How to Monitor Gitlab Metrics using Prometheus and Grafana

#DevOps #OptimizeCIPipelines #Third-party Exporter #Customized Exporter

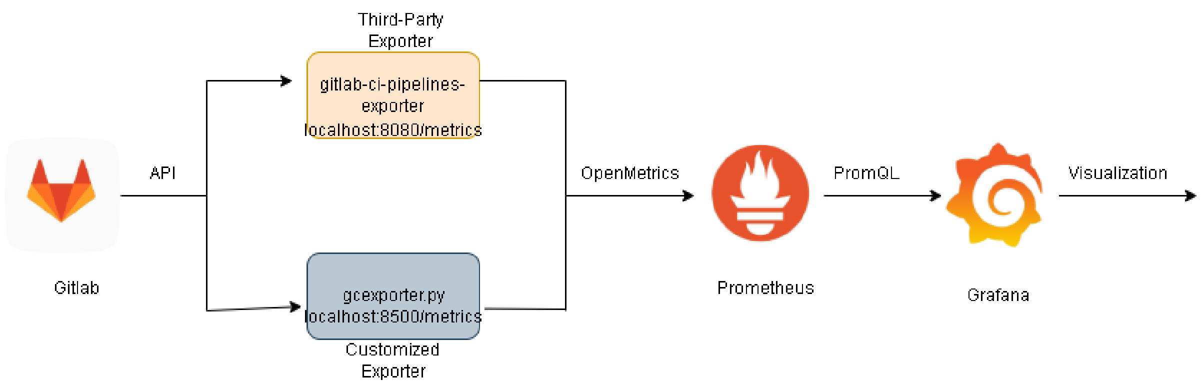
Continuous integration and continuous delivery (CI/CD) pipeline is the backbone of DevOps practice . A well-built pipeline helps to automate software delivery processes and ensure seamless delivery of your application. Hence, monitoring your CI/CD pipelines can help in improving the overall performance of your software delivery process.

There are few ways through which you can export the Gitlab Metrics :

1. Write Customized Exporter to extract data from Gitlab using API and convert it into Open Metrics Format using Prometheus Client library .

2. Use third party exporter "gitlab-ci-pipelines-exporter" to extract Gitlab pipeline events for Prometheus.

3. Configure Gitlab Webhooks to receive event notifications and integrate it with gitlab-ci-pipelines-exporter webhook feature. This is an advanced feature to reduce the amount of requests being made onto your GitLab API endpoint.



In this article, we'll learn to configure third party exporters .Along with that we'll write a Customized Exporter script to export data from Gitlab in the form of OpenMetrics. We'll also learn to configure Prometheus to scrape the metrics and visualize the data using Grafana.

Prerequisites

1. Personal Access Token to access Gitlab Metrics through API. To configure PAT, please refer to https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#create-a-personal-access-token

2. Docker Desktop application should be installed on your system.

Metrics

In this tutorial, we are going to monitor different pipeline events and visualize using Prometheus and Grafana. Along with that we'll write a customized exporter using python to create our own metrics. For this tutorial, we'll fetch the count of existing branches within the project that's being monitored.

For more details on metrics provided by gitlab-ci-pipelines-exporter , please refer to https://github.com/mvisonneau/gitlab-ci-pipelines-exporter/blob/main/docs/metrics.md

Getting Started

In this step we'll create docker containers to set up our Gitlab exporters(third-party and Customized) , Prometheus , and Grafana.

```
version: '3.8'
services:
   grafana:
    image: docker.io/grafana/grafana:latest
    ports:
      - 3010:3000
    environment:
      GF_INSTALL_PLUGINS: grafana-polystat-panel,yesoreyeram-boomtable-panel
    links:
      - prometheus

  prometheus:
      image: docker.io/prom/prometheus:v2.28.1
      ports:
       - 9090:9090
      links:
       - gitlab-ci-pipelines-exporter
      volumes:
```

```
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml

gitlab-ci-pipelines-exporter:
   image: docker.io/mvisonneau/gitlab-ci-pipelines-exporter:v0.5.2
    ports:
     - 8080:8080
   environment:
     GCPE_CONFIG: /etc/gitlab-ci-pipelines-exporter.yml
   volumes:
     - type: bind
       source: ./gitlab-ci-pipelines-exporter.yml
       target: /etc/gitlab-ci-pipelines-exporter.yml

 customized_exporter:
       build: .
       ports:
           - "8500:8500"
```

Third-Party Exporter Setup

Now we'll configure the third-party exporter file "gitlab-ci-exporter.yml" file. In the configuration file, we'll update the project name and gitlab token details as shown below.

```

log:
  level: info

gitlab:
  url: https://gitlab.com
  token: <your access token>

# Example public projects to monitor
projects:
  - name: demo/test_pipeline
    # Pull environments related metrics prefixed with 'stable' for
this project
    pull:
      environments:
        enabled: true
      refs:
        branches:
          # Monitor pipelines related to project branches
          # (optional, default: true)
          enabled: true

          # Filter for branches to include
          # (optional, default: "^main|master$" -- main/master
branches)
          regexp: ".*"
        merge_requests:
          # Monitor pipelines related to project merge requests
          # (optional, default: false)
          enabled: true
```

Customized Exporter Setup

In this section, we'll learn to write a customized exporter to retrieve total branch count in project . We'll use Python Libraries "Python-Gitlab" to fetch Gitlab data and "Prometheus_Client" to convert the data into OpenMetrics.

```
import random
import time
import re
import gitlab
import requests
from requests.auth import HTTPBasicAuth
import json
import time
import random
from prometheus_client import start_http_server, Gauge

group_name=demo
gitlab_server='https://gitlab.com/'
auth_token=<your personal access token>

gl = gitlab.Gitlab(url=gitlab_server, private_token=auth_token)
group = gl.groups.get(group_name)
project_id=<your project id>
project = gl.projects.get(project_id)
```

```
gitlab_branch_count = Gauge('gitlab_branch_count', "Number of Branch
Count")
#Function to Fetch branch count
def get_metrics():
   gitlab_branch_count.set(len(project.branches.list()))

if __name__ == '__main__':
    start_http_server(8500)
    while True:
        get_metrics()
        time.sleep(43200)
```

The customized metrics should be available at localhost:8500/metrics. From the host address, Prometheus will scrape and store the metrics.

```
# HELP gitlab_branch_count Number of Branch Count
# TYPE gitlab_branch_count gauge
gitlab_branch_count 2.0
```

Configure Prometheus and Scrape the Metrics!!!

In this step, we'll configure "prometheus.yml" . The configuration file contains information on the scrape interval i.e the frequency of scraping metrics from the http endpoints along with target addresses that should be monitored by Prometheus.

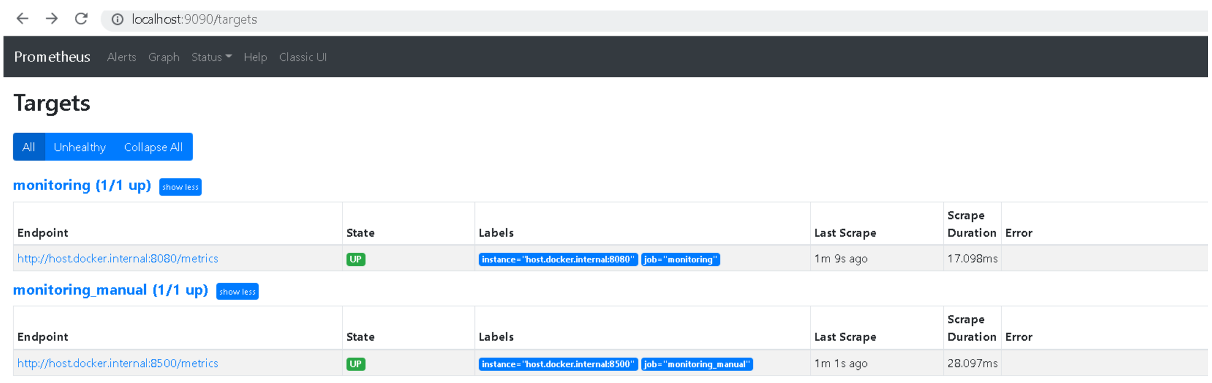
```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: monitoring
  static_configs:
    - targets: ['host.docker.internal:8080']

- job_name: monitoring_manual

  static_configs:
    - targets: ['host.docker.internal:8500']
```

On successful configuration of Prometheus, the target exporters should be visible at the url http://localhost:9090/targets



Targets

monitoring (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://host.docker.internal:8080/metrics	UP	instance="host.docker.internal:8080" job="monitoring"	1m 9s ago	17.098ms	

monitoring_manual (1/1 up)

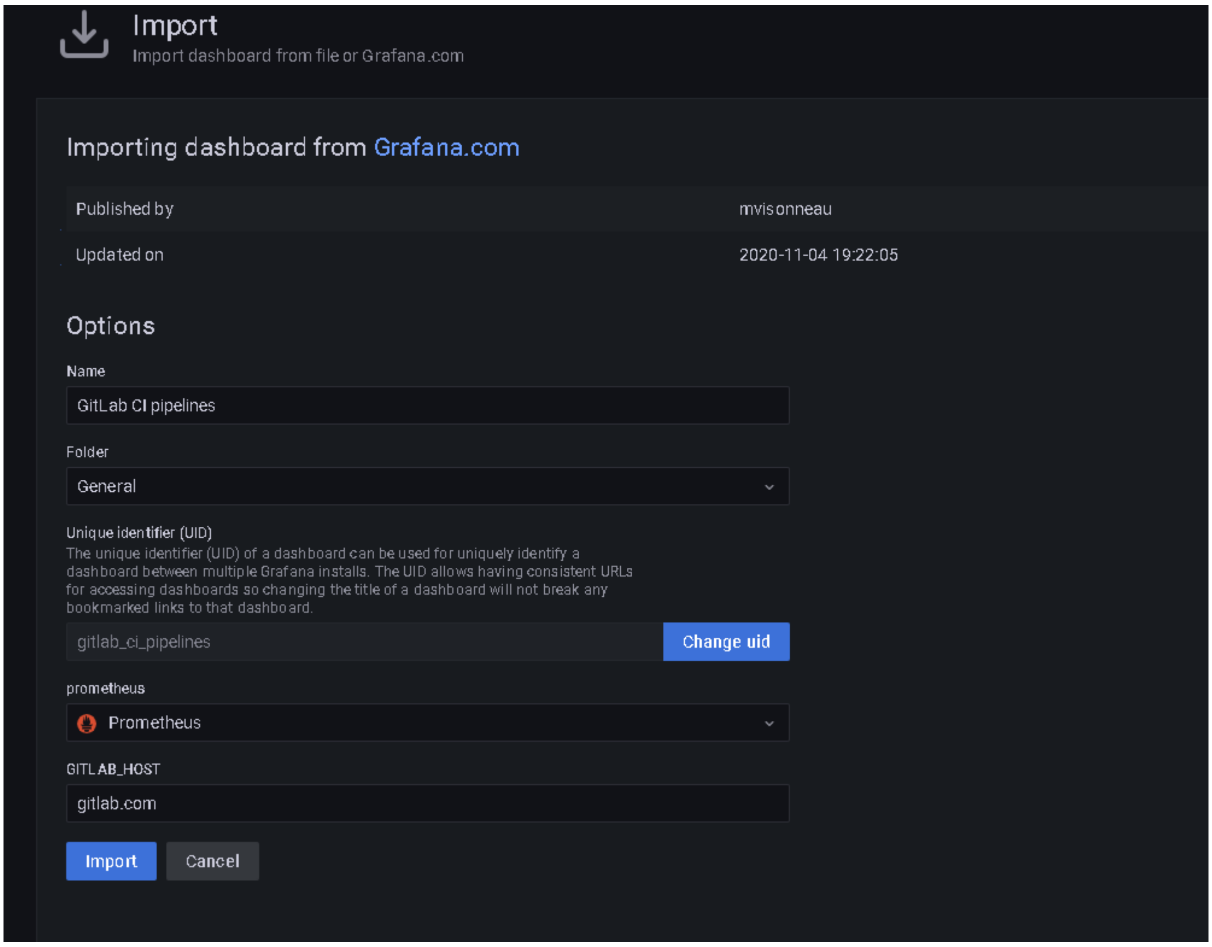
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://host.docker.internal:8500/metrics	UP	instance="host.docker.internal:8500" job="monitoring_manual"	1m 1s ago	28.097ms	

Configure Grafana and Visualize the metrics!!!!!

In this section, we'll install dashboard "Gitlab CI Pipelines" (https://grafana.com/grafana/dashboards/10620) to visualize the third-part exporter metrics as well as create our own panel to display the data fetched through Customized Exporter.

First you need to configure the datasource in Grafana such that it points to Prometheus endpoint

An existing dashboard can be imported into Grafana using the "Import" feature. To add "Gitlab CI Pipelines" , you can either provide the dashboard ID "10620" or fill in JSON configuration of the dashboard.


Import

Import dashboard from file or Grafana.com

Importing dashboard from Grafana.com

Published by	mvisonneau
Updated on	2020-11-04 19:22:05

Options

Name: GitLab CI pipelines

Folder: General

Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

gitlab_ci_pipelines

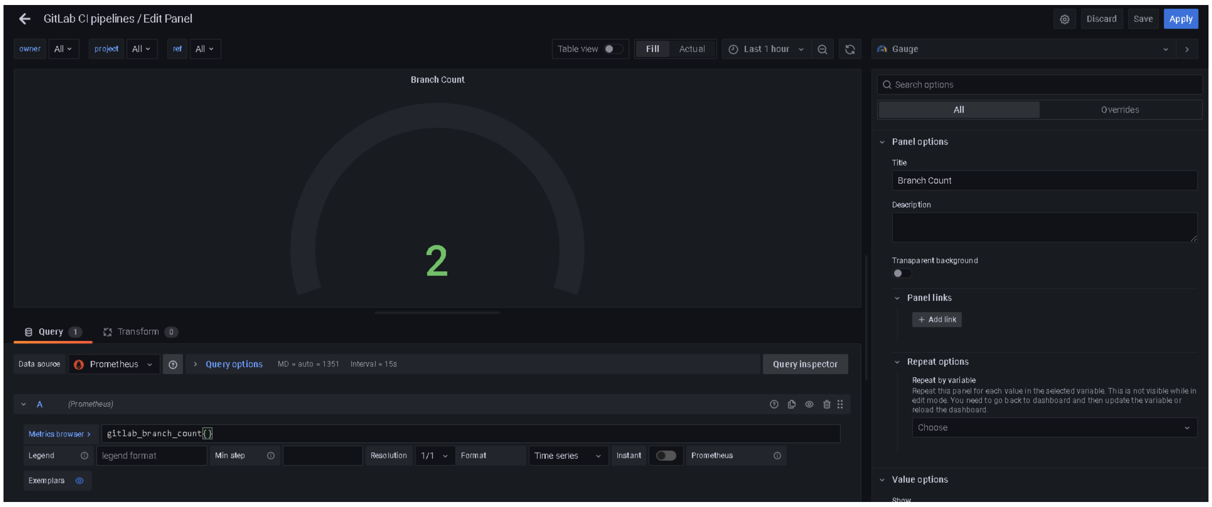
prometheus: Prometheus

GITLAB_HOST: gitlab.com

For more details on importing a dashboard , you can refer to Grafana Official guide
https://grafana.com/docs/grafana/v7.5/dashboards/export-import/

Once your dashboard is configured, you should be able to see the metrics scrapped by the third part“Gitlab-CI-Exporter”
You can add a new panel in the same dashboard to visualize branch count retrieved through Customized Exporter “gcexporter.py”

For this tutorial, I ‘ve used Gauge Visualization to display branch count . Save the updated dashboard.



Title: Branch Count

Repeat by variable: Repeat this panel for each value in the selected variable. This is not visible while in edit mode. You need to go back to dashboard and then update the variable or reload the dashboard.

Query 1 | Transform 0

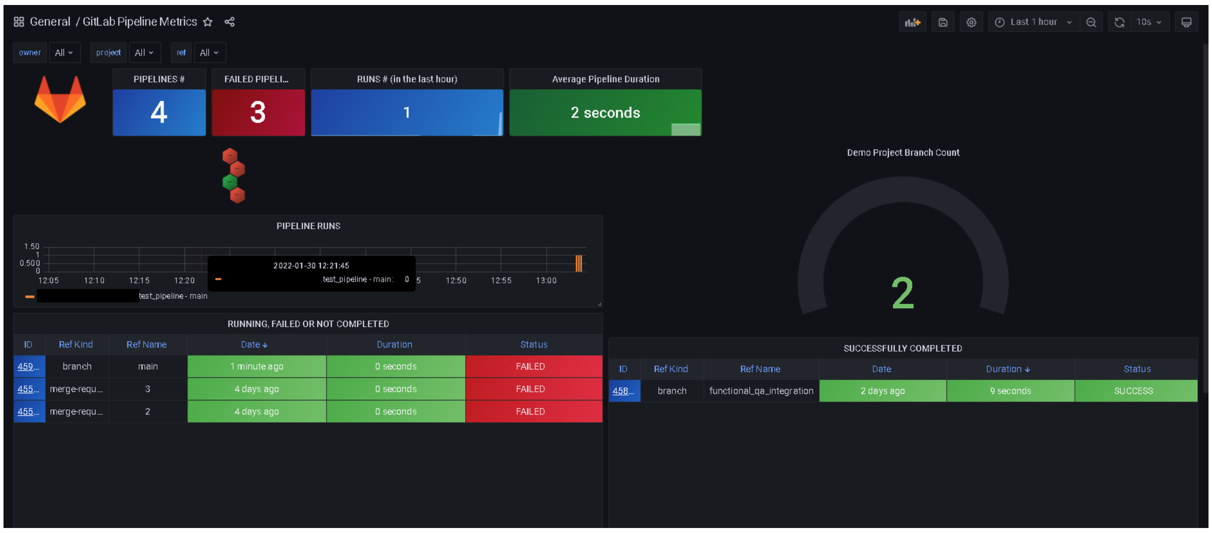
Data source: Prometheus | Query options | MD = auto = 1351 | Interval = 15s | Query inspector

A (Prometheus)

```
gitlab_branch_count{}
```

Legend: legend format | Min step | Resolution: 1/1 | Format: Time series | Instant | Prometheus | Exemplars

You are now ready to monitor your Gitlab Pipelines using Prometheus and Grafana



RUNNING, FAILED OR NOT COMPLETED

ID	Ref Kind	Ref Name	Date ↓	Duration	Status
459...	branch	main	1 minute ago	0 seconds	FAILED
455...	merge-requ...	3	4 days ago	0 seconds	FAILED
455...	merge-requ...	2	4 days ago	0 seconds	FAILED

SUCCESSFULLY COMPLETED

ID	Ref Kind	Ref Name	Date	Duration ↓	Status
458...	branch	functional_qa_integration	2 days ago	9 seconds	SUCCESS

The complete code for this tutorial is available at https://github.com/sharmaranupama/Gitlab_Pipeline_Metrics

Reference:

1.https://github.com/mvisonneau/gitlab-ci-pipelines-exporter
2.https://docs.gitlab.com/ee/ci/pipelines/pipeline_efficiency.html#pipeline-monitoring