



****

**题 目:**  Introduction to Prometheus

**学 号：** 2020141461137

**年 级：** 2020级

**姓 名：** 熊永美

**任课教师：** Jeff Tao

二 Ο 二 二 年 7 月 7 日

Introduction to Prometheus

**Abstract:** Prometheus is an open source monitoring solution for collecting and aggregating metrics as time series data. This article mainly introduces the role, features, components, workflow, deployment mode, indicators and its data query language PromQL of Prometheus. Learn more about Prometheus by consulting various sources.

**Keywords:** Prometheus, components, workflow, deployment mode, metrics, PromQL

**1.Introduction to Prometheus**

Prometheus is an open source system monitoring and alarm system. It has now joined the CNCF Foundation and has become the second project hosted by CNCF after k8s. In the kubernetes container management system, it is usually monitored with Prometheus, and it also supports multiple A variety of exporters collect data, and also support pushgateway for data reporting, Prometheus performance is enough to support tens of thousands of clusters.

**2. Features of Prometheus**

**2.1. Features of Prometheus**

1) Multidimensional data model

Each time series data is uniquely identified by a set of metric metric name and its label labels key-value pair: the metric metric name specifies the measurement characteristics of the monitoring target system (eg: http\_requests\_total - the total count of http requests received). Labels enables Prometheus's multidimensional data model: for the same measure name, through the combination of different label lists, a specific measure dimension instance will be formed. (For example: all http requests containing the metric name /api/tracks, marked with method=POST, form a specific http request). This query language filters and aggregates based on these lists of measures and labels. Change any label value on any measure and a new time series graph will be formed.

2) Flexible query language (PromQL): operations such as addition, multiplication, and connection can be performed on the collected metrics;

3) It can be deployed directly locally without relying on other distributed storage;

4) Collect time series data through HTTP-based pull method;

5) The time series data can be pushed to the Prometheus server through the intermediate gateway pushgateway;

6) The target service objects (targets) can be discovered through service discovery or static configuration.

7) There are a variety of visual image interfaces, such as Grafana and so on.

8) Efficient storage, each sampled data occupies about 3.5 bytes, 3 million time series, 30s interval, 60 days retention, and consumes about 200G of disk.

9) To be highly available, you can do remote backup of data, federate clusters, deploy multiple sets of Prometheus, and pushgateway to report data

**2.2. What is a sample**

Sample: Each point in the time series is called a sample, and the sample consists of the following three parts:

Metric: metric name and labelsets that describe the characteristics of the current sample;

Timestamp: a timestamp accurate to milliseconds;

Sample value (value): A folat64 floating point data representing the value of the current sample.

Representation: Represent the time series of the specified metric name and the specified label set by the following expression: <metric name>{<label name>=<label value>, ...}

For example, a time series with the metric name api\_http\_requests\_total and the tags method="POST" and handler="/messages" can be represented as: api\_http\_requests\_total{method="POST", handler="/messages"}.

**3. Introduction to Prometheus components**

1) Prometheus Server: Used to collect and store time series data.

2) Client Library: The client library detects the application code. When Prometheus crawls the HTTP endpoint of the instance, the client library will send the current status of all tracked metrics to the Prometheus server.

3) Exporters: Prometheus supports a variety of exporters. Through the exporter, metrics data can be collected and sent to the Prometheus server. All programs that provide monitoring data to the Promtheus server can be called exporters.

4) Alertmanager: After receiving alerts from the Prometheus server, it will deduplicate, group, route to the corresponding receiver, and issue an alarm. Common receiving methods are: email, WeChat, DingTalk, slack, etc.

5) Grafana: monitoring dashboard, visual monitoring data

6) pushgateway: Each target host can report data to pushgateway, and then Prometheus server pulls data from pushgateway uniformly.

The entire ecosystem of Prometheus mainly includes Prometheus server, Exporter, pushgateway, alertmanager, grafana, and Web ui interface. Prometheus server consists of three parts, Retrieval, Storage, and PromQL. Retrieval is responsible for grabbing monitoring indicator data on the active target host. Storage storage mainly stores the collected data to the disk. PromQL is a query language module provided by Prometheus.

**4. Prometheus workflow**

1) The Prometheus server can periodically pull monitoring indicator data from the active (up) target host (target). The monitoring data of the target host can be collected by the Prometheus server by configuring static jobs or service discovery. This method defaults to The indicators are pulled in the pull method; the collected data can also be reported to the Prometheus server through the pushgateway; the data of the corresponding components can also be collected through the exporter that comes with some components;

2) The Prometheus server saves the collected monitoring indicator data to the local disk or database;

3) The monitoring indicator data collected by Prometheus is stored in time series, and the triggered alarms are sent to the alertmanager by configuring the alarm rules

4) Alertmanager can send the alarm to email, WeChat or DingTalk by configuring the alarm receiver

5) The web ui interface that comes with Prometheus provides PromQL query language, which can query monitoring data

6) Grafana can access the Prometheus data source and display the monitoring data in a graphical form

**5.Prometheus and zabbix comparative analysis**

1) The back end of Prometheus is developed with golang, and the front end is Granfana, JSON, which can be solved by editing, and the difficulty of customization is low. The zabbix backend is developed in C, and the interface is developed in PHP, which is very difficult to customize.

2) Prometheus supports larger cluster size and faster speed. The upper limit of zabbix cluster size is 10,000 nodes.

3) Prometheus is more suitable for monitoring the cloud environment and has better integration with OpenStack and Kubenetes. Zabbix is more suitable for monitoring physical machine environments.

4) Prometheus monitoring data is stored in a time series-based database, which facilitates new aggregation of existing data. Zabbix monitoring data is stored in a relational database, such as MySQL, which is difficult to expand dimensions from existing data.

5) The installation of Prometheus is relatively complicated, and monitoring, alarms and interfaces belong to different components. Zabbix is easy to install, and zabbix-server includes all server-side functions in one package.

6) The Prometheus interface is relatively weak, and many configurations need to modify the configuration files. The zabbix graphical interface is relatively mature, and basically all configuration operations can be completed on the interface.

1. **Several deployment modes of Prometheus**

**6.1. Basic high availability mode**

The basic HA mode can only ensure the availability of the Prometheus service, but it does not solve the problem of data consistency and persistence between Prometheus Servers (which cannot be recovered after data loss), nor can it be dynamically expanded. Therefore, this deployment method is suitable for scenarios where the monitoring scale is not large, Prometheus Server does not migrate frequently, and only short-period monitoring data needs to be saved.

**6.2. Basic high availability + remote storage**

On the basis of solving the availability of Prometheus service, and ensuring the persistence of data, when the Prometheus Server is down or data is lost, it can be quickly recovered. At the same time Prometheus Server may be well migrated. Therefore, this solution is suitable for scenarios where the user monitoring scale is not large, but it is hoped that the monitoring data can be persisted and the portability of the Prometheus Server can be ensured.

**6.3. Basic HA + remote storage + federated cluster solution**

The performance bottleneck of Prometheus mainly lies in a large number of collection tasks. Therefore, users need to use the characteristics of the Prometheus federated cluster to divide different types of collection tasks into different Prometheus sub-services to achieve functional partitioning. For example, a Prometheus Server is responsible for collecting infrastructure-related monitoring indicators, and another Prometheus Server is responsible for collecting application monitoring indicators. Then there is the upper-layer Prometheus Server to realize the aggregation of data.

**7. Prometheus’s Metrics**

The Prometheus client libraries offer four core metric types. These are currently only differentiated in the client libraries (to enable APIs tailored to the usage of the specific types) and in the wire protocol.

**7.1. Counter**

Counter is a counter type:Counter is used to accumulate values, such as recording the number of requests, the number of task completions, and the number of errors.Always increasing, never decreasing.After restarting the process, it will be reset.

Counter type data allows users to easily understand the changes in the rate of event generation. The related operation functions built in PromQL can provide corresponding analysis, such as the amount of HTTP application requests to illustrate:

1) Obtain the growth rate of HTTP requests through the rate() function: rate(http\_requests\_total[5m])

2) Query the top 10 HTTP addresses in the current system: topk(10, http\_requests\_total)

**7.2. Gauge**

Gauge is the gauge type:Gauge is a general value, such as temperature change, memory usage change.Can be big, can be small.After restarting the process, it will be reset.

For Gauge type monitoring indicators, the PromQL built-in function delta() can be used to obtain changes in samples over a period of time, for example, to calculate the difference in CPU temperature within two hours:

dalta(cpu\_temp\_celsius{host="zeus"}[2h])

You can also use the PromQL built-in function predict\_linear() to predict the trend of sample data based on simple linear regression. For example, to predict the remaining disk space on the host after 4 hours based on 2 hours of sample data: predict\_linear(node\_filesystem\_free{job="node"}[2h], 4 \* 3600) < 0.

**7.3. Histogram**

Histogram is a histogram, which has three functions in the query language of the Prometheus system:

1) Sampling data (usually request duration or response size, etc.) over a period of time, and count it into a configurable bucket. The samples can be filtered by the specified interval later, or the samples can be counted The total, and finally the data is generally displayed as a histogram.

2) Cumulative sum (sum) for each sampling point value

3) Cumulative sum of the number of sampling points (count)

**7.4 Histogram**

Similar to the Histogram type, the summary is used to represent the results of data sampling over a period of time (usually request duration or response size, etc.), but it directly stores quantiles (calculated by the client, and then displayed), not Calculated by interval (the quantile of the Histogram needs to be calculated by the histogram\_quantile (φfloat, b instant-vector) function). Therefore, for the calculation of quantiles, Summary has better performance when querying through PromQL, while Histogram consumes more resources. Conversely, for the client, the Histogram consumes less resources. When choosing these two methods, users should choose according to their actual scenarios.

**8.Prometheus: PromQL**

PromQL is the built-in data query language of Prometheus, which provides support for rich query, aggregation and logical computing capabilities of time series data. And it is widely used in the daily applications of Prometheus, including data query, visualization, and alarm processing. It can be said that PromQL is the foundation of all application scenarios of Prometheus, and understanding and mastering PromQL is the first lesson for getting started with Prometheus.

**8.1. Query time series**

After Prometheus collects the corresponding monitoring indicator sample data through Exporter, we can query the monitoring sample data through PromQL. When we query directly using the monitoring indicator name, we can query all time series under this indicator.

PromQL also supports users to filter time series according to the label matching mode of the time series. Currently, it mainly supports two matching modes: exact matching and regular matching. In addition to filtering time series by exact matching, PromQL can also support the use of regular expressions as matching conditions, and use | to separate multiple expressions:

Use label=~regx to select time series whose labels conform to the regular expression definition; otherwise, use label !~regx to exclude.

**8.2, range query**

When querying a time series directly through a PromQL expression like http\_requests\_total, the return value will only contain the latest sample value in the time series, and we call this return result an instantaneous vector. The corresponding such expressions are called \_\_ instantaneous vector expressions.

If we want sample data over a period of time in the past, we need to use an interval vector expression. The difference between the interval vector expression and the instantaneous vector expression is that in the interval vector expression we need to define the range of time selection, and the time range is defined by the time range selector [].

**9. Summary**

Prometheus is a time-series data store that can be used to manage any time-based sequential data. It is most commonly used to monitor metrics from other applications in the stack. While Prometheus is an efficient system for storing and querying metrics, it is often integrated with other solutions to support graphical dashboards and advanced visualizations. Its popularity rests on its ability to handle custom metrics, support rich queries, and interoperate with other members of the cloud-native ecosystem.

**References**

<https://prometheus.io/>

https://baijiahao.baidu.com/