Volume: 10 Issue: VII Month of publication: July 2022
DOI: https://doi.org/10.22214/ijraset.2022.45716

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

Building a Monitoring Framework for a Distributed Cloud Application Using Prometheus and Chef - An Overview

Adamyaa DN[1], Dinesh Babu S[2], Priya D[3] , , Soumya A[4]
[1, 2, 3, 4]*Dept. of Information Science and Engineering Rashtreeya Vidyalaya College of Engineering, Bengaluru, India*

Abstract: A Distributed Cloud Application has multiple in-stances dedicated for specific use cases, which requires constant monitoring to avoid hassles in the clients' experience. This in-cludes checking for services, checking for ports and the statuses of various components and component groups present and running in the application. The monitoring framework would alert the support team of any issues, thereby helping them resolve issues immediately. This paper provides a high level generalized overview on building this monitoring framework and automating its installation using Chef.

I. INTRODUCTION

Creating a Monitoring Framework in a distributed cloud application is a significantly complex process and the de-velopers would be better off leveraging any open-source monitoring frameworks available in the market. Prometheus is one such monitoring framework with easily accessible comprehensive documentation. In order to interface a ma-chine with Prometheus, an exporter needs to be designed and executed as a background process. [6] [7] [8]

II. WRITING AN EXPORTER IN PYTHON

As mentioned earlier, an exporter needs to be written and executed, that exposes metrics through a port in the machine. A metric captures a value pertaining to you system at a specific point in time. Prometheus has some default metric exporters in various languages like Python and Go. Prometheus also offers extensive support and client libraries for developers to write their own exporters in multiple languages. This section elaborates on the exporter build-ing process in one of the most intuitive and widely-used languages in the monitoring sphere - Python. [2]

A. Testing Netstat Commands

One of the most common use cases in any monitoring setup is to look for the status of various ports in the machine. For example, there could be a metric to determine whether port '9001' is listening. This could be of importance to the support team. How does one tackle this? This is where 'netstat' commands come in.

Netstat is a command-line network utility that provided information about connections and other network statistics. This command can be frequently used in conjunction with 'grep'. Grep is acommand-line utility that checks for regular expressions in a data-set with text lines.

For example, look at the following command :

```
$ netstat -anp | grep 9001
```

Here, the netstat command lists all the ports, and the grep checks the data for the regular expression which in this case matches the port number, and prints it out. Netstat can be used in conjunction with grep for various other utilities that involves ports and networking.

[20] The final output of this 'netstat' command can be taken as an input stream in the exporter, and then tested using string handling commands. [10]

B. Checking For Services

The concept of checking for services is more complex than netstat usage and would require personalized com-mands. For example, the 'systemctl' command can be used to introspect and control the state of 'systemd' system and service manager. Here's an example:

```
$ systemctl status servicename.service
```

This command provides and output where the status of the service is given. This output can then be passed tothe python exporter as an input, where the string handling functions can look for a particular substring to determine whether the service is running. [12]

Another useful command is ’ps’. The ’ps’ command is used to list all processes running at any point in time. Like netstat, ’ps’ can be used in conjunction with ’grep’ to match regular expressions and limit the output size. Here’s an example:

```
$ ps -fA | grep python3
```

Here, we list all processes that are currently using python3 for execution.

C. Python Commands And Tools

operators. It is used as a part of Linux commands along with other commands mentioned earlier in this paper. For example:

```
$ free -t | awk 'NR == 2 {printf("%.2f"), $3/$2*100}'
```

In this command mentioned above, the free command communicates the amount of free memory left in the ma-chine. The awk programming part of the command takes the second row of the output and performs some calculations on it. [17]

This paper focuses on writing an exporter using the Python language. This section will focus on the different useful things to know and keep in mind while writing an exporter in the python language.

1) *Prometheus Client Library:* The exporter needs the Prometheus client libraries as a prerequisite to execute. There is extensive open source documentation on the same. Some important types of metrics are Gauge, Histogram, Counter, etc. Each type of metric has a particular use-case. The most commonly used metric type in a monitoring use-case is the Gauge. This is because the Gauge type metricenables values to go both ways on the rational number line. Besides this, there is documentation on the program structure when it comes to writing an exporter. Metrics will have to be pulled constantly, and this would require the exporter to be run as a background process, and the metrics to be scraped repeatedly at fixed intervals. [3]

2) *Python Language Functions:* The ’json’ package is very useful in writing an exporter, as the metrics and other data can be fed to the exporter in the form of a JSON file which is imported as a python dictionary data structure. It can be used as follows:

```
metric_file=open('filename.json') data=json.load(metric_file)
```

The ’os’ package has also repeatedly proven useful in writing an exporter. As mentioned earlier, the commands have to be interfaced with the python language exporter. The implementation for that is as follows:

```
stream = os.popen('ps -fA | grep python3') output = stream.read()
```

Here, the ’popen()’ function is fed with the command that needs to be executed, and the output is stored as a string for further processing.

D. Awk Programming

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling and allows the user to use variables, numeric functions, string functions, and logical

III. VISUALIZATION ON PROMETHEUS

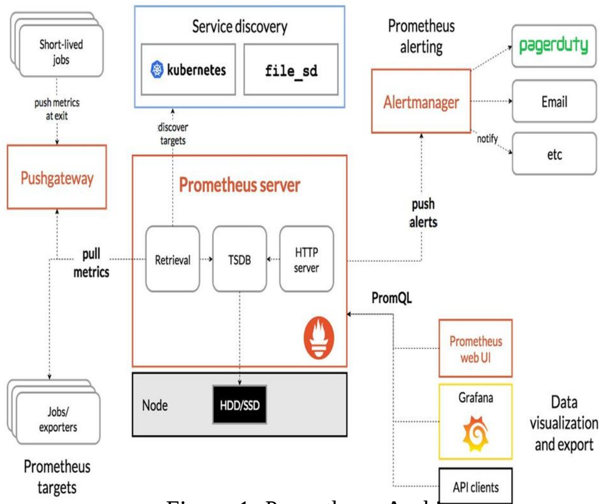

Figure 1. Prometheus Architecture

As shown in the figure, the Prometheus server pulls metrics from the Prometheus targets. The target URLs are fed to the Prometheus server through the configuration file. These metrics are then visualized and analysed for patterns. [9]

A. Finding Patterns in Resource Usage

Usage of resources like CPU, Memory and Heap Mem-ory can be visualised and analysed to determine trends in usage and take corrective action for time frames where resource usage is high.

B. Finding Patterns in Metric Variations

Similarly, there are trends in the different metrics which determine whether ports are listening or if services are running. For example, when certain services are run, they may result in adverse affects on certain other services. These trends need to be found and rectified.

IV. ALERTING USING PROMETHEUS

Alerting is done in Prometheus by using two separate kinds of '.config' type files - Prometheus and Alertmanager. Prometheus worked with the Target URLs, port numbers and domiain names, whereas Alertmanager worked with inhibi-tion rules, alerting intervals, etc. Alertmanager is linked to the Prometheus Server as shown in the figure, where alerts are pushed if generated. Alertmanager is then interfaced with Slack, Outlook and other communication channels. [16]

Formulating Alerting Rules

Alerting rules have to be formulated for the different metrics and for the machine itself. If the machine itself is not exporting metrics, an alert needs to be generated. In terms of metrics, if a resource metric goes beyond a certain threshold, an alert needs to be generated.

An expression is written as a conditional statement that needs to be fulfilled for an alert to be generated. An alert text and summary is written with some labels related to the metric and the machine. When an alert is generated, the summary contains the details of the issue that will help an support team rectify any issues. [18]

V. AUTOMATION USING CHEF

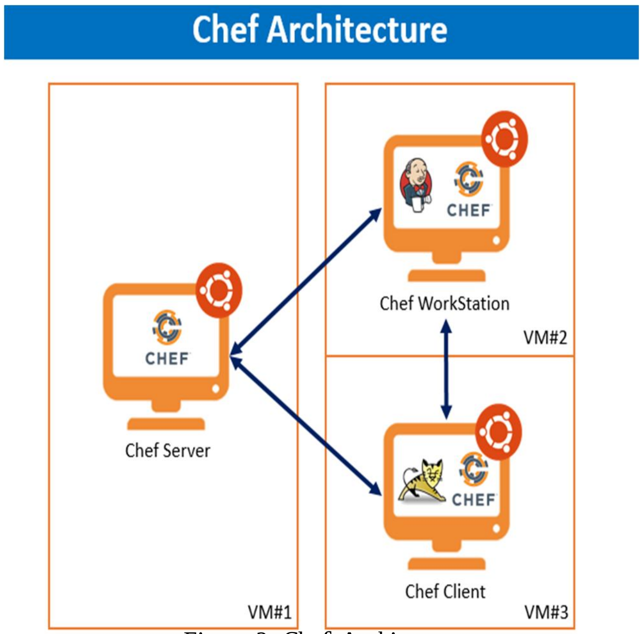

Figure 2. Chef Architecture

Chef is an automation software generally used for In-frastructure Provisioning and managing a variety of node types like servers, virtual machines, etc. In this use-case, the monitoring framework's installation on a distributed cloud has to be automated. As in, the monitoring framework has to be installed in multiple nodes at once, where each node is not of the same type. Chef uses a three-tier client server model where all working units are developed in the Chef Work Station and uploaded to the Chef Server using com-mand line utilities like knife. Once uploaded, these scripts are then used on the the different Chef Clients or Target Nodes to achieve the desired output. [13] [4] [1]

A. Roles
A role is a means of designating specific patterns and procedures that are present across organisational nodes abelonging to a particular job function. Each role has a run-list and zero or more attributes. There may be 0 or more roles assigned to a node. When a role is applied to a node, the node's configuration information is compared to the attributes of the role, after which the run-list's contents are applied to the node's configuration information. [15] [11]

1) Classification of Roles. Roles for web servers, database servers, and other servers are common examples. All nodes can have a custom run list configured, and roles can override attribute values.

2) Usage of a Runlist. All the details required for Chef to set up a node in the appropriate state are defined in a run-list. A run-list is a list of roles or recipes that are executed in the precise order specified in the run-list; if a recipe occurs more than once in the run-list, Chef won't execute it more than once.

3) Utilizing Global attributes. Global attributes can be specified in the role files, which can then be used in the recipes. This would categorically reduce the number of times the recipe needs to be changed, because any hard coding of values will be done in a role file and the recipe just needs to invoke the variable, thereby making cookbooks more resilient and independent.

B. Recipes
The most basic configuration component in an organi-sation is a recipe. A recipe is written in the computer lan-guage Ruby, which is made to read and behave predictably. It mostly consists of a collection of resources that have been specified using patterns (resource names, attribute-value pairs, and actions); when necessary, auxiliary code is added using Ruby around this.

1) Resources. Chef resources are an example of an operating system component in their ideal state. It is a declaration of configuration policy that explains the desired state of a node to be reached by resource providers using the existing configuration. Using Chef's OHAI mechanism, it helps to know the target machine's current condition. Additionally, it aids in outlining the actions necessary to bring the target machine into that state. The resources are arranged as recipes that explain the working setup. [5] [14]

2) Some Important Resources. Some important re-sources used are:

a) directory: Creates and modifies the properties and permissions of a directory

b) cookbook_file: Transfers and places a file from the Chef Workstation to a Target Node

c) execute: Executes a Linux command in the Target Node

d) file: Creates a file with the content specified in the resource block

e) bash: Execute Linux commands in a bash environ-ment [19]

C. Files
Files used in a cookbook with the cookbook file resource are kept in the files directory in the Chef cookbooks. Any and all dependencies to be installed are placed in the files folder in the Chef Workstation and then transferred using the cookbook_file resource before being installed with the execute resource.

VI. CONCLUSION
This paper gives an overview of how to tackle the setup of a monitoring framework for a distributed cloud application and automate the end-to-end installation of the framework. This paper is in no way a complete solution and is designed to give a structural overview and guidance for an implementation.

Besides giving a structural overview, this paper also provides some useful tips and tricks that would make the process of building a monitoring framework significantly easier than it was for the authors of this paper.

Areas of improvement and further research would include the building of a Python exporter that is more generalised and can scrape a wide array of metrics. This would mitigate the exporter writing process in multiple different use-cases.

VII. ACKNOWLEDGMENTS
The authors would like to thank the guides and mentors at Rashtreeya Vidyalaya College of Engineering for the constant support and guidance.

REFERENCES
[1] James O. Benson, John J. Prevost, and Paul Rad. Survey of automated software deployment for computational and engineering research. 2016 Annual IEEE Systems Conference (SysCon), 2016.
[2] Brian Brazil. Prometheus: Up amp; running: Infrastructure and application performance monitoring. O'Reilly Media, Inc., 2018.

[3] Mainak Chakraborty and Ajit Pratap Kundan. Prometheus. Monitor-ing Cloud-Native Applications, page 99–131, 2021.
[4] Chef. Chef documentation.
[5] Chef. Ruby for chef.
[6] Shivakumar R Goniwada. Cloud native architecture and design patterns. Cloud Native Architecture and Design, page 127–187, 2021.
[7] Shivakumar R Goniwada. Enterprise cloud native automation. Cloud Native Architecture and Design, page 523–553, 2021.
[8] Shivakumar R Goniwada. Infrastructure automation. Cloud Native Architecture and Design, page 619–634, 2021.
[9] Grafana. Prometheus.
[10] Kenneth Hitchcock. Monitoring. Linux System Administration for the 2020s, page 203–240, 2022.
[11] Waldemar Hummer, Florian Rosenberg, Fabio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In David Eyers and Karsten Schwan, editors, Middleware 2013, pages 368– 388, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
[12] Philip Kirkbride. Systemd. Basic Linux Terminal Tips and Tricks, page 221–234, 2020.
[13] R.O. Kostromin. Survey of software configuration management tools of nodes in heterogeneous distributed computing environment. The International Workshop on Information, Computation, and Control Systems for Distributed Environments, 2020.
[14] Matthias Marschall. Chef Infrastructure Automation Cookbook -Second Edition. Packt Publishing, 2015.
[15] Stuart Preston. Using chef with Microsoft Azure. Apress, 2016.
[16] Prometheus. Alertmanager: Prometheus.
[17] Arnold Robbins. Effective awk programming. O'Reilly, 2015.
[18] Navin Sabharwal and Piyush Pandey. Getting started with prometheus and alert manager Monitoring Microservices and Containerized Applications, page 43–83, 2020.
[19] Navin Sabharwal and Manak Wadhwa Lightweight resource providers. Automation through Chef Opscode, page 169–178, 2014.
[20] G. Vigna and R.A. Kemmerer. Netstat: a network-based intrusion detection approach. In Proceedings 14th Annual Computer Security Applications Conference (Cat. No.98EX217), pages 25–34, 1998.