Kubernetes监控方案

前言

- 本文主要介绍kubernetes集群监控的三大主流方案，包括weave scope，Heapster和Prometheus，着重介绍了Prometheus的架构，工作流程和多维度数据模型特点。

目标

- 学完本课程后，您将能够：
  - 列举kubernetes几大主流的监控方案
  - 描述Prometheus监控方案的组成架构，各组件的作用
  - 描述什么是Prometheus的多维度数据模型

目录

Kubernetes常用监控

- Weave Scope
- Heapster
- Prometheus

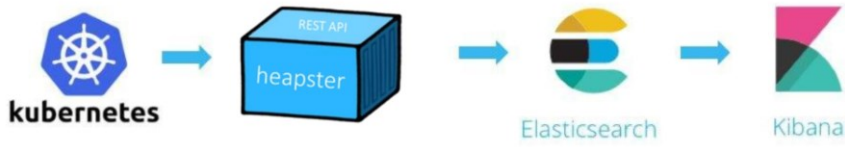
- Heapster是kubernetes项目下的一个监控项目
- Weave scope是weaveworks开发的监控项目
- Prometheus是CNCF下欢迎度仅次于kubernetes的项目

Weave Scope

- Scope是weave公司开源的用于监控，可视化，管理kubernetes集群的一个类似于Bashbaord的UI系统。有以下几大特点：
  - 拓扑映射
  - 实时应用和容器指标
  - 在线排障并管理容器
  - 强大的搜索功能

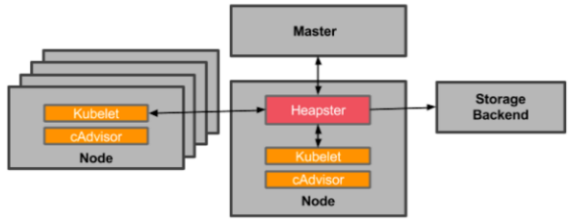
Heapster

- Heapster是Kubernetes原生的集群监控方案。天然的支持Kubernetes和CoreOS，Heapster以Pod的形式运行，它会自动发现集群节点、从节点上的Kubelet获取监控数据。
- Kubernetes有个出名的监控agent---cAdvisor（在新版本中，Kubernetes已经将cAdvisor功能集成到kubelet组件中，kubelet中的数据便是从它获取的），可以监控并汇总出有价值的性能数据(Metrics)：CPU、内存、网络流量等。
- 这些数据被Heapster收集并输出到外部存储，如InfluxDB，最后就可以通过相应的UI界面显示出来，如grafana。 另外Heapster的数据源和外部存储都是可插拔的，所以可以很灵活的组建出很多监控方案，如：Heapster+ElasticSearch+Kibana等等。



Heapster

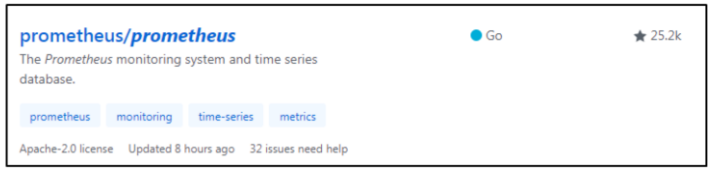
- Heapster工作流程简述如下：
  - Heapster首先从kubernetes Master获取集群中所有Node的信息
  - 通过这些Node上的kubelet获取有用数据，而kubelet本身的数据则是从cAdvisor得到。
  - 获取到的数据都被推到Heapster配置的后端存储中，同时支持数据可视化，比如InfluxDB + grafana的解决方案中，InfluxDB就是广泛应用的时序数据库，grafana就是可视化工具。



- 图片出处：heapster项目官网

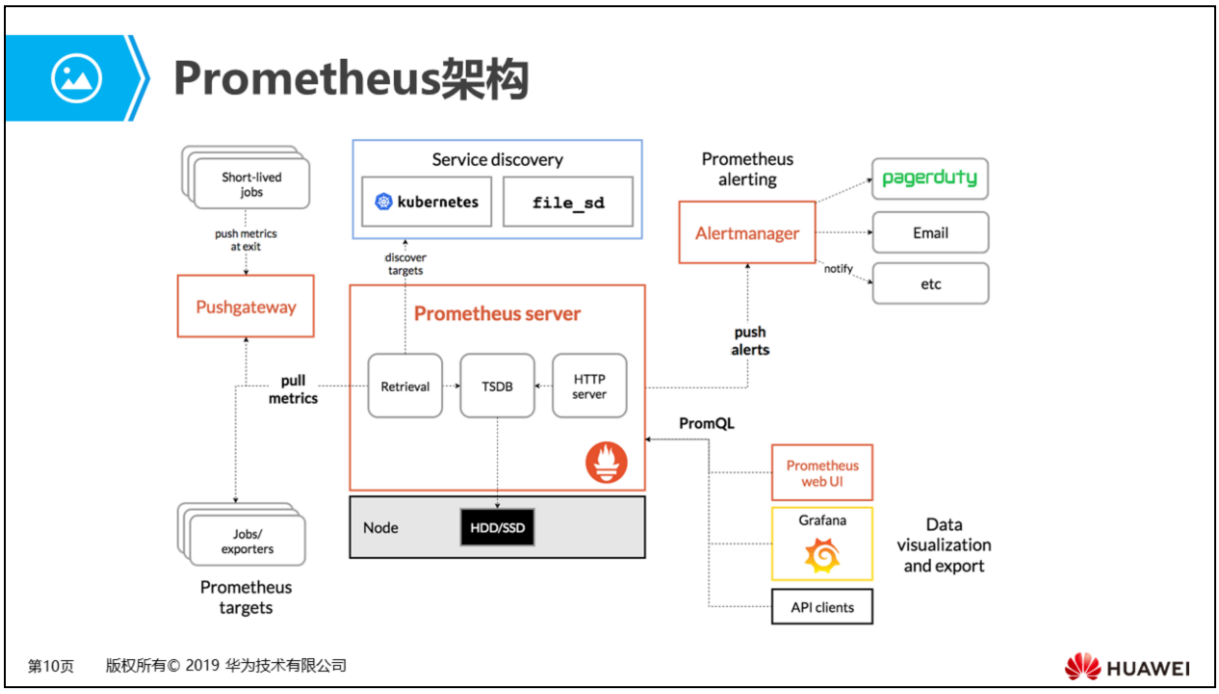
Prometheus

- Heapster与Weave scope主要侧重于监控Node和Pod，这些数据当然很重要，但是还不够。对于集群本身的运行状况，比如Kubernetes的API Server、Scheduler、Controller Manager等管理组件是否正常工作，负荷如何，前面提到的两种监控方案就无法满足。
- 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25k+关注，以及很多大公司的使用；2016年Prometheus成为继Kubernetes后，第二名CNCF成员。作为新一代开源解决方案，很多理念与Google SRE运维之道不谋而合。



- Prometheus项目与Kubernetes项目一样，都来自Google的Borg体系，它的原型系统叫做BorgMon，是一个几乎与Borg同时诞生的内部监控系统。而Prometheus项目的发起原因也跟Kubernetes很类似，都是希望通过对用户更友好的方式，将Google内部系统的设计理念，传递给用户和开发者。
- SRE是DevOps的实践者，工作内容和职责和传统运维工程师差不多，但是SRE的手段更加自动化和高效，这种高效来自于自动化工具、监控工具的支撑。

目录



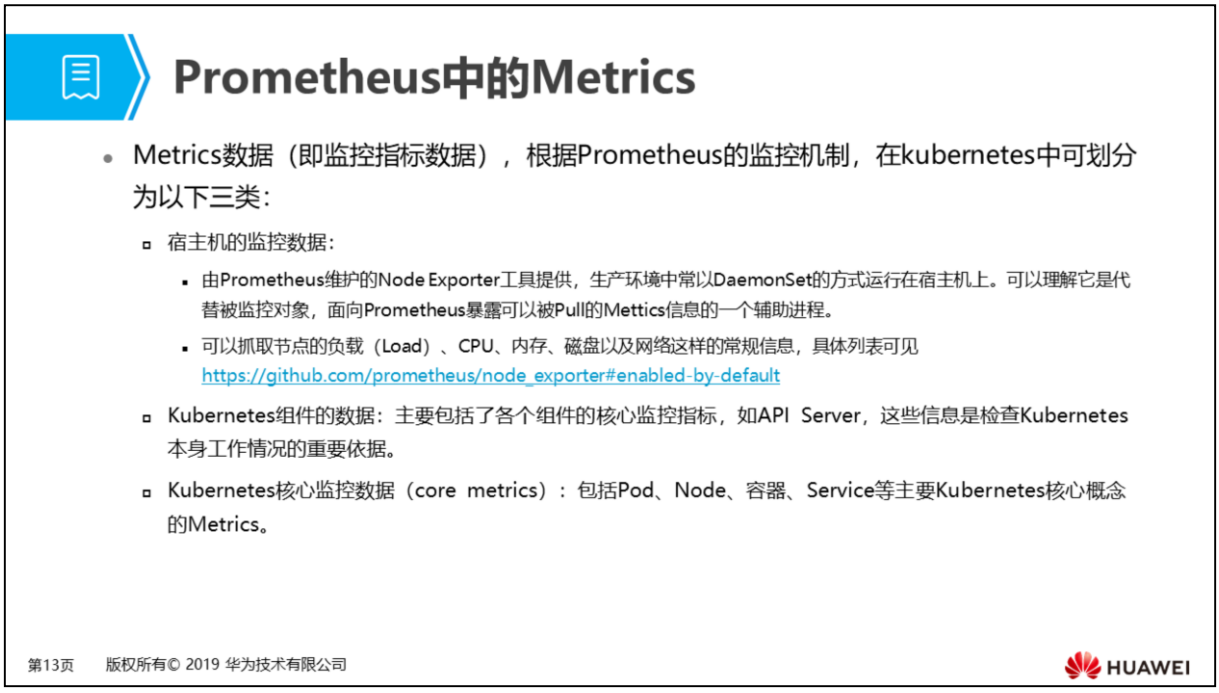
- Prometheus的项目核心，是使用Pull（抓取）的方式去收集被监控对象的Metrics数据（监控指标数据），然后，再把这些数据保存在一个TSDB（时间序列数据库，比如OpenTSDB，InfluxDB）当中，以便后续可以按时间进行检索。
- 有了这套核心监控机制，Prometheus剩下的组件就是用来配合这套机制的运行，比如Pushgateway，可以允许被监控对象以Push的方式向Prometheus推送Metrics数据。而Aletmanager，则可以根据Metrics信息灵活地设置报警。
- 图片出处prometheus官网

Prometheus架构

- Prometheus的主要模块包括：Prometheus server, exporters, Pushgateway, PromQL, Alertmanager以及图形界面。
  - **Prometheus Server**：Prometheus Server用于收集和存储时间序列数据。并提供一套灵活的查询语言（PromQL）供用户使用。
  - **Exporter**：Exporter负责收集目标对象（host, container...）的性能数据，并通过HTTP接口供Prometheus Server获取。
  - **可视化组件**：监控数据的可视化展现对于监控方案至关重要。以前Prometheus自己开发了一套工具，不过后来废弃了，因为开源社区出现了更为优秀的产品Grafana。Grafana能够与Prometheus无缝集成，提供完美的数据展示能力。

Prometheus架构

- Prometheus的主要模块包括：Prometheus server, exporters, Pushgateway, PromQL, Alertmanager以及图形界面。
  - **Alertmanager**：用户可以定义基于监控数据的告警规则，规则会触发告警。一旦Alermanager收到告警，会通过预定义的方式发出告警通知。支持的方式包括Email、PagerDuty、Webhook等.
  - **Push Gateway**：主要用于短期的jobs。由于这类jobs存在时间较短，可能在Prometheus来pull之前就消失了。为此，这些jobs可以直接向Prometheus server端推送它们的metrics。这种方式主要用于服务层面的metrics，对于机器层面的metrices，需要使用node exporter。



- 其中，容器相关的Metrics主要来自kubelet内置的cAdvisor服务，在kubelet启动后，cAdvisor也随之启动，而它能提供的信息，可以细化到每一个容器的CPU、文件系统、内存、网络等资源的使用情况。

Prometheus工作流程

- 其工作的工作流程简述为4个步骤：
  - Prometheus server定期从配置好的jobs或者exporters中拉取metrics，或者接收来自Pushgateway发过来的metrics，或者从其他的Prometheus server中拉metrics
  - Prometheus server在本地存储收集到的metrics，并运行已定义好的alert.rules，记录新的时间序列或者向Alertmanager推送警报。
  - Alertmanager根据配置文件，对接收到的警报进行处理，发出告警。
  - 在图形界面中，可视化采集数据。

Prometheus数据模型

- Prometheus中存储的数据为时间序列，是由metric的名字和一系列的标签（键值对）唯一标识的，不同的标签则代表不同的时间序列。
  - metric名字：该名字应该具有语义，一般用于表示metric的功能，例如：http_requests_total，表示 http请求的总数。其中，metric名字由ASCII字符，数字，下划线，以及冒号组成，且必须满足正则表达式[a-zA-Z_:][a-zA-Z0-9_:]*。
  - 标签：使同一个时间序列有了不同维度的识别。例如：http_requests_total{method="Get"}表示所有http请求中的Get请求。当method="post" 时，则为新的一个metric。标签中的键由ASCII字符，数字，以及下划线组成，且必须满足正则表达式 [a-zA-Z_:][a-zA-Z0-9_:]*。

Prometheus数据模型

- Prometheus中存储的数据为时间序列，是由metric的名字和一系列的标签（键值对）唯一标识的，不同的标签则代表不同的时间序列。
  - 样本：实际的时间序列，每个序列包括一个float64的值和一个毫秒级的时间戳。
  - 格式：<metric name>{<label name>=<label value>, ...}，例如：http_requests_total{method="POST",endpoint="/api/tracks"}。

Prometheus的核心：多维度模型

- Prometheus的标签功能给数据监控带来了强大的扩展性：
  - 传统方式下：当我们需要监控容器webapp1的内存使用情况，最传统和典型的方法是定义一个指标container_memory_usage_webapp1来记录webapp1内存使用数据，假如有多个类似的容器需要监控，则需要增加新的指标，如：
    - container_memory_usage_webapp2、 container_memory_usage_webapp3
  - 进阶模式：比如Graphite这类高级的监控方案采用了更优雅的层次化模型，对于上面的需求，Graphite会定义指标，如：
    - container.memory_usage.webapp1、container.memory_usage.webapp2、container.memory_usage.webapp3
    - 对于这种层级化模型，我们就可以使用container.memory_usage_bytes.webapp*来得到所有webapp的内存使用数据。
    - 此外，Graphite还支持sum() 等函数对指标进行计算和处理，比如sum(container.memory_usage_bytes.webapp*) 可以得到所有webapp容器占用的总内存量

Prometheus的核心：多维度模型

- Prometheus的标签功能给数据监控带来了强大的扩展性：
  - 多维度模型：Prometheus只需要定义一个全局的指标container_memory_usage_bytes，然后通过添加不同的维度数据来满足不同的业务需求，如下表:

time	Container_memory_usage_bytes	container_name	image	env
00:01:00	37738736	webapp1	webapp:1.2	test
00:02:00	37736822	webapp2	webapp:1.2	test
00:03:00	37723425	webapp3	webapp:1.2	test
.....	.....	.....	.....	.....

Prometheus的核心：多维度模型

- Prometheus的标签功能给数据监控带来了强大的扩展性：
  - 后面三列（container_name,image,env）就是数据的三个维度：
    - container_name定义了容器的名字（表中是webapp1）
    - env定义了部署的环境（表中是test）
    - image定义了容器使用的镜像（表中是webapp:1.2）
  - 以表中为例，有了这三个维度，就可以进行组合满足很多业务监控需求，比如
    - 计算webapp2的平均内存使用情况：avg（container_memory_usage_bytes{container_name=“webapp2”})
    - 计算运行webapp:1.2镜像的所有容器内存使用总量：sum(container_memory_usage_bytes{image=“mycom/webapp:1.3”})
    - 统计test运行环境中webapp1容器内存使用总量：sum(container_memory_usage_bytes{container_name=“webapp1”,env=“test”})

实验任务

- 实验任务
  - 请按照手册2.19章节完成日志相关实验，包括：
    - Prometheus的简单使用

本章总结

- 本章主要介绍了三种kubernetes集群的监控解决方案，包括：
  - Weave Scope
  - Heapster
  - Prometheus


谢 谢
www.huawei.com