>
<a href="#">Result.....</a>	<a href="#">12</a>
<a href="#">SAM template.....</a>	<a href="#">13</a>
<a href="#">Code explanation.....</a>	<a href="#">14</a>
<a href="#">Code structure.....</a>	<a href="#">14</a>
<a href="#">The main program.....</a>	<a href="#">14</a>
<a href="#">Modifications in detect-secrets.....</a>	<a href="#">17</a>
<a href="#">The PII filter class.....</a>	<a href="#">18</a>
<a href="#">Conclusion.....</a>	<a href="#">23</a>
<a href="#">Bibliography.....</a>	<a href="#">24</a>

## LIST OF ABBREVIATIONS

AWS	Amazon Web Services
S3	Simple Storage Service
NPM	Node Package Manager
EC2	Elastic Compute Cloud
HTTP	Hypertext Transfer Protocol
AMI	Amazon Machine Image
HTML	HyperText Markup Language
XML	Extensible Markup Language
SSH	Secure Socket Shell
Repo	Repository

## JEST PIPELINE INTEGRATION + GRAFANA DASHBOARD

The idea is to automatically do a Jest test for the repository `CloudRanger_app_2_processing` during the Codebuild process so that a notification can be displayed in a channel on Slack. The pipeline should fail if a unit test fails and/or if the coverage percentage went down. The Slack message should contain links to the reports and explain why the build failed. Additionally, we want to have a graphical overview of the reports in Grafana.

This should eventually be applied to all repositories & environments so the code must be flexible.

### Plan of approach

My project consists of three major components:

- 1) Creating a bash script in the `buildspec yml` file of our repository to format Jest's output and get clear information out of it. We also want the pipeline to stop if test(s) fails and/or if the coverage went down.
- 2) Create a bot in Slack and make it act as a webhook.
- 3) Visualize data in Grafana.

### The (Bash) script

The script will be implemented in the existing `buildspec yml` file which is used to run a build job in AWS CodePipeline.

When Jest is run in the pipeline, I want to save its reports in files to check if failures occurred while the program was running and check the coverage percentages of the code. We upload these reports in an S3 bucket with a directory named after the repository the code originated from.

After deciding what kind of information I want to send, I will make a script that sends that data to the Slack bot. It would be nice to put hyperlinks in the message to the reports so that our developers can quickly find out what the reason is for their builds failing.

In the end, we test to see if we got any errors from Jest. If this is true, we post the notification and make the build pipeline stop. Another reason to stop the pipeline is when the current coverage percentage is lower than the previous one. If no errors were found, then the pipeline may proceed to build the code of our repository, and a message is sent to Slack to notify all tests were successful.

## Slack bot

I want to create a bot that has a webhook so I can post messages to it with the script. I will create a new channel in Slack for the notifications.

Slack bots can also use different text formats which I might use to make it more interactive.

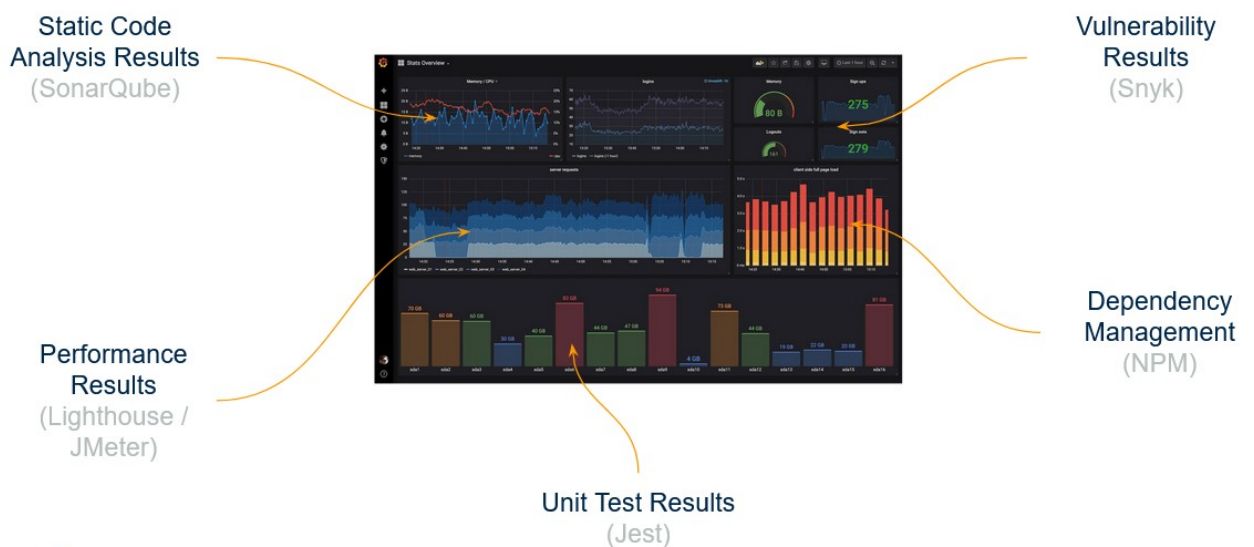
## Grafana

Grafana is one of the world's most popular and flexible data visualization tools. It is ideal for creating a simple, sensible dashboard of your data.

I was ordered to visualize the Jest reports and coverage in Grafana. The intention is to create a single pane of glass for each code repository to track all kinds of data points. Later on, other metrics for Snyk, NPM, SonarQube, and Jmeter will be added.

Here is a representation of what it should become:

## Basic Dashboard Example



Such a dashboard would be ideal for developers to get a brief understanding of what their code repositories' health looks like.

## Implementation

This chapter will explain what the result looks like & works.

### The script

This is the script that I've written in the buildspec.yml file. Note that this is just a piece of the full script and is executed right after the installation phase which installs node modules, environment variables, etc.

```
208 - node ./node_modules/jest/bin/jest.js --silent --coverage --coverageReporters=html; EXITCODE=$?
209
210 - |
211 #grab timestamp for creating unique files
212 JEST_COUNT=0
213 PREVIOUS_COVERAGE_PERCENT_STATEMENTS=0
214 TIME=$(date +%Y%m%d-%H%M%S")
215
216 #count amount of failures in report
217 JEST_COUNT=$(grep "<failure>" ./jest/report.xml | wc -l)
218
219 aws s3 cp s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/previous-coverage-percentage.txt ./jest/previous-coverage-percentage.txt
220 PREVIOUS_COVERAGE_PERCENT_STATEMENTS=$(awk '{ printf "%s", $0 }' ./jest/previous-coverage-percentage.txt || echo 0)
221 COVERAGE_PERCENT_STATEMENTS=$(grep -i "%%" jest/index.html | awk 'NR==1{print $2}' | sed 's/.*/>;s%/')
222 COVERAGE_PERCENT_BRANCHES=$(grep -i "%%" jest/index.html | awk 'NR==2{print $2}' | sed 's/.*/>;s%/')
223 COVERAGE_PERCENT_FUNCTIONS=$(grep -i "%%" jest/index.html | awk 'NR==3{print $2}' | sed 's/.*/>;s%/')
224 COVERAGE_PERCENT_LINES=$(grep -i "%%" jest/index.html | awk 'NR==4{print $2}' | sed 's/.*/>;s%/')
225
226 echo $COVERAGE_PERCENT_STATEMENTS > jest/coverage-percentage.txt
227
228 #send report and coverage to their buckets
229 aws s3 cp ./jest/coverage-percentage.txt s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/previous-coverage-percentage.txt
230 aws s3 cp ./jest/report.xml s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/jest-report-$TIME.xml
231 aws s3 cp ./jest/report.xml s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/jest-report-latest.xml
232 aws s3 cp ./jest/ s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/jest-coverage-$TIME/ --recursive --exclude "report.xml"
233 aws s3 cp ./jest/ s3://cloudranger-dev-jest-reports/$codebuild_ENV_GIT_REPO/jest-coverage-latest/ --recursive --exclude "report.xml"
234
235 #creating url vars to access the files
236 URL_COVERAGE="https://c[REDACTED]amazonaws.com/$codebuild_ENV_GIT_REPO/jest-coverage-$TIME/index.html"
237 URL_REPORT="https://c[REDACTED]amazonaws.com/$codebuild_ENV_GIT_REPO/jest-report-$TIME.xml"
238 URL_GRAFANA="http://44.200.100.101:3000/d/j0wqzCYVv/codehealth-dashboards?orgId=1&var-repo=$codebuild_ENV_GIT_REPO&kiosk"
239
240 #uncomment the next line to force JEST_COUNT to be 0 (force a succesfull Slack message)
241 #JEST_COUNT=0
242 #COVERAGE_PERCENT_STATEMENTS="10"
243 #PREVIOUS_COVERAGE_PERCENT_STATEMENTS="10"
244
245 #install maths package bc
246 apt update
247 apt install bc -y
248
249 #send messages to the slack bot depending on the amount of failures Jest found and if the coverage went down.
250 if [ $JEST_COUNT = 0 ] && [ $(echo "if (${COVERAGE_PERCENT_STATEMENTS}>=${PREVIOUS_COVERAGE_PERCENT_STATEMENTS}) 1 else 0" | bc) = 1 ]; then
251     echo "All tests completed SUCCESSFULLY"
252     curl -X POST \
253     -H 'Content-type: application/xml' \
254     -d "${\"blocks\": [ {\"type\": \"section\\\", \"text\": {\"type\": \"mrkdwn\\\", \"text\": \"=====
255
256 elif [ $JEST_COUNT != 0 ] && [ $(echo "if (${COVERAGE_PERCENT_STATEMENTS}>=${PREVIOUS_COVERAGE_PERCENT_STATEMENTS}) 1 else 0" | bc) = 1 ]; then
257     curl -X POST \
258     -H 'Content-type: application/json' \
259     -d "${\"blocks\": [ {\"type\": \"section\\\", \"text\": {\"type\": \"mrkdwn\\\", \"text\": \"=====
260     exit 1
261 elif [ $JEST_COUNT = 0 ] && [ $(echo "if (${COVERAGE_PERCENT_STATEMENTS}<=${PREVIOUS_COVERAGE_PERCENT_STATEMENTS}) 1 else 0" | bc) = 1 ]; then
262     curl -X POST \
263     -H 'Content-type: application/json' \
264     -d "${\"blocks\": [ {\"type\": \"section\\\", \"text\": {\"type\": \"mrkdwn\\\", \"text\": \"=====
265     exit 1
266 else
267     curl -X POST \
268     -H 'Content-type: application/json' \
269     -d "${\"blocks\": [ {\"type\": \"section\\\", \"text\": {\"type\": \"mrkdwn\\\", \"text\": \"=====
270     exit 1
271 fi
```

### Explanation of script

Line number: Explanation

208: Run the unit tests and create coverage files for it. `EXITCODE=$?` Is used to ignore an exit code that occurs when there are failures because we want to decide to exit the Codebuild ourselves later in the script.

212-214: Predefine some variables to make the code more stable. Also, create a `TIME` variable to create a unique identifier for the files that will be uploaded to the S3 bucket.

217: We count the number of failures reported in `report.xml`.

219-224: First, we copy the file `previous-coverage-percentage.txt` from the S3 bucket to a local file. After this, we store the value of that file in a variable `PREVIOUS-COVERAGE-PERCENT-STATEMENTS`. If the file didn't exist in the bucket, we store a 0 in that variable. Finally, we store the four different coverage percentages in variables with matching names by doing text transformations in the `index.html` file of the coverage generated by Jest.

226: We store the main coverage percentage (statements) in a `coverage-percentage.txt` file so we can send it later to the S3 bucket and overwrite the `previous-coverage-percentage.txt` within.

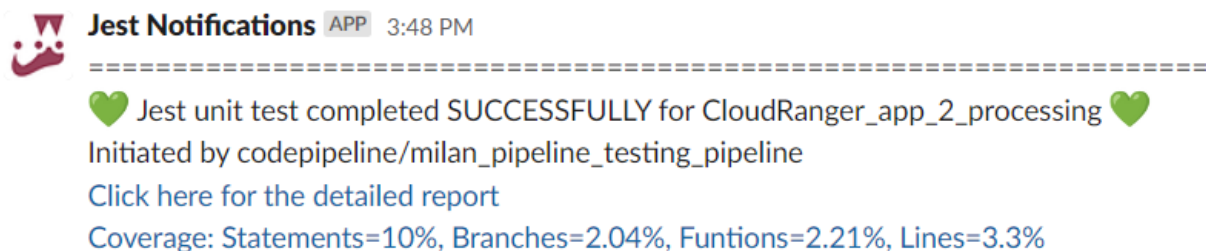
229-233: copies all the output generated from Jest to the S3 bucket. It also overwrites all the `*-latest` files. These are used for Grafana to show the latest reports in the dashboard.

236-238: URL variables are generated so they can be used later in the curl command to post a message to Slack.

247: We install the math package `bc` in to do math operations with floating point numbers. In our case, these are the coverage percentages that we'll use for the equations in our conditionals below.


250-271:

- First statement: If no failures were found AND the latest coverage percentage is higher than the previous one, an HTTP post message will be sent to our webhook which will show this message:



- Second statement: If one or more failures were found AND the latest coverage percentage is higher than the previous one, an HTTP post message will be sent

to our webhook which will show this message:

 **Jest Notifications** APP 4:37 PM  
=====

! Jest unit test FAILED for CloudRanger\_app\_2\_processing !  
REASON: 11 test(s) failed  
Initiated by codepipeline/milan\_pipeline\_testing\_pipeline  
[Click here for the detailed report](#)  
Coverage: ▲ Statements=3.24%, Branches=2.06%, Functions=2.21%, Lines=3.29%  
[Grafana Dashboard](#)

- Third statement: If no failures were found AND the latest coverage percentage is lower than the previous one, an HTTP post message will be sent to our webhook which will show this message:

=====

! Jest unit test FAILED for CloudRanger\_app\_2\_processing !  
REASON: Coverage went down  
Initiated by codepipeline/milan\_pipeline\_testing\_pipeline  
[Click here for the detailed report](#)  
Coverage: ▼ Statements=3.25%, Branches=2.04%, Functions=2.21%, Lines=3.3%  
[Grafana Dashboard](#)

- Last statement: If one or more failures were found AND the latest coverage percentage is lower than the previous one, an HTTP post message will be sent to our webhook which will show this message:

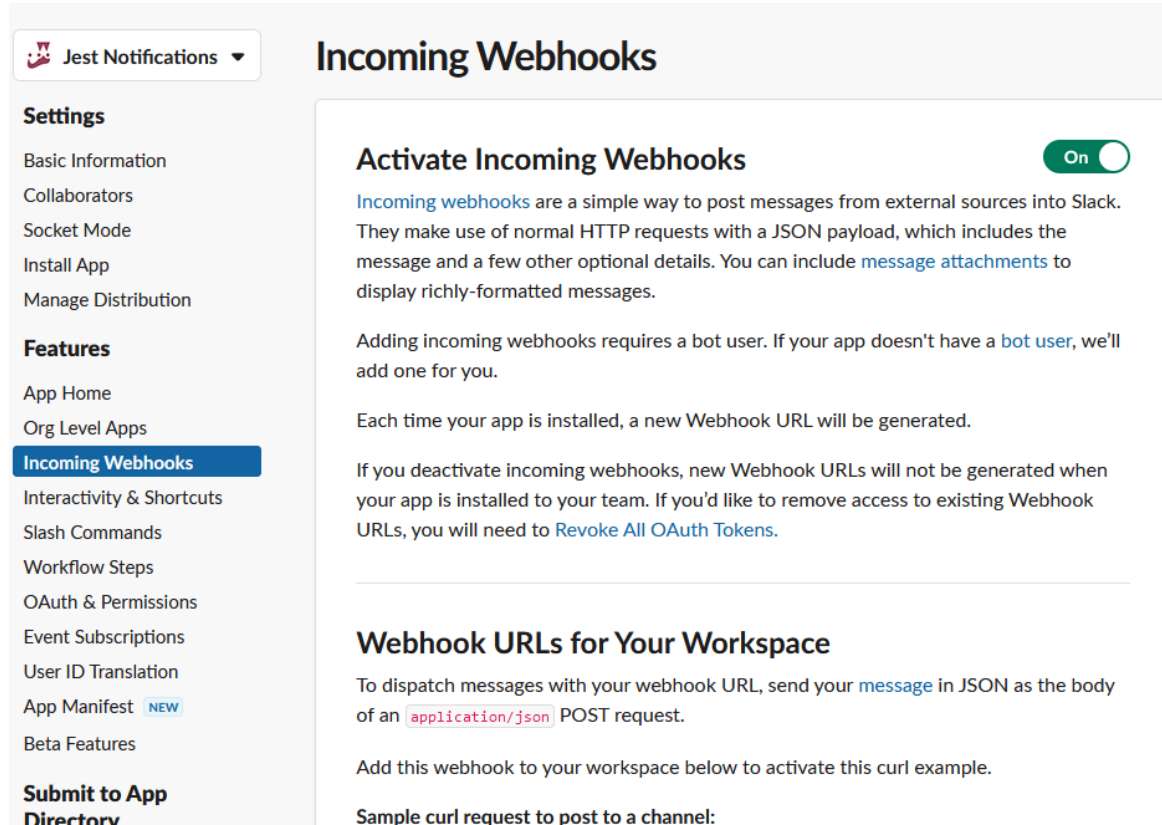
=====

! Jest unit test FAILED for CloudRanger\_app\_2\_processing !  
REASON: 11 Test(s) failed AND coverage went down  
Initiated by codepipeline/milan\_pipeline\_testing\_pipeline  
[Click here for the detailed report](#)  
Coverage: ▼ Statements=3.24%, Branches=2.06%, Functions=2.21%, Lines=3.29%

## Slack bot

Creating a bot in Slack was quite the easiest part of this task. After logging in to [api.slack.com](https://api.slack.com), you can create a new "app". You have the option to create one from scratch or to use a template. I chose to create one from scratch.

After doing that you'll be redirected to the app configuration page where you can customize the bot to your liking.



**Jest Notifications** ▾

## Incoming Webhooks

**Settings**

- Basic Information
- Collaborators
- Socket Mode
- Install App
- Manage Distribution

**Features**

- App Home
- Org Level Apps
- Incoming Webhooks**
- Interactivity & Shortcuts
- Slash Commands
- Workflow Steps
- OAuth & Permissions
- Event Subscriptions
- User ID Translation
- App Manifest NEW
- Beta Features

**Submit to App Directory**

### Activate Incoming Webhooks On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include [message attachments](#) to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a [bot user](#), we'll add one for you.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to [Revoke All OAuth Tokens](#).

---

### Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your [message](#) in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

**Sample curl request to post to a channel:**

At the incoming webhooks tab, you can enable "activate incoming webhooks" and generate one at the bottom of the page. Choose the Slack channel where the bot should post the messages. A webhook URL will look like this:

<https://hooks.slack.com/services/T02abcd2L5/B04abcdeS2DV/Babcd3rYmbmXIgrabcdefgEEVzNV>

I used such an URL to post the Slack messages with the curl command.

## Grafana

It was very straightforward to set up Grafana on an ec2 instance. It surprised me how easy it was to parse single files using the Infinity plugin.

## Setting up

First, I launched an Amazon Linux instance with a security group that allows port 22 for SSH access and port 3000 for the Grafana portal. Then, I connected to it via SSH using the key I specified during setup.

I didn't use APT to install Grafana since Amazon Linux AMI comes with the YUM package manager. Before installing the package, I had to add the Grafana repository to its source list. After that, I could install the open-source version of Grafana using the following command:

```
sudo yum update && sudo yum install grafana -y
```

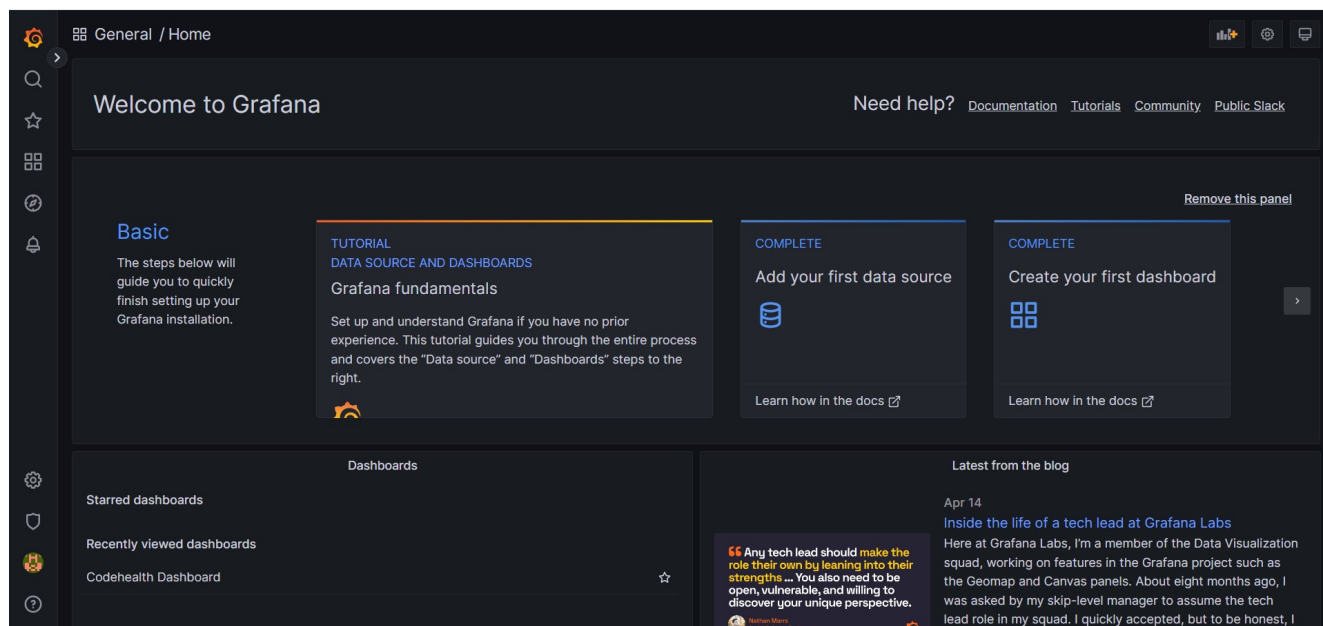


Note: If we want to switch to the enterprise version, we just substitute grafana to grafana-enterprise in the command.

I used the Grafana Infinity datasource plugin to parse different file formats like XML and HTML and use them as data points. It can be installed like this:

`grafana-cli plugins install yesoreyeram-infinity-datasource && sudo systemctl restart grafana`

After installation, I could surf to the Grafana portal on <http://<ip-adres>:3000>, log in using the username admin and password admin, and change the password for something safer.



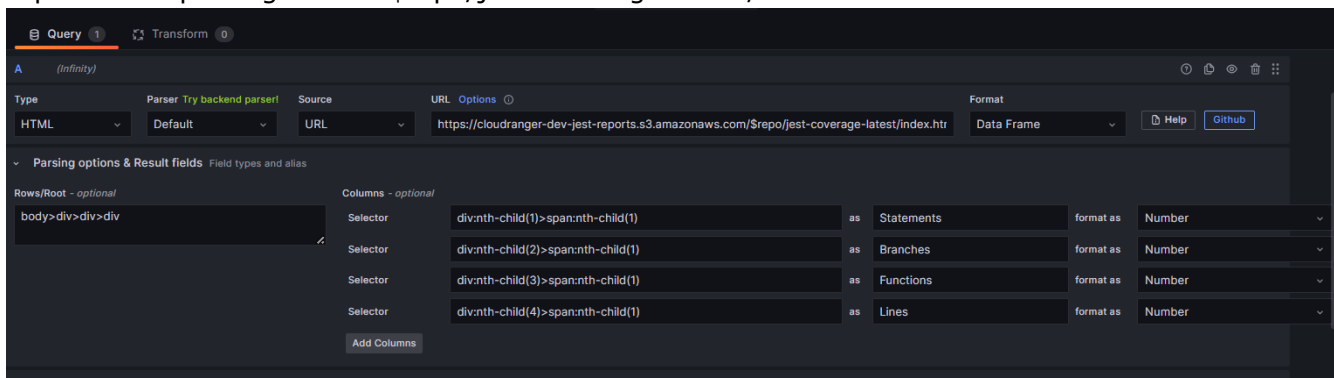
## Creating a dashboard

Before creating a dashboard, I had to tell Grafana to use the Infinity plugin as a datasource. I did this by hovering over the gear icon (bottom left) and clicking on the datasources link. Then I came to the datasources page where I could add the Infinity plugin.

After clicking on the four squares on the left, I could create my first dashboard. There I created two panels to show the Jest report statistics and a panel to show the coverage summary. I made the dashboard more dynamic by adding a repository and stage variable which you can change by clicking on it. The variables will change the query location to a different folder in the S3 bucket we created for our reports.

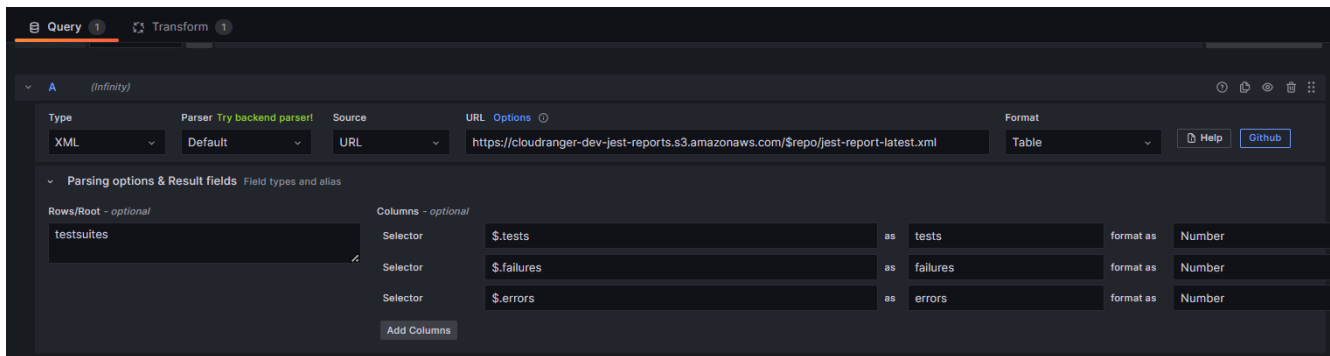


The first panel shows a bar chart using the latest coverage percentages of the chosen repository folder. These are the query settings to get the data out of the coverage report. I am parsing the file `$repo/jest-coverage-latest/index.html`:



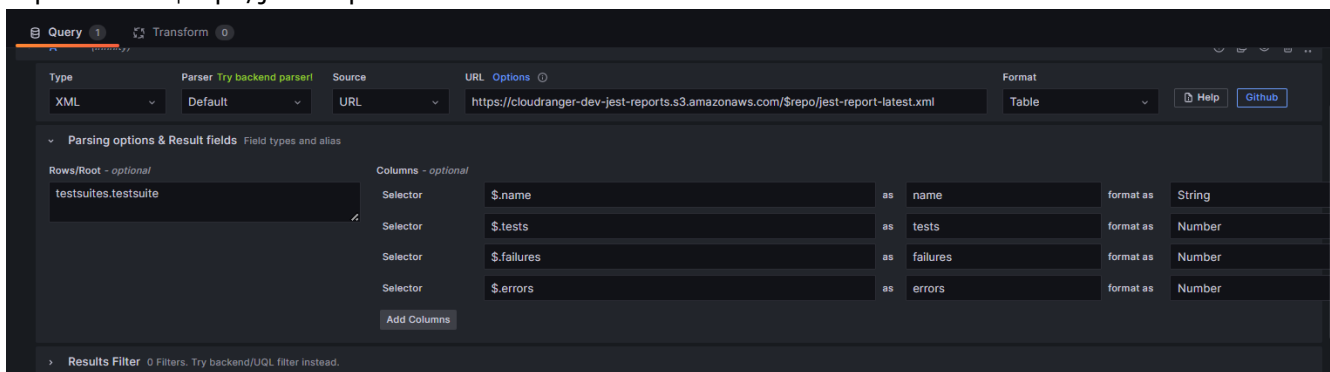
As you can see, we select the different percentages using HTML selectors.

The second panel is more a summary of successful tests vs failures. These are the query settings to get the data out of the report from `$repo/jest-report-latest.xml`:



We use "\$." in our parsing selectors because it is used to retrieve the XML attributes for the root element we declared on the left.

The third panel is a more detailed version of the second panel. It shows the numbers for all testsuites separately. These are the query settings to get the data out of the report from \$repo/jest-report-latest.xml:



Same thing as the second panel but going one level deeper to the testsuite level as root.

## LAMBDA FUNCTION: SECRETS & PII OBFUSCATION

My assignment was to create a Lambda function that detects and obfuscates secrets and Personal Identifiable Information in each string. This function will filter a query that originates from a chatbot in Slack. This document explains how it works.

### Result

Input in AWS:

```

Event JSON
1 - {
2   "prompt": "password = 'mysecretpassword'\napi_key = 'sk_test_1234567890abcdefghijklmnopqrstuvwxyz'\naws_secret_key = 'AKIAIOSFODNN7EXAMPLE'\njwt_secret = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX
3 }
  
```

=

```
Unset ▾ ↻
{
  "prompt": "password = 'mysecretpassword'
  api_key = 'sk_test_1234567890abcdefghijklmnopqrstuvwxyz'
  naws_secret_key = 'AKIAIOSFODNN7EXAMPLE'
  jwt_secret =
  'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaWbmFtZSI6IkpvaG4gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c' "
}
```

Output:

✔ Executing function: succeeded ([logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": [
    {
      "response": "password = '<SECRET_KEYWORD>'\napi_key = '<SECRET>'\naws_secret_key = '<AWS_KEY>'\njwt_secret = '<SECRET_KEYWORD>'",
      "found_secret": true,
      "found_pii": true
    }
  ]
}
```

### SAM template

*template.yml* deploys a Lambda function with 1024MB of memory which was the best size for running the code.

```
! template.yaml
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3  Description: Lambda function for secret detection and replacement to gibberish
4
5  Resources:
6    SecretDetectionFunction:
7      Type: AWS::Serverless::Function
8      Properties:
9        CodeUri: src/
10       Handler: main.lambda_handler
11       Runtime: python3.10
12       Description: Lambda function for secret detection and replacement to gibberish
13       Timeout: 3
14       MemorySize: 1024
```

### Code explanation

#### Code structure

- The main program in *main.py*
- A slightly modified version of the [detect-secrets](#) GitHub repository
- A Python class *PIIFilter.py*

## The main program

The first four lines of code import the necessary libraries and classes including detect-secrets and PII filter classes.

```
Python
from detect_secrets import SecretsCollection
from detect_secrets.settings import default_settings
import json
from PIIFilter import PIIClass
```

After that, we create objects of those classes and initialize the piifilter.

```
Python
piifilter = PIIClass()
piifilter.initialize()

secrets = SecretsCollection()
```

Then, the function `lambda_handler` is declared which will be executed when the Lambda function is invoked. In that function, two booleans called `found_secret` and `found_pii` are declared. These are used later to tell if we detected secrets or PII in the query the program will inspect. We save that query in a variable called `prompt` which is passed to `event['prompt']` when invoking the Lambda function.

```
Python
def lambda_handler(event, context):
    found_secret=False
    found_pii=False
    prompt=event['prompt']
```

Because the `detect-secrets` library doesn't support scanning string variables, but does support scanning files, the query is stored in `/tmp/text.txt` and scanned with `detect-secrets' scan_file()` method.

```
Python
f = open("/tmp/test.txt", "w")
f.writelines(prompt)
f.close()

with default_settings():
    secrets.scan_file('/tmp/test.txt')
```

Then, the `filter()` method of `PIIClass` is used to obfuscate personal identifiable information of the prompt string and is put in a new response variable. After that, the

boolean `found_pii` is set to `True` if the method found something.

```
Python
response=piifilter.filter(prompt)

if response != prompt:
    found_pii=True
```

Now, if `detect-secrets` found any secrets in `/tmp/text.txt`, it will set `found_secret` to `True` and replace the secrets with a replacement value which is given in the json output of `detect-secrets`' `json()` method. This replacement value is part of the modification I did to the `detect-secrets` library.

```
Python
if secrets.json():
    print("found secret(s)")
    for secret in secrets.json()["/tmp/test.txt"]:
        response=response.replace(secret["secret_value"],
        secret["replacement_value"])
    found_secret=True
else:
    print("found no secrets")

print(response)
```

Finally, we jsonify the `response`, `found_secret`, and `found_pii` together and return it in the body of the Lambda response.

```
Python
json_data = [{"response": response,
              "found_secret": found_secret,
              "found_pii": found_pii}]

return {
    'statusCode': 200,
    'body': json_data
}
```

Complete code:

```

src > main.py > ...
1  from detect_secrets import SecretsCollection
2  from detect_secrets.settings import default_settings
3  import json
4  from PIIFilter import PIIClass
5
6
7  piifilter = PIIClass()
8  piifilter.initialize()
9
10 secrets = SecretsCollection()
11
12 def lambda_handler(event, context):
13     found_secret=False
14     found_pii=False
15     prompt=event['prompt']
16
17     f = open("/tmp/test.txt", "w")
18     f.writelines(prompt)
19     f.close()
20
21     print(prompt)
22
23     with default_settings():
24         secrets.scan_file('/tmp/test.txt')
25
26     #print(json.dumps(secrets.json(), indent=2))
27
28
29     response=prompt
30
31     if secrets.json():
32         print("found secret(s)")
33         for secret in secrets.json()["/tmp/test.txt"]:
34             response=response.replace(secret["secret_value"],secret["replacement_value"])
35         found_secret=True
36     else:
37         print("found no secrets")
38
39     response=piifilter.filter(response)
40
41     if response != prompt:
42         found_pii=True
43
44     print(response)
45
46     json_data = [{"response": response,
47                 "found_secret": found_secret,
48                 "found_pii": found_pii}]
49
50     return {
51         'statusCode': 200,
52         'body': json_data
53     }

```

## Modifications in detect-secrets

The json() method we used in the main program outputted a json which included a replacement value. This value did not exist in the first place, so some modifications were necessary to potential\_secret.py of the detect-secrets library.

I used a match statement to add a replacement\_value attribute to the attributes dictionary with a value based on the secret type. I marked my additions in yellow below.

json() method in potential\_secret.py:

```
Python ↻  
def json(self) -> Dict[str, Union[str, int, bool]]:  
    """Custom JSON encoder"""  
    attributes: Dict[str, Union[str, int, bool]] = {  
        'type': self.type,  
        'filename': self.filename,  
        'hashed_secret': self.secret_hash,  
        'is_verified': self.is_verified,  
        'is_secret': self.is_secret,  
        'secret_value': self.secret_value,  
    }  
  
    match self.type:  
        case "AWS Access Key":  
            attributes['replacement_value'] = '<AWS_KEY>  
        case "Secret Keyword":  
            attributes['replacement_value'] = '<SECRET_KEYWORD>  
        case "Hex High Entropy String":  
            attributes['replacement_value'] =  
'<HIGH_ENTROPY_STRING>  
  
        case default:  
            attributes['replacement_value'] = '<SECRET>  
  
    if hasattr(self, 'line_number') and self.line_number:  
        attributes['line_number'] = self.line_number  
  
    if hasattr(self, 'is_secret') and self.is_secret is not None:  
        attributes['is_secret'] = self.is_secret  
  
    return attributes
```

## The PII filter class

I tried to use the Scrubadub Python package in the first place, but this was impossible due to the heavy dependency requirements which made the package larger than 300MB. Since Lambda functions cannot be larger than 250MB, I decided to create my own Python class which should work just as well as Scrubadub, if not even better. In this chapter, I explain the methods of the class.

These three methods are used to replace phone numbers from Irish, U.S., and Indian phone numbers with a placeholder <PHONE>. The methods use regex to detect phone numbers. These methods are used in the filter() method.



Python

```
@staticmethod
def remove_phone_numbers_ie(text, numbers_to_zero):
    """replace Irish phone numbers"""
    if numbers_to_zero:
        return re.sub('\d', '0', text).strip()
    else:
        return re.sub("(\\(?\\d{3}\\)?[ -]*(\\d{3}[ -]*)\\d{4})",
'<PHONE>', text).strip()

@staticmethod
def remove_phone_numbers_us(text, numbers_to_zero):
```

```
    if numbers_to_zero:
        return re.sub('\d', '0', text).strip()
    else:
        return
re.sub("(?:(?:\+?1\s*(?:[.-]\s*)?)?(?:\(\s*([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9])\s*\)|([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9]))\s*(?:[.-]\s*)?([2-9]1[02-9]|[2-9][02-9]1|[2-9][02-9]{2})\s*(?:[.-]\s*)?([0-9]{4})", '<PHONE>', text).strip()

@staticmethod
def remove_phone_numbers_in(text, numbers_to_zero):
    """replace Indian phone numbers"""
    if numbers_to_zero:
        return re.sub('\d', '0', text).strip()
    else:
        return re.sub("((\+*)((0[ -]*)*|((91
)*)((\d{12})+|(\d{10})+))|\d{5}([- ]*)\d{6}", '<PHONE>',
text).strip()
```

This method is used to replace timestamps to a placeholder <TIME>. It also uses regex to detect it and is used in the filter() method.

Python

```
@staticmethod
def remove_times(text):
    return re.sub('(\\d{1,2})[.:](\\d{1,2})?([ ])?(am|pm|AM|PM))?',
'<TIME>', text)
```

This method is used to replace dates to a placeholder <DATE>. It also uses regex to detect it and is used in the filter() method.

Python

```
@staticmethod
```

```
def remove_dates(text):
```

```
    text = re.sub("\d{2}[- /.]\d{2}[- /.]\d{4}", "<DATE>",  
text)
```

```
    text = re.sub(
```

```
"(\d{1,2}[\^w]{,2}(Januari|Februari|March|April|May|June|July|Aug  
ust|September|October|November|December)"
```

```
    "([- /.]{,2}(\d{4}|\d{2}))?",
```

```
    "<DATE>", text)
```

```
    text = re.sub(
```

```
"(\d{1,2}[\^w]{,2}(jan|feb|mar|apr|may|jun|jul|aug|sep|okt|nov|de  
c))[- /.](\d{4}|\d{2})?",
```

```
    "<DATE>", text)
```

```
    return text
```

This method is used to replace email-adresses to a placeholder <EMAIL>. It also uses regex to detect it and is used in the filter() method.

Python

```
@staticmethod
```

```
def remove_email(text):
```

```
    return
```

```
re.sub("((([a-zA-Z0-9_]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66}  
))\.[a-z]{2,6}(?:\.[a-z]{2})?))"
```

```
    "(?![^\<]*>)",
```

```
    "<EMAIL>",
```

```
    text)
```

This method is used to replace urls to a placeholder <URL>. It also uses regex to detect it and is used in the filter() method.

Python

```
def remove_url(self, text):
```

```
    text = re.sub(self.url_re, "<URL>", text)
```

```
    return text
```

This method is used to replace postal codes to a placeholder <POSTCODE>. It also uses regex to detect it and is used in the filter() method.

```
Python
@staticmethod
def remove_postal_codes(text):
    return re.sub(r"\b([0-9]{4}\s?[a-zA-Z]{2})\b", "<POSTCODE>",
text)
```

This method is used to remove accents.

```
Python
@staticmethod
def remove_accents(text):
    text = unicodedata.normalize('NFD', text).encode('ascii',
'ignore')
    return str(text.decode("utf-8"))
```

The filter() method applies the previous methods. So you can choose those you want to be applied.

```
Python
def filter(self, text):
    """Filters PII out of the given string"""
    #text = self.remove_url(text)
    #text = self.remove_accents(text)
    text = self.remove_email(text)

    #text = self.remove_dates(text)
    #text = self.remove_times(text)
    text = self.remove_postal_codes(text)
```

```

    text = self.remove_phone_numbers_ie(text,
self.numbers_to_zero)
    text = self.remove_phone_numbers_us(text,
self.numbers_to_zero)
    text = self.remove_phone_numbers_in(text,
self.numbers_to_zero)
    return text
```



```

37 @staticmethod
38 def remove_phone_numbers_in(text, numbers_to_zero):
39     """replace Indian phone numbers"""
40     if numbers_to_zero:
41         return re.sub('\d', '0', text).strip()
42     else:
43         return re.sub("((\+*)((0[ -]*)|((91 )*))((\d{12})+|(\d{10}+))|\d{5}([- ]*\d{6})", '<PHONE>', text).strip()
44
45 @staticmethod
46 def remove_times(text):
47     return re.sub('(\d{1,2})[:-](\d{1,2})?[ ]?(am|pm|AM|PM))?', '<TIME>', text)
48
49 @staticmethod
50 def remove_dates(text):
51     text = re.sub("\d{2}[- ./]\d{2}[- ./]\d{4}", "<DATE>", text)
52
53     text = re.sub(
54         "(\d{1,2}[\^w]{,2}(Januari|Februari|March|April|May|June|July|August|September|October|November|December))"
55         "([- ./,;]{,2}(\d{4}|\d{2}))?",
56         "<DATE>", text)
57
58     text = re.sub(
59         "(\d{1,2}[\^w]{,2}(jan|feb|mar|apr|may|jun|jul|aug|sep|okt|nov|dec))[- ./](\d{4}|\d{2})?",
60         "<DATE>", text)
61     return text
62
63 @staticmethod
64 def remove_email(text):
65     return re.sub("([a-zA-Z0-9_+]{,25}[\^w-]+)?(?:\.[\^w-]+)*@((?:[\^w-]+\.)*\w{,66})\.[a-z]{2,6}(?:\.[a-z]{2})?)",
66         "(?![^<*>])",
67         "<EMAIL>",
68         text)
69
70 def remove_url(self, text):
71     text = re.sub(self.url_re, "<URL>", text)
72     return text
73
74 @staticmethod
75 def remove_postal_codes(text):
76     return re.sub(r"\b([0-9]{4})\s?[a-zA-Z]{2}\b", "<POSTCODE>", text)
77
78 @staticmethod
79 def remove_accents(text):
80     text = unicodedata.normalize('NFD', text).encode('ascii', 'ignore')
81     return str(text.decode("utf-8"))
82
83 def filter(self, text):
84     """Filters PII out of the given string"""
85     #text = self.remove_url(text)
86     #text = self.remove_accents(text)
87     text = self.remove_email(text)
88
89     #text = self.remove_dates(text)
90     #text = self.remove_times(text)
91     text = self.remove_postal_codes(text)
92
93     text = self.remove_phone_numbers_ie(text, self.numbers_to_zero)
94     text = self.remove_phone_numbers_us(text, self.numbers_to_zero)
95     text = self.remove_phone_numbers_in(text, self.numbers_to_zero)
96     return text
97

```

## CONCLUSION

I feel like I made a difference at Druva. The unit test integration is going to be implemented in all their repositories and the Lambda function is also running in production. I think that it makes the life of the developers easier & safer. It was also nice to be able to help another intern and that I could help my other colleagues on certain tasks.

I learned a lot about AWS in general. During my internship, I had the opportunity to use AWS courses on Udemy to learn more about the different services they offer. The Codebuild part took most of my time with the Dash script for the buildspec file. The funny thing is that I thought that the Amazon Linux AMI image used Bash. Apparently, it uses Dash which meant I had to rewrite the script to be POSIX compliant.

## BIBLIOGRAPHY

THE GRAFANA DOCS: [HTTPS://GRAFANA.COM/DOCS/GRAFANA/LATEST/](https://grafana.com/docs/grafana/latest/)

SLACK DOCS: [HTTPS://API.SLACK.COM/DOCS](https://api.slack.com/docs)

AWS DOCS: [HTTPS://DOCS.AWS.AMAZON.COM/](https://docs.aws.amazon.com/)

Detect-secrets Github: <https://github.com/Yelp/detect-secrets>