International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 3, May -June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Pragmatic Analysis of Custom Jenkins Jobs' Metrics Using Prometheus and Grafana

Jayanth G*

*(Information Science Dept., R.V. College of Engineering, Bengaluru, India

jayanthg.is17@rvce.edu.in)

----------**********----------

Abstract:

CI/CD, in software development, is the process of continuous integration (CI) and continuous delivery/deployment (CD) of code. By imposing automation in the construction, testing, and deployment of applications, CI/CD facilitates development and operations (DevOps) activities and teams. Prometheus is a popular open-source alerting and monitoring software that allows flexible queries via its query language, PromQL. Grafana is one of the widely used open-source tools that can be hooked up with Prometheus' time-series database for metric analytics and visualization. This paper explains a comprehensive approach of extracting, creating, and pushing custom parameters associated with Jenkins jobs as metrics to Prometheus using Python language, filtering them using Prometheus Query Language (PromQL), and visualizing them using Grafana dashboards. These metrics can also serve as a purpose in determining high resource-intensive jobs.

Keywords **—Continuous Integration, Continuous Delivery/Deployment, DevOps, PromQL, Prometheus, Grafana, Jenkins, cron expression, GitHub**

----------**********----------

I. INTRODUCTION

The technique of software development is continually evolving as software businesses introduce faster updates and capabilities to market. To have a competitive edge, development teams ought to optimize their development workflow for more speed, quality, efficiency, and reliability. To do so, devOps teams implement CI/CD in order to automate and accelerate the software delivery lifecycle. The CI/CD workflow continuously integrates and deploys code which benefits the company as a whole [1].

Jenkins is a popular automation tool that facilitates CI/CD, which is being used or adapted in many software companies. A Jenkins job is a runnable task with a specific objective that is controlled by Jenkins. The Jenkins pipeline consists of a set of plugins that help in the CI/CD process, handling integration and deployment of code. For each job that is created, Jenkins stores related metadata within a directory named after the job.

In large companies, the Jenkins resources are more often exploited by high resource-intensive jobs. Depending on the importance of such jobs, they can be flagged for decommissioning.

This paper describes how various resource utilization parameters like average build time, size of GitHub repository, the total number of runs, recent runs of job, inactive and periodically scheduled jobs can be extracted from Jenkins' metadata using Python and exposed as Prometheus metrics. These metrics can then be queried using PromQL and visualized using Grafana.

The visualization can be studied to scrutinize resource-intensive jobs for decommissioning, thereby reducing the load on Jenkins' server.

II. LITERATURE SURVEY

A. B. Brazil, "Prometheus: Up & Running"

This book [2] gives a hands-on introduction to Prometheus' most significant features, such as alerting and dashboarding, metric gathering from third-party systems using exporters, Prometheus Query Language and direct code instrumentation. It also includes instructions for setting up Prometheus, the Node exporter, and the Alert manager, as well as a demonstration of how to utilise them for application and infrastructure monitoring.

B. N. Sabharwal, P. Pandey

This chapter [3] gives readersstep-by-step instructions on how to use PromQL. PromQL (Prometheus Query Language) allows users to query real-time data and conduct various analyses, aggregations, and operations on it.

C. L. Chen, M. Xian, and J. Liu

This paper [4] describes the design and implementation of a comprehensive, intelligent, and efficient monitoring system that uses Prometheus to collect the monitoring data of OpenStack cloud platform and Grafana to display the monitoring data in real time. It also explains how the system can effectively improve the reliability and stability of OpenStack cloud platform through testing.

D. M. Shahin, M. Ali Babar and L. Zhu

This paper [5] aims to conduct a comprehensive evaluation of the state of the art in continuous practises in order to classify methodologies and tools, identify problems and practises, and identify research gaps. It also demonstrates that both greenfield and maintenance projects have benefited from ongoing procedures.

E. I. Nurgaliev, E. Karavakis and A. Alberto

This paper [6] intends to investigate novel visualisation techniques in order to improve dashboards. It explains the use of tools like Kibana, Grafana, and Zeppelin to create new dashboards for the authors' research organisation (CERN) using data stored in ElasticSearch and Hadoop Distributed File System (HDFS).

F. V. Armenise

This paper [7] shows how Jenkins developed from a pure CI platform to a CD platform, embracing the design trend of automating not only the product build but also the release and delivery process. The goal of this paper is to not only present Jenkins as a CD hub, but also to highlight the challenges that still need to be addressed in order to strengthen Jenkins' tracking functionalities.

III. METHODOLOGY

A. Extracting parameters for identifying associated resource-intensive jobs

Before Jenkins stores jobs metadata information within a root directory with the path ~/.jenkins by default [8]. Each job that has a build history is present within the jobs sub-directory, which consists of a directory with its name, containing configuration and other information.

The config.xml file within each job sub-directory can be used to identify cron expressions present, the corresponding GitUrl of the job and the type of job. The cron expression can be parsed according to the syntax to obtain the number of runs of the job in one year. The GitUrl can be used to get the corresponding repository size via REST API provided by GitHub.

The latest subfolder consists of details of the latest successful build. The build.xml file within the folder consists of the start timestamp of the build. This can be used to identify jobs that are inactive for more than a certain number of days.

The builds sub-directory contains information on all the build versions completed so far, for a given job. The build.xml within each build file consists of the duration and start timestamp of the build. The average build time, the total number of runs, and number of recent runs could be hence found easily. An effective load considering average build time and frequency is calculated.

All these parameters thus calculated can then be exposed as Prometheus metrics, which is explained in the following sub-section.

B. Exposing parameters as Prometheus metrics and visualization using Grafana

The following figure explains the process of visualization using Grafana.

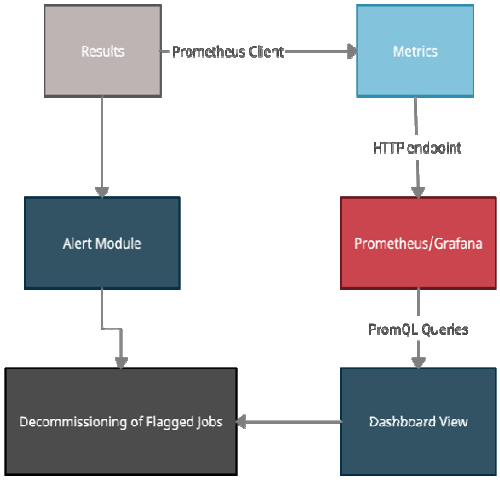

Fig. 1. Visualization of exposed metrics and decommissioning of high resource utilization jobs

The parameters obtained are termed results. There are mainly four types of metrics provided by Prometheus [1]:

- Counter
- Gauge
- Histograms
- Summary

Since all the parameters are numerical, Gauge metric suits best for the same. The Prometheus client provided as a python library is used to create and instantiate Gauge metrics using the various parameters [1]. An HTTP server is set up on a specific port for exposing the metrics, and another HTTP endpoint is used to run Prometheus. Prometheus is configured to listen to the endpoint and pull the exposed metrics.

A graph is set up within Grafana dashboard panel for each metric using PromQL aggregate functions and sorted. The metrics thus visualized are studied and high resource-intensive jobs are flagged for decommissioning.

IV. IMPLEMENTATION AND RESULTS

- The A regex to match all kinds of cron expression within config.xml file is used to find the same if any. The frequency of cronexpressions is calculated based on the syntax shown in Fig. 2 [9].

```
# ┌──────────── minute (0 - 59)
# │ ┌────────── hour (0 - 23)
# │ │ ┌──────── day of the month (1 - 31)
# │ │ │ ┌────── month (1 - 12)
# │ │ │ │ ┌──── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

Fig. 2. Nature of cron syntax]

- Another regex is used to look for GitUrl in the same file which is used as a parameter to get the Git repository size via REST API.
- The start xml tag within build.xml of the latest sub-directory can be searched for to the find timestamp of the latest build. If the time difference between this timestamp and the current timestamp is more than certain time period, such job is termed as inactive.
- The number of directories that are numbered within ‘builds’ sub-directory gives the total number of runs. The start timestamps of builds which fall under the past certain number of days are considered for recent number of runs.
- The duration xml tag within the build.xml file of all the builds can be searched to find build times in milliseconds. These build times of all the builds are averaged out to calculate the average build time of a particular job.
- The cron job frequency and average build time parameters are multiplied to find an approximation of effective resource utilization of cron jobs.
- The python package “Prometheus_client” is used to start the HTTP server for Promethes and set up Gauge metrics.

- Gauge metric is used for each calculated parameter, which is initialized with job name as a label, and parameter as the value.
- A separate HTTP server is set up to run the Python script once GET request is sent.
- Prometheus is configured to listen on the above HTTP endpoint using 'prometheus.yml' config file [10].
- Grafana dashboard is created with graphs for visualization for all metrics. The panel of each graph is configured using PromQL "sum" aggregate function over the label containing the job name. The result is sorted in descending order.

The following views are obtained:

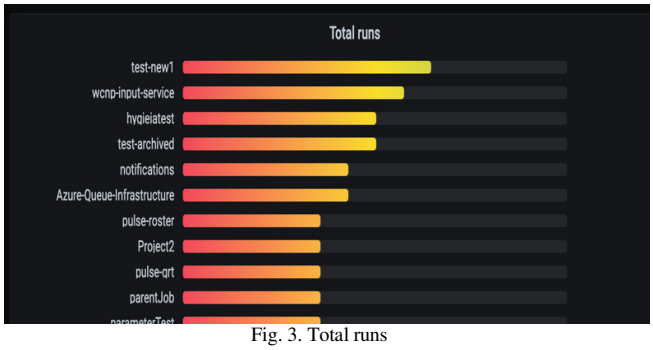

Fig. 3. Total runs

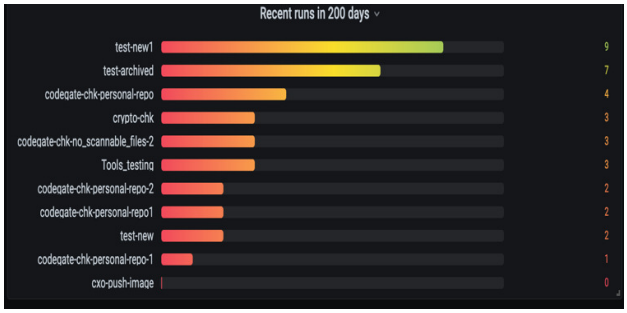

Fig. 4. Recent runs

Fig. 5. Average build time of jobs (seconds)

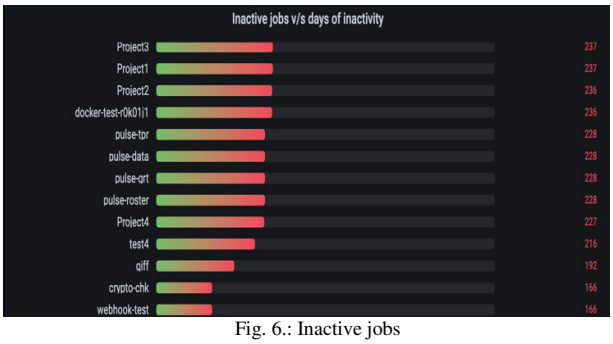

Fig. 6.: Inactive jobs

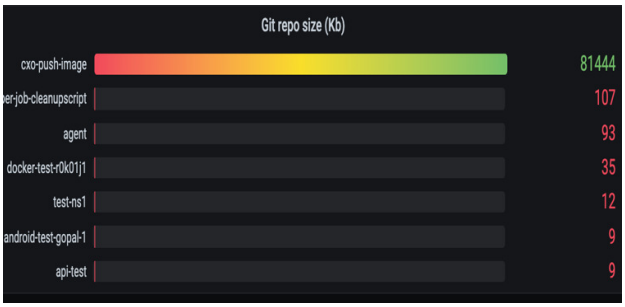

Fig. 7. Git repository size (Kilobytes)

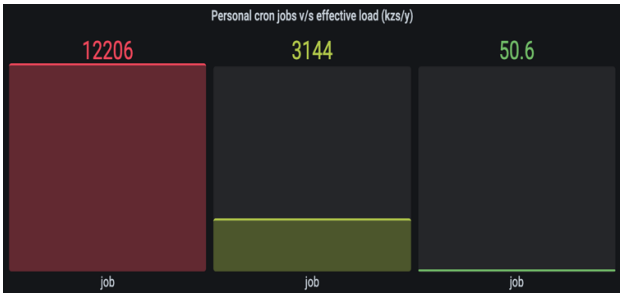

Fig. 8. Cron job effective load

V. CONCLUSIONS

With the near-ubiquitous nature of CI/CD practice in software development, performing regular clean-up tasks is necessary for optimal resource utilization.

Jenkins is the CI/CD tool of choice in many software companies that strive to have a competitive edge in the market. Although Jenkins provides a Prometheus plugin to perform server health monitoring, it is preferable to create custom metrics to analyze various other parameters that can be extracted from the Jenkins root directory. Since a huge number of jobs run every second in a large organization, Jenkins' server resources can easily be exploited if not used efficiently and effectively.

This paper provides a comprehensive walkthrough of the extraction of parameters, conversion of parameters into Prometheus metrics using Python and visualizing them using PromQL queries and Grafana dashboards. The results can also serve as a reference to server maintenance personnel to flag and decommission high resource-intensive and obsolete Jenkins jobs.

REFERENCES

[1] Thoughtworks Incorporation, "Continuous Integration", Accessed on :May 31, 2021. [Online]. Available: https://www.thoughtworks.com/es/continuous-integration

[2] B. Brazil, "Prometheus: Up & Running", O'Reilly Media, Inc, 2018.

[3] N. Sabharwal, P. Pandey. "Working with Prometheus Query Language (PromQL)." In: 'Monitoring Microservices and Containerized Applications.' Apress, Berkeley, CA, 2020. https://doi.org/10.1007/978-1-4842-6216-0_5

[4] L. Chen, M. Xian and J. Liu, "Monitoring System of OpenStack Cloud Platform Based on Prometheus", 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), 2020, pp. 206-209.

[5] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", in IEEE Access, vol. 5, pp. 3909-3943, 2017

[6] I. Nurgaliev, E. Karavakis and A. Alberto. "Kibana, Grafana and Zeppelin on Monitoring data", The European Organization for Nuclear Research (CERN), 2016.

[7] V. Armenise, "Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery," 2015 IEEE/ACM 3rd International Workshop on Release Engineering, 2015, pp. 24-27

[8] Atlassian Confluence, "Administering Jenkins", Aug. 2019. Accessed on: June 1, 2021. [Online]. Available: https://wiki.jenkins.io/display/JENKINS/Administering+Jenkins

[9] R. Boyett, "Cron Job: A Comprehensive Guide for Beginners 2021", May 2021. Accessed on: June 1, 2021. [Online]. Available: https://www.hostinger.com/tutorials/cron-job

[10] Prometheus Authors, The Linux Foundation, "Getting started", Prometheus version 2.27. Accessed on: June 1, 2021. [Online]. Available: https://prometheus.io/docs/prometheus/latest/getting_started.

