devfest
2022

Prometheus监控及GCP代管式服务

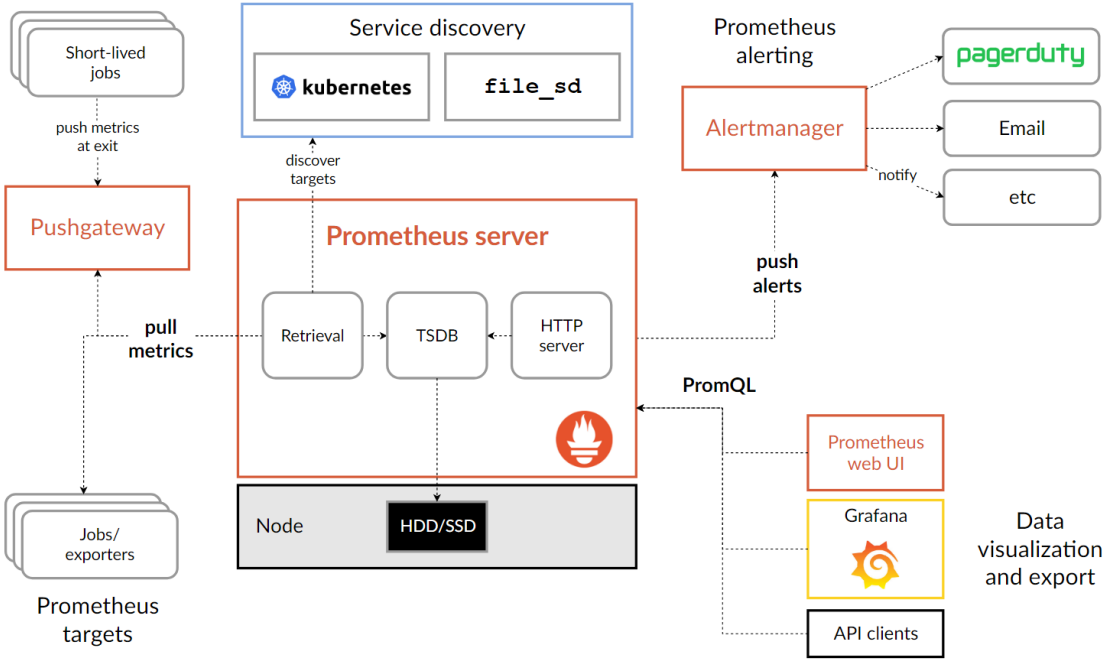
Google Developer Groups
[Location]

devfest 2022

Prometheus 是什么

张宇 厦门过时智能有限公司

首先看一下官方的架构图



优点：

1. Prometheus 是按照 Google SRE 运维之道的理念构建的，具有实用性和前瞻性。

2. Prometheus 社区非常活跃，基本稳定在 1个月1个版本的迭代速度，从 2016 年 v1.01 开始接触使用以来，到目前发布的 v2.37.4LTS 以及最新最新的 v2.40 ，Prometheus 一直在进步、在优化。

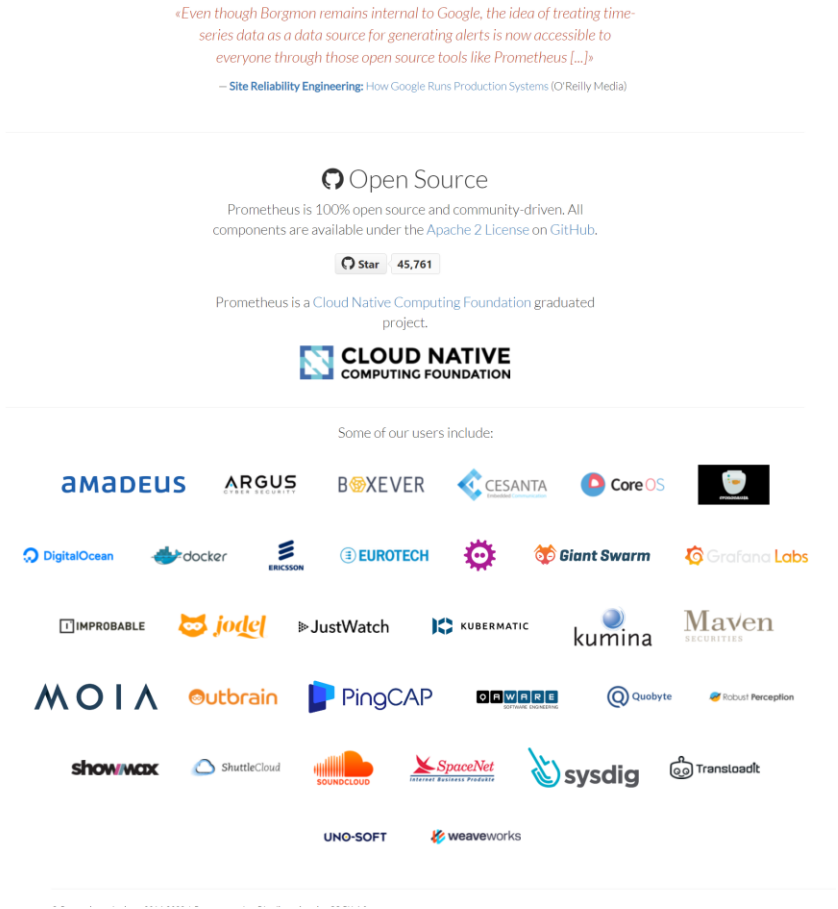
3. Go 语言开发，性能不错，安装部署简单，多平台部署兼容性好。

4. 丰富的数据收集客户端，官方提供了各种常用 exporter。

5. 丰富强大的查询能力。

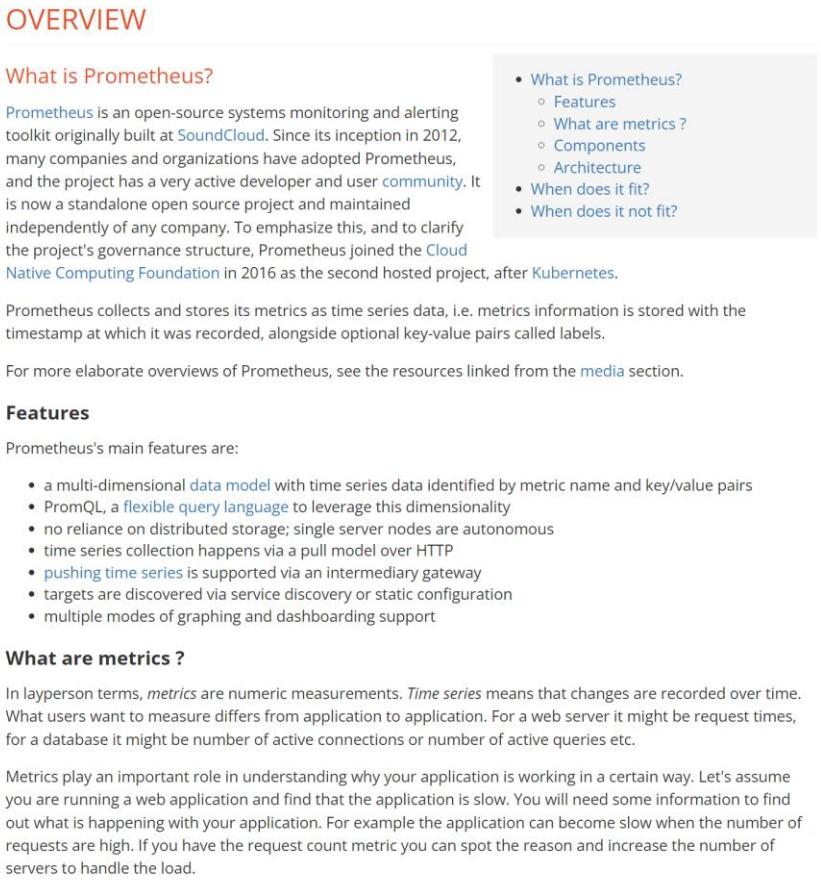
Prometheus 是什么

Prometheus 是由 SoundCloud 开源监控告警解决方案，从 2012 年开始编写代码，再到 2015 年 github 上开源以来，已经吸引了 9k+ 关注，以及很多大公司的使用；2016 年 Prometheus 成为继 k8s 后，第二名 CNCF(Cloud Native Computing Foundation) 成员。



主要功能

多维 数据模型（时序由 metric 名字和 k/v 的 labels 构成）。
灵活的查询语句（PromQL）。
无依赖存储，支持 local 和 remote 不同模型。
采用 http 协议，使用 pull 模式，拉取数据，简单易懂。
监控目标，可以采用服务发现或静态配置的方式。
支持多种统计数据模型，图形化友好。



OVERVIEW

What is Prometheus?

- What is Prometheus?
  - Features
  - What are metrics ?
  - Components
  - Architecture
- When does it fit?
- When does it not fit?

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes.

Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

For more elaborate overviews of Prometheus, see the resources linked from the media section.

Features

Prometheus's main features are:

- a multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- no reliance on distributed storage; single server nodes are autonomous
- time series collection happens via a pull model over HTTP
- pushing time series is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support

What are metrics ?

In layperson terms, *metrics* are numeric measurements. *Time series* means that changes are recorded over time. What users want to measure differs from application to application. For a web server it might be request times, for a database it might be number of active connections or number of active queries etc.

Metrics play an important role in understanding why your application is working in a certain way. Let's assume you are running a web application and find that the application is slow. You will need some information to find out what is happening with your application. For example the application can become slow when the number of requests are high. If you have the request count metric you can spot the reason and increase the number of servers to handle the load.

核心组件

Prometheus Server： 主要用于抓取数据和存储时序数据，另外还提供查询和 Alert Rule 配置管理。
client libraries：用于对接 Prometheus Server, 可以查询和上报数据。
push gateway ：用于批量，短期的监控数据的汇总节点，主要用于业务数据汇报等。
各种汇报数据的 exporters ：例如汇报机器数据的 node_exporter, 汇报 MongoDB 信息的 MongoDB exporter 等等。
以及：用于告警通知管理的 alertmanager 。

Components

The Prometheus ecosystem consists of multiple components, many of which are optional:

- the main Prometheus server which scrapes and stores time series data
- client libraries for instrumenting application code
- a push gateway for supporting short-lived jobs
- special-purpose exporters for services like HAProxy, StatsD, Graphite, etc.
- an alertmanager to handle alerts
- various support tools

Most Prometheus components are written in Go, making them easy to build and deploy as static binaries.

Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

When does it fit?

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength.

Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems. Each Prometheus server is standalone, not depending on network storage or other remote services. You can rely on it when other parts of your infrastructure are broken, and you do not need to setup extensive infrastructure to use it.

When does it not fit?

Prometheus values reliability. You can always view what statistics are available about your system, even under failure conditions. If you need 100% accuracy, such as for per-request billing, Prometheus is not a good choice as the collected data will likely not be detailed and complete enough. In such a case you would be best off using some other system to collect and analyze the data for billing, and Prometheus for the rest of your monitoring.

大致使用逻辑是这样：
Prometheus server 定期从静态配置的 targets 或者服务发现的 targets 拉取数据。

当新拉取的数据大于配置内存缓存区的时候，Prometheus 会将数据持久化到磁盘（如果使用 remote storage 将持久化到云端）。

Prometheus 可以配置 rules，然后定时查询数据，当条件触发的时候，会将 alert 推送到配置的 Alertmanager。

Alertmanager 收到警告的时候，可以根据配置，聚合，去重，降噪，最后发送警告。可以使用 API， Prometheus Console 或者 Grafana 查询和聚合数据。

Prometheus 的数据是基于时序的 float64 的值，如果你的数据值有更多类型，无法满足。

Prometheus 不适合做审计计费，因为它的数据是按一定时间采集的，关注的更多是系统的运行瞬时状态以及趋势，即使有少量数据没有采集也能容忍，但是审计计费需要记录每个请求，并且数据长期存储，这个 Prometheus 无法满足，可能需要采用专门的审计系统。

为什么选择 Prometheus

在前言中，简单介绍了我们选择 Prometheus 的理由，以及使用后给我们带来的好处。
在这里主要和其他监控方案对比，方便大家更好的了解 Prometheus。

Prometheus vs Zabbix

Zabbix 使用的是 C 和 PHP, Prometheus 使用 Golang, 整体而言 Prometheus 运行速度更快一点。
Zabbix 属于传统主机监控，主要用于物理主机，交换机，网络等监控，Prometheus 不仅适用主机监控，还适用于 Cloud, SaaS, Openstack，Container 监控。
Zabbix 在传统主机监控方面，有更丰富的插件。
Zabbix 可以在 WebGui 中配置很多事情，但是 Prometheus 需要手动修改文件配置。

Prometheus vs Graphite

功能较少，它专注于两件事，存储时序数据，
可视化数据，其他功能需要安装相关插件，而 Prometheus 属于一站式，提供告警和趋势分析的常见功能，它提供更强的数据存储和查询能力。
在水平扩展方案以及数据存储周期上，Graphite 做的更好。

Prometheus vs InfluxDB

是一个开源的时序数据库，主要用于存储数据，如果想搭建监控告警系统，需要依赖其他系统。

InfluxDB 在存储水平扩展以及高可用方面做的更好, 毕竟核心是数据库。

Prometheus vs Nagios

数据不支持自定义 Labels, 不支持查询，告警也不支持去噪，分组, 没有数据存储，如果想查询历史状态，需要安装插件。

Nagios 是上世纪 90 年代的监控系统，比较适合小集群或静态系统的监控，显然 Nagios 太古老了，很多特性都没有，相比之下Prometheus 要优秀很多。

总结

Prometheus 属于一站式监控告警平台，依赖少，功能齐全。

Prometheus 支持对云或容器的监控，其他系统主要对主机监控。

Prometheus 数据查询语句表现力更强大，内置更强大的统计函数。

Prometheus 在数据存储扩展性以及持久性上没有 InfluxDB，OpenTSDB好。

数据模型

Prometheus 存储的是, 即按照相同时序(相同的名字和标签)，以时间维度存储连续的数据的集合。

时序索引

时序(time series) 是由名字(Metric name)和一组 key/value 标签定义的，具有相同的名字以及标签属于相同时序。

时序的名字由 ASCII 字符，数字，下划线，以及冒号组成， 其名字应该具有语义化，一般表示一个可以度量的指标，例如: http_requests_total, 可以表示 http 请求的总数。

标签名称由 ASCII 字符，数字，以及下划线组成， 其中 __ 开头属于 Prometheus 内部保留，标签的值可以是任何 Unicode 字符，支持中文。

时序样本

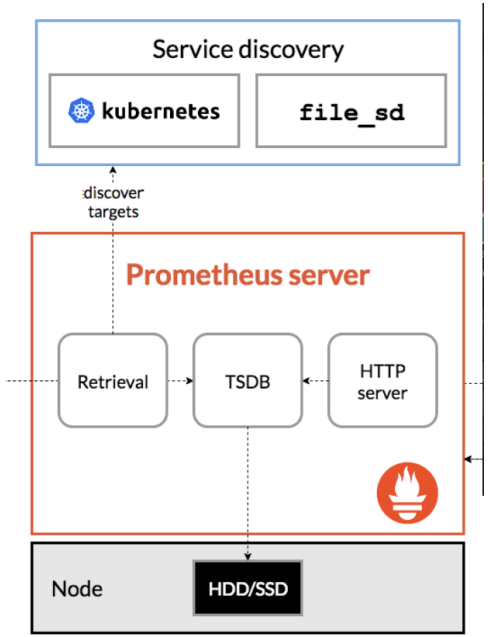
按照某个时序以时间维度采集的数据，称之为样本，其值包含：

一个 float64 值

一个毫秒级的 unix 时间戳

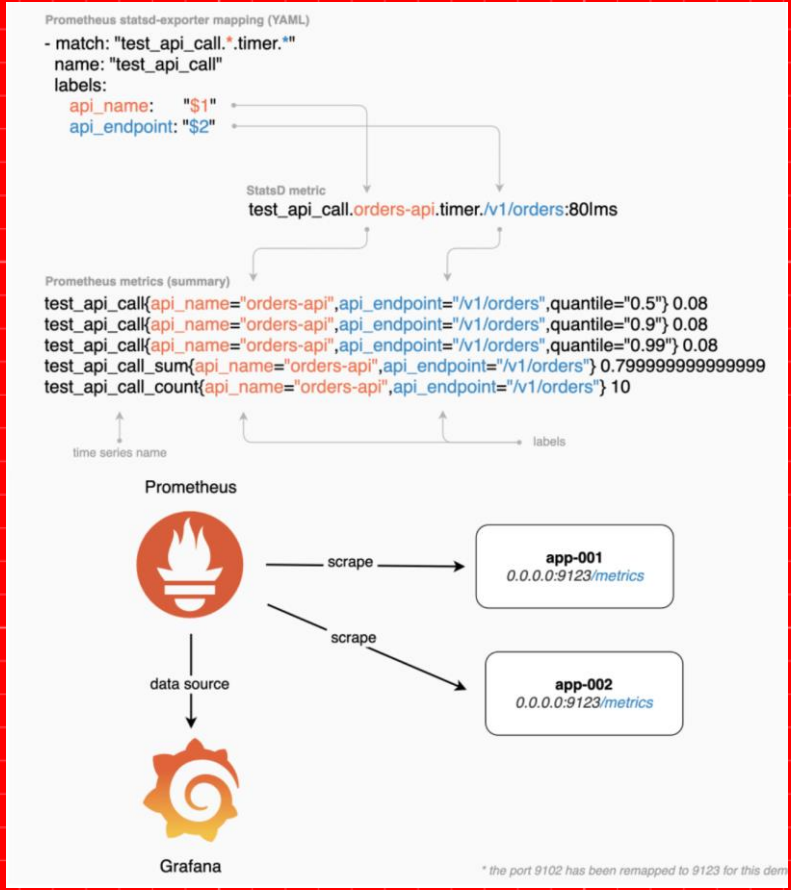
服务发现

prometheus的服务发现能自动检测分类，并且可以识别新节点和变更节点，目前prometheus已经支持了很多常见的自动发现服务，比如：
consul,ec2,kubernetes_sd_config,file_sd_config,dns等
比较常用的就是基于文件的自动发现，和k8s的自动发现。



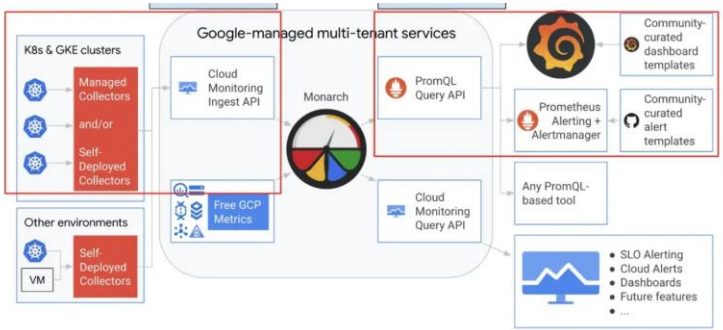
Exporter

在 Prometheus 中负责数据汇报的程序统一叫做 Exporter, 而不同的 Exporter 负责不同的业务。 它们具有统一命名格式, 即 xx_exporter, 例如负责主机信息收集的 node_exporter。
Prometheus 社区已经提供了很多 exporter, 详情请参考这里 。



Google Cloud Managed Service for Prometheus

Google引领Kubernetes生态系已经不是新闻了，但在GCP的这些年，一直没有使用CNCF里最多人使用的监控技术：Prometheus，这一点一直很困扰我。 我自己的解读是因为Cloud上很久以前已经采用了StackDriver的监控机制，同时也有非常多不属于容器的服务（BigQuery， CloudSQL等），如同Cloudwatch之于AWS，Google Cloud的整体监控就是StackDriver （如 Monarch），并提供类似PromQL的MQL语法，虽然可用，但仍觉得有些美中不足的地方。



Google Cloud Managed Service for Prometheus

GKE内建了使用Cloud Monitoring收集Node，Pod，仅需指定对的参数就可以自动开启，这些参数也支持以PromQL方式 查询。 但若希望连收集部分，都要使用PodMonitoring机制，我们也可以部署Node Exporter + Kube-State-Metrics等服务与对应的ClusterPodMonitoring/ PodMonitoring，Deploy FrontEnd to Simulate Prometheus DB (for Grafana)
至于整合的部分，GMP还很贴心的提供一个兼容于现有Prometheus Query API的Frontend 服务

```yaml
apiVersion: monitoring.googleapis.com/v1
kind: PodMonitoring
metadata:
  name: prom-example
spec:
  selector:
    matchLabels:
      app: prom-example
  endpoints:
  - port: metrics
    interval: 30s
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prom-example
  labels:
    app: prom-example
spec:
  selector:
    matchLabels:
      app: prom-example
  replicas: 3
  template:
    metadata:
      labels:
        app: prom-example
    spec:
      nodeSelector:
        kubernetes.io/os: linux
        kubernetes.io/arch: amd64
      containers:
      - image: nilebox/prometheus-example-app@sha256:dab60d038c5d6915af5bcbe5f0279a22b95a8c8be2541
        name: prom-example
        ports:
        - name: metrics
          containerPort: 1234
        command:
        - "/main"
        - "--process-metrics"
        - "--go-metrics"
```

Why Changed?

很多人会问说，除了Prometheus的开源性，有其他的诱因让我们由Cloud Monitoring换到GMP吗？ 事实上是有的，主要在于成本

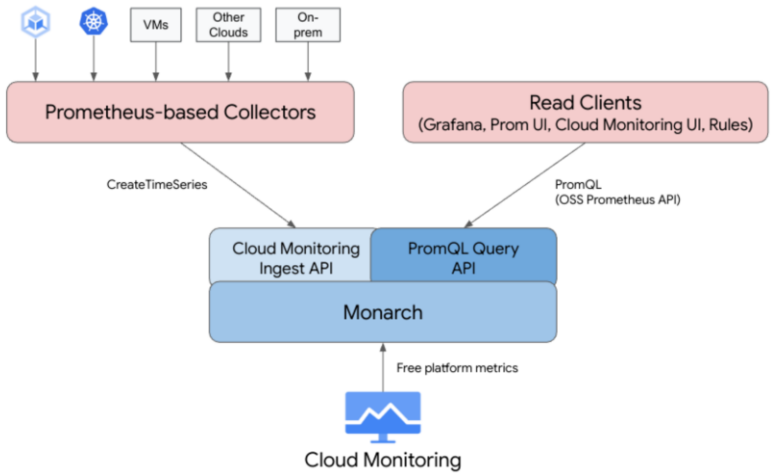
。
举个范例，假设每天收集1，000个参数，每30秒一次，一天就会产生约2，880，000个采样， 一个月约86.4M 个采样。 每个采样 50 字节，

Cloud Monitoring会需要4.32GB/月的量，大概0.151x4，320=653美元.
但换算成GMP，0.15 x 86.4=13美元
是不是很有诱惑力呢？

Self-Deployed Collection顾名思义是通过部署一个Prometheus Instance在自己的集群内，在Self-Deploy模式裡，我们所部署的Prometheus Instance是一个GCP定制过的镜像，与OSS差异最大的是这个Instance只负责通过Prometheus的方式收集数据，然后再丢入Google的时序数据库：Monarch裡，作为管理者，我们Leverage Google Monitoring Service写入资讯，而不需要维护对应的本地存储。

总结1:

Google Cloud Managed Service for Prometheus 是针对 Prometheus 指标的 Google Cloud 全代管式多云解决方案。您可以使用 Prometheus 全局监控工作负载并发出提醒，而无需大规模手动管理和操作 Prometheus，无需手动管理本地存储系统。大大节约运维成本。



总结2：

Managed Service for Prometheus 会从 Prometheus 导出器中收集指标，并支持使用 PromQL 全局查询数据，这意味着您可以继续使用任何现有的 Grafana 信息中心、基于 PromQL 的提醒和工作流。它与混合云和多云兼容，可以监控 Kubernetes 和虚拟机工作负载，将数据保留 24 个月，并与上游 Prometheus 保持兼容以维持可移植性。您还可以使用 PromQL 在 Cloud Monitoring 中查询超过 1500 个免费指标（包括免费的 GKE 系统指标），作为 Prometheus 监控的补充。