GSoC
2020

Google Summer of Code 2020

Prometheus

Rule formatting tool and support for langserver

Personal Information

Name:	Patel Drumil
Github:	drumilpatel2000
Email:	drumilpatel720@gmail.com
Phone No:	+91-6351173421
Location:	India
Time Zone:	India (UTC +5:30)
Riot:	Flipbyte
LinkedIn:	Drumil Patel

University Information

University:	Indian Institute of Technology, Roorkee
Majors:	Electrical Engineering
Current:	IInd Year (batch 2022)
Degree:	Bachelor of Technology (4 Year Program)

Contact and Working hours

Reachable anytime through **email**, **riot** or **contact number**

Typical working hours :-

- UTC 0430 - 0730 hrs (IST 1000 - 1300 hrs)
- UTC 0930 - 1230 hrs (IST 1500 - 1800 hrs)
- UTC 1530 - 2030 hrs (IST 2100 - 0200 hrs)

Coding Skills

Programming Languages:

- Proficient **PHP**, **Python, Go, Javascript**
- Intermediate Knowledge in **C++**, **Java**, **Typescript**
- Worked with **Django**, **React**, **Laravel**, **Yii 2.0**, **Symfony**, **Figma**
- Databases - **MySQL**, **PostgreSQL, MongoDB**
- Devops - **Prometheus, Kubernetes, Grafana**

Experience

Full Stack Web Developer and Devops Engineer, currently Hub-Coordinator at Information Management Group (IMG), IIT Roorkee. During my first year at college my workspace was confined to web development, during which I was selected for GSoC 2019 to work on **Automating configuration for Assignment uploads** project in Submitty, in my second year my workspace expanded to Cloud Computing and some major projects listed in the following section.

My Projects:

- **CMS** :-
  CMS is abbreviation for Content Management System which is manages more than 6000 static pages of the official website of IIT Roorkee, the use of a Content Management system is to make a template for certain type of page (say news announcement page) so that anyone without coding knowledge (say some staff member or professor) can make changes to all those pages of that particular type by just changing the base template and publish changes.

  ❖ Feature additions to the Institute **CMS** (23 feature commits)
  ❖ Role: Backend developer
  ❖ CMS handles the official website of IITR website
  ❖ It manages more than 6000 static pages on the website
  ❖ Tech Stack includes **Yii 2.0**, **PostgreSQL**, **RabbitMq**, **Apache2**

- **Forminator** :
  Forminator also known as google forms of IIT Roorkee. Main aim behind Forminator is to have access and authorization based on certain qualifications. Forminator is used by instructors and other corresponding authorities to abstract information from students having certain institutional based qualifications.
    - ❖ The official Forms app of IIT Roorkee
    - ❖ Role: Full stack developer
    - ❖ Added support for OAuth based authorization
    - ❖ Tech Stack: **Laravel, Django Rest Framework**

- **Lectures and Tutorials Portal(LecTut):**
  LecTut is the study portal, used by unique 4k campus students and faculty every month with the aim of assisting them to achieve their academic goals. Implemented new design using react and redux
    - ❖ Open Source Login Template
    - ❖ Role: Repository creator and maintainer (22 commits)
    - ❖ Includes Form validation
    - ❖ Implemented new design
    - ❖ Tech stack: **Django, Django Rest Framework, React, Redux**

- **Cortago:**
  Cortago is MVC based backend framework written in Golang. Main aim behind development of cortago is to add inbuilt support for RBAC(Role-based access control) in an MVC framework.
    - ❖ Role: Maintainer (17 Commits)
    - ❖ Code can be viewed here
    - ❖ Presently Cortago is in its development stage
    - ❖ Basic Views , Routing and Controller support are added
    - ❖ Tech Stack: **Golang, SQL languages**

Contributions

- **Prometheus**

  Prometheus(**2 Commits**)
  - ❖ 6762 :- Formatting short tables for readability**(Merged)**
  - ❖ 6761 :- Add conditional rendering of Navlink for Consoles**(Merged)**

  Pushgateway(**22 Commits**)
  - ❖ 327 :- APIs for metadata of metrics**(Merged)**
  - ❖ 328 :- New Revamped UI in React**(Closed)**
  - ❖ 329 :- Change httprouter to common/route**(Closed)**
  - ❖ 332 :- Add test for APIs**(Merged)**
  - ❖ 334 :- API documentation for v1**(Merged)**

- **Prometheus-community**

  Promql Langserver(**4 Commits**)
  - ❖ 122 :- Add Support for new metadata api for hover**(Merged)**
  - ❖ 125 :- Use metadata api for completion**(Merged)**
  - ❖ 138 :- Use documented type for promql type on hover**(Merged)**

- **Submitty (GSoC 2019)**

  Submitty(**35 Commits**)
  - ❖ **Automating Configuration For Assignments Uploads** is an upgrade to old assignment configuration. Previously all inputs and outputs file was to be provided at time of configuration of assignment this made input and output constant for every submission. After this project we can add random inputs and corresponding output at the time of submission.This is possible by providing an instructor solution and python script which can generate random inputs. We can also generate only outputs from predefined inputs at the time of building gradable. This can be used to remove dependencies on predefined output.
  - ❖ The main PRs which sumifies my whole GSoC period are:-
  - ❖ 4335 :- [Feature:Autograding] solution to generate expected output
  - ❖ 4280 :- [BUGFIX:Tests] Validate solution_containers in complete_config.json
  - ❖ 3882 :- [Feature:Autograding] Randomized input and generated output
  - ❖ 3744 :- new test_output and solution in TMP_WORK
  - ❖ Other PRs can be viewed here.

  Submitty.github.io(**2 Commits**)
  - ❖ 162 :- [GSoC] 2019 report from Drumil
  - ❖ 85 :- Incorrect Directory for autograding_workers.json

Project Overview

Rule formatting tool

Prometheus supports two types of rules which can configured and revaluated at regular intervals:-

1. **Alerting Rules**
   Alerting rules allow you to define alert conditions based on Prometheus expression language expressions and to send notifications about firing alerts to an external service.
2. **Recording Rules**
   Recording rules allow you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series.

Rules are written in a yaml file and it mainly consists of a promql expression. Presently, **promtool** supports a syntax checking for rules using following commands :-

```
./promtool check rules /path/to/example.rules.yml
```

Promtool uses rulefmt which eventually uses the syntax tree provided by the parser package in prometheus to check syntax of the expression. We can eventually use this parser to add support for formatting expressions in such a way that they produce uniform style that minimizes deviation and learning curve. Eventually lead into a set of commands to format expressions in rules files

```
./promtool format rules /path/to/example.rules.yml
```

Present problems that should be overcome to add support for promfmt is as follows :-

1. Include Comments
2. Pretty Print expression rather than printing in a single line

Promql Language Server

Promql Language server is an implementation of the language server protocol for promql. Promql language presently supports features :-

1. Show documentation on hover
2. Auto completion
3. Signature information for functions
4. Show error messages for incorrect queries
5. Connect to prometheus to get labels and metric data

Presently, promql language server is in its almost mature condition but it can be improved by providing features like :-

1. Formatting
2. Syntax Highlighting
3. etc.

Deliverables

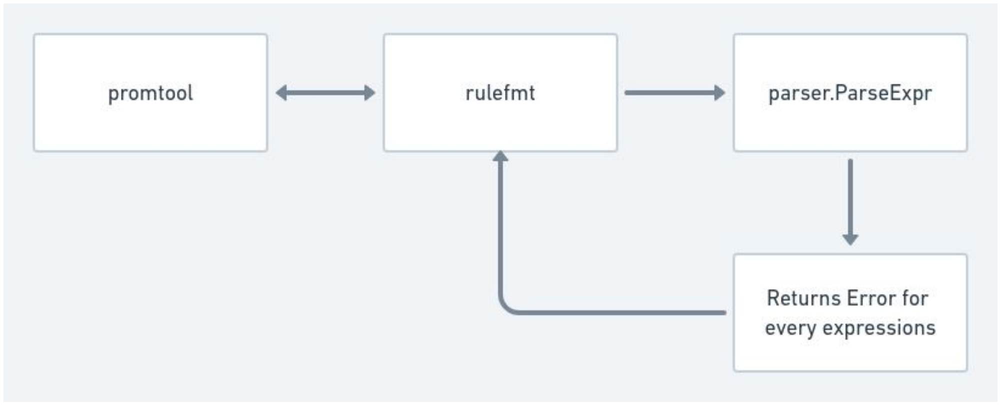
Rule Formatting Tool

rulefmt(Rule Format Checker)

Presently, prometheus have support of syntax checking in promtool for expression via following command

```
./promtool check rules /path/to/example.rules.yml
```

For displaying error promtool uses following work flow :-



promfmt(Promql Formatter)

Promfmt is a system which produces uniform style for rules that minimizes deviation and learning curves. Promfmt aims to have similar features like gofmt but for promql expressions.

Presently, Prometheus has a syntax tree defined in promql/parser which can be used to format expressions for rules. Prometheus also has **`string()`** functions defined in promql/parser/printer.go which can be used to print expressions in a single line.

For supporting formatting we should add following methods :-

1. PrettyPrint
2. Comments should be preserved

PrettyPrint

As discussed earlier, Prometheus presently have **`string()`** method defined in promql/parser/printer.go which can print expression in a single line but to have uniform style that aims to minimizes deviation and learning curves Prometheus should have have some set of rules to format promql expression similar to golang. Using these rules we can add functions which can pretty print our promql expressions that can be overly used in formatting rules.

We can also use these pretty print functions to display expressions in web UI in following pages

1. Rules
2. Graph

Comments should be preserved
Yaml v3 has good support for comments, we can use it to unmarshal yaml files without worrying about comments.

```
// HeadComment holds any comments in the lines preceding the node
// and not separated by an empty line.
HeadComment string

// LineComment holds any comments at the end of the
// line where the node is in.
LineComment string

// FootComment holds any comments following the node and before
// empty lines.
FootComment string
```

You can see example for comments preservation using yaml.v3 here. This way can parse yaml files without worrying about comments and then we can replace expressions with pretty printed expressions. After which we can format yaml.

Problems In Comment Preservation(Discussed with Tobias Guggenmos)

Although,we can solve some common comments problems using the yaml package, but the problem lies for promql comments where it consists of multiline expressions. For eg :-

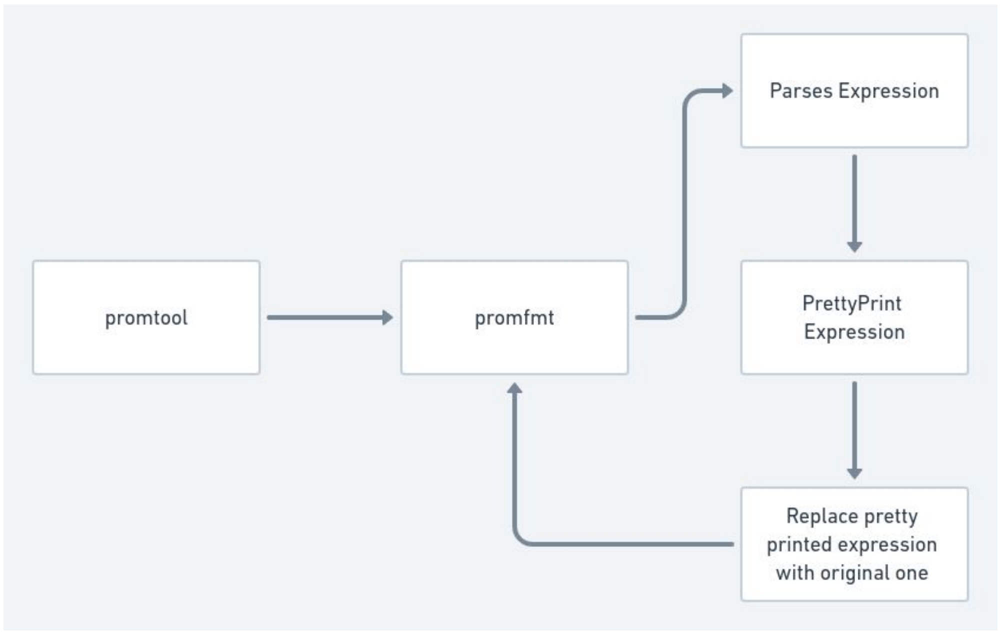
```
expr: |
     rate(errors_total[5m]) # This are comments
     / # If this parses format into one line
     rate(requests_total[5m]) # What we will do
```

Problem lies if after parsing this expression into pretty print function it turns out to be one line expression like :-

```
expr: rate(errors_total[5m]) / rate(requests_total[5m])
```

We can add up multiple lines into one **Line Comment** but the problem is this it destroys the purpose of adding comments. Although it can be solved using different ways, It should be discussed to optimize readability for comments.

Flow Diagram for promfmt :-



Possible Changes in Code

Possible changes in promql/parser are as follows :-

1. **ast.go**

   Add prettyPrint() function in node interface

```
type Node interface {
   // String representation of the node that returns the
given node when parsed
   // as part of a valid query.
   String() string

   // PrettyPrint returns a pretty formatted string for
given node which is used in
   // promfmt to replace it with normal expression
   PrettyPrint() string

   // PositionRange returns the position of the AST Node
in the query string.
   PositionRange() PositionRange
}
```

2. **promfmt.go or prettyPrinter.go**

   Not sure about exact name but this adds implementations **`PrettyPrint()`** for every nodes similar to **`String()`** written in **`printer.go`**

3. **parser.go and lex.go (optional)**

   As discussed with **Tobias Guggenmos,** there are two ways of implementing comments preservation :-

   1. **By directly using yaml v3 package :-** We can directly use yaml v3 package which inbuilts supports for comments so directly we have to add way to replace yaml expressions with pretty printed by parsing yaml node and then a function to print well formatted yaml file (which has prettyPrinted expression)
   2. **By changing Parser code(Selected Approach) :-** Change parser.go and lex.go to support comments by changing parser code. Presently, following code skips comment when parsing

```go
func (p *parser) Lex(lval *yySymType) int {
   var typ ItemType

   if p.injecting {
       p.injecting = false
       return int(p.inject)
   } else {
       // Skip comments.
       for {
           p.lex.NextItem(&lval.item)
           typ = lval.item.Typ
           if typ != COMMENT {
               break
           }
       }
   }

   switch typ {

   case ERROR:
       p.addParseErrf(lval.item.PositionRange(),
"%s", lval.item.Val)
       p.InjectItem(0)
   case EOF:
       lval.item.Typ = EOF
       p.InjectItem(0)
   case RIGHT_BRACE, RIGHT_PAREN, RIGHT_BRACKET,
DURATION:
       p.lastClosing = lval.item.Pos +
Pos(len(lval.item.Val))
   }

   return int(typ)
}
```

Promql Language Server

Promql Language server is an implementation of the language server protocol for promql. Promql language presently supports features :-

6. Show documentation on hover
7. Auto completion
8. Signature information for functions
9. Show error messages for incorrect queries
10. Connect to prometheus to get labels and metric data

Presently, promql language server is in its almost mature condition but it can be improved by providing features like :-

Formatting of Expression

Adding support for formatting of expressions. This can be done using **prettyPrint** function defined in parser and text document formatting protocol in LSP

Nice to do :-

Persist Retroactive Rule Reevaluations

Presently , PR 5887 is working on support for TSDB data import tool which aims to solve the problem of Issue 535(Add mechanism to perform bulk imports). This PR is vital as it adds support for data addition in TSDB.

As per comment by Brian Brazil we can add this feature via promtool. This can be achieved by pre computing rules before creation time and then importing it into TSDB via import tool provided by PR 5887.

Add support of pretty print functions in Web UI

Add support for pretty print functions in following pages :-

1. Rules
2. Graph

Syntax Highlighting for expressions

Add support of server side syntax highlighting language server. This can be achieved by storing a list of lexer tokens with a syntax tree. And a way to combine information provided by lexer with a parsed syntax tree to accurately figure out highlighting. Coordinate yaml language server with promql language server.

Project Timeline

Community Bonding	
4th May - 31st May	- Get to know the Community more, and bond with mentors, admins and developers. - Get feedback if something in the project needs amendment. - Learn more about parser in promql

Phase - 1	
Week 1 **1st June - 7th June**	**Rules for pretty printing expressions** Discuss rules related to pretty print expressions with mentors and community
Week 2 -4 **8th June - 28st June**	**Preserve comments in promql** Update parser.go and lex.go to implement comments inside promql/parser package.
Phase - 2	
Week 5 - 6 **29th June - 5th July**	**PrettyPrint functions in promql/parser/printer.go** Add pretty print functions depending on the rules discussed above.
Week 7 - 8 **6th July - 26th July**	**Support formatting of parsed yaml file** Use pretty printed expressions and formatting rules for yaml to format yaml **Add support for Promfmt in Promtool**
Phase - 3	
Week 9 **27th July - 2nd Aug**	**Add Documentations** Add documentations for promfmt support in promtool.
Week 10 - 11 **3rd Aug - 16th Aug**	**Add Formatting support for promql language server** Add support for formatting in promql language server using pretty print function written in parsers. Presently there is no official support for formatting but it can be extended using extensions
Week 12 **17th Aug - 23th Aug**	**1. Fix bugs and polish new features** **2. Fulfil all remaining backlogs**
Final Evaluation	

Activity Deliverables

- Write blog posts on medium every alternate week.
- Write weekly report of accomplished tasks on the mailing list (or something more relevant)
- Prepare and submit a presentation for this project.

Motivation

Google Summer of Code is an excellent platform to get acquainted with the open source community and their skillful mentors. It gives one a professional work experience in their college years where they collaboratively build a product for the welfare of the society. In this process, both the individual and the community improve and grow.

A couple of months have passed since I have been contributing to the Prometheus organisation, I have had a phenomenal learning experience throughout my journey with prometheus, one specific point that really makes me fond about this organisation is the way they appreciate their contributors for their work.

Resources

- CNCF SOC Page
- Flow Chart for rulefmt
- Flow chart for promfmt
- Prometheus Repo
- Promql language Server
- Whimsical for flowcharts