Prometheus监控系统

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

话题

- 监控系统基础概念
- Prometheus快速入门
- 部署及运行Prometheus
- 使用Exporters进行节点及应用监控
- PromQL基础
- PromQL进阶
- 服务发现
- 查询持久化及可视化
- Alertmanager与告警规则
- Prometheus Server高可用
- Alertmanager高可用

监控系统基础

讲师：马永亮（马哥）
http://www.magedu.com
https://github.com/ikubernetes/

监控系统发展史

- SNMP监控时代
- 当今的监控系统
- 未来的监控系统

监控系统组件

- 监控系统功能组件
  - 指标数据采集（抓取）
  - 指标数据存储
  - 指标数据趋势分析及可视化
  - 告警

监控体系

- 监控体系（自底向上）
  - 系统层监控
    - 系统监控：CPU、Load、Memory、Swap、Disk IO、Processes、Kernel Parameters、……
    - 网络监控：网络设备、工作负载、网络延迟、丢包率、……
  - 中间件及基础设施类系统监控
    - 消息中间件：Kafka、RocketMQ和RabbitMQ等；
    - Web服务容器：Tomcat和Jetty等；
    - 数据库及缓存系统：MySQL、PostgreSQL、MogoDB、ElasticSearch和Redis等；
    - 数据库连接池：ShardingSpere等；
    - 存储系统：Ceph等
  - 应用层监控
    - 用于衡量应用程序代码的状态和性能
  - 业务层监控
    - 用于衡量应用程序的价值，例如电子商务网站上的销售量
    - QPS、DAU日活、转化率；
    - 业务接口：登录数、注册数、订单量、搜索量和支付量等；

云原生时代的可观测性

- 可观测性系统
  - 指标监控（Metrics）：随时间推移产生的一些与监控相关的可聚合数据点；
  - 日志监控（Logging）：离散式的日志或事件；
  - 链路跟踪（Tracing）：分布式应用调用链跟踪；
- CNCF将可观测性和数据分析归类一个单独的类别，且划分成了4个子类
  - 监控系统：以Prometheus等为代表；
  - 日志系统：以ElasticStack和PLG Stack等为代表；
  - 分布式调用链跟踪系统：以Zipkin、Jaeger、SkyWalking、Pinpoint等为代表；
  - 混沌工程系统：以ChaosMonkey和ChaosBlade等为代表

著名的监控方法论

- Google的四个黄金指标
  - 常用于在服务级别帮助衡量终端用户体验、服务中断、业务影响等层面的问题
  - 适用于应用及服务监控
- Netflix的USE方法
  - 全称为“Utilization Saturation and Errors Method”
  - 主要用于分析系统性能问题，可以指导用户快速识别资源瓶颈以及错误的方法
  - 应用于主机指标监控
- Weave Cloud的RED方法
  - Weave Cloud基于Google的四个黄金指标的原则下结合Prometheus以及Kubernetes容器实践，细化和总结的方法论，特别适合于云原生应用以及微服务架构应用的监控和度量

黄金指标

- 四个黄金指标源自Google的SRE一书
  - 延迟（Latency）
    - 服务请求所需要的时长，例如HTTP请求平均延迟；
    - 需要区分失败请求和成功请求；
  - 流量（Traffic）
    - 衡量服务的容量需求，例如每秒处理的HTTP请求数或者数据库系统的事务数量；
  - 错误（Errors）
    - 请求失败的速率，用于衡量错误发生的情况
    - 例如，HTTP 500错误数等显式失败，返回错误内容或无效内容等隐式失败，以及由策略原因导致的失败（例如强制要求响应时间超过30毫秒的请求视为错误）；
  - 饱和度（Saturation）
    - 衡量资源的使用情况，用于表达应用程序有多“满”
    - 例如内存、CPU、I/O、磁盘等资源的使用量

USE方法

- USE方法由Netflix的内核和性能工程师Rendan Gregg提出，主要用于分析系统性能问题
  - 使用率(Utilization)
    - 关注系统资源的使用情况。 这里的资源主要包括但不限于：CPU，内存，网络，磁盘等等
    - 100%的使用率通常是系统性能瓶颈的标志
  - 饱和度(Saturation)
    - 例如CPU的平均运行排队长度，这里主要是针对资源的饱和度(注意，不同于4大黄金信号)
    - 任何资源在某种程度上的饱和都可能导致系统性能的下降
  - 错误(Errors)
    - 错误计数
    - 例如：“网卡在数据包传输过程中检测到的以太网网络冲突了14次”

RED方法

- RED方法是Weave Cloud在基于Google的4个黄金指标的原则下结合Prometheus以及Kubernetes容器实践，细化和总结的方法论，特别适合于云原生应用以及微服务架构应用的监控和度量
- 在四大黄金指标的原则下，RED方法可以有效地帮助用户衡量云原生以及微服务应用下的用户体验问题；
- RED方法主要关注以下3种关键指标
  - (Request)Rate：每秒钟接收的请求数；
  - (Request)Errors：每秒失败的请求数；
  - (Request)Duration：每个请求所花费的时长；

Prometheus快速入门

讲师：马永亮（马哥）

http://www.magedu.com

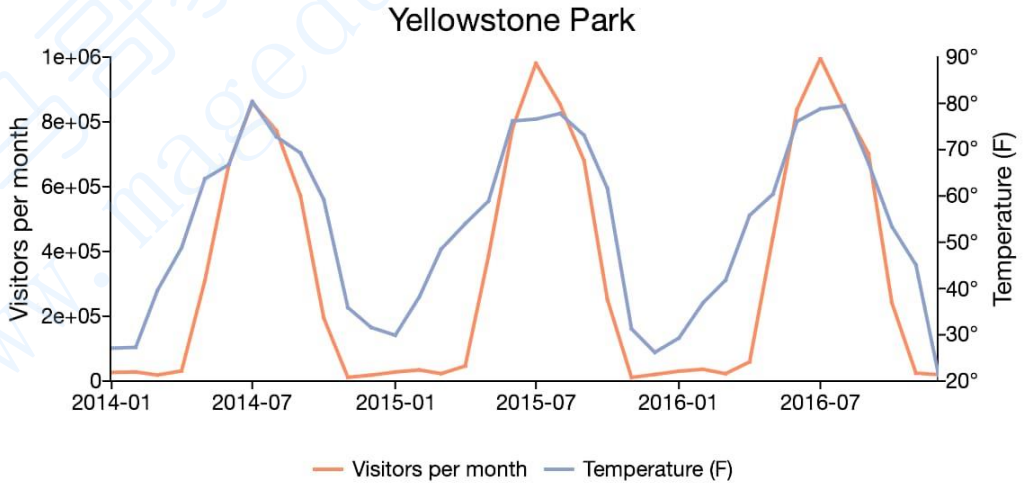
https://github.com/ikubernetes/

What is Prometheus Monitoring?

- 首先，Prometheus是一款**时序（time series）数据库**；但它的功能却并非止步于TSDB，而是一款设计用于进行目标（Target）监控的关键组件；
- 结合生态系统内的其它组件，例如Pushgateway、Altermanager和Grafana等，可构成一个完整的IT监控系统；

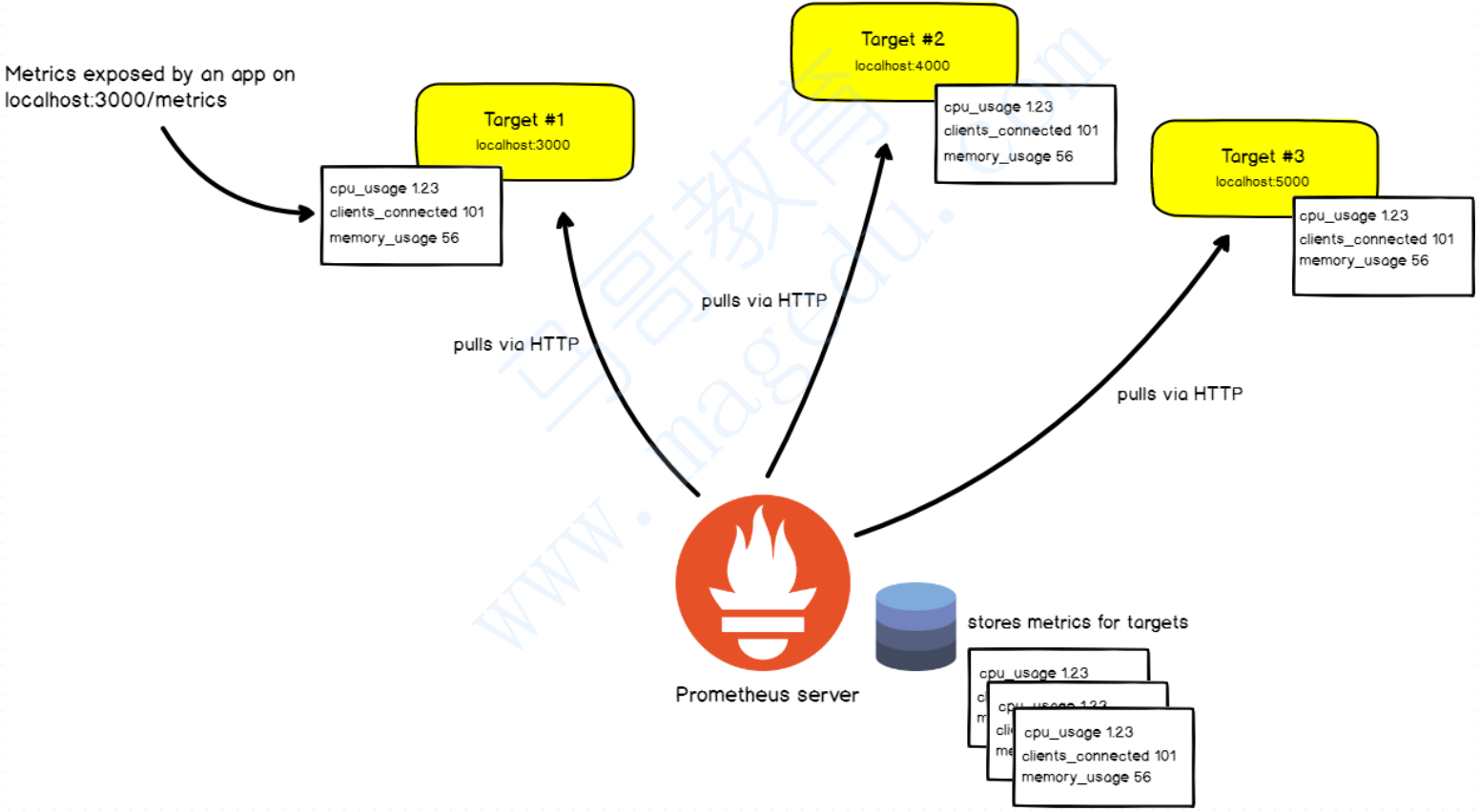
时序数据简介

- 时序数据，是在一段时间内通过重复测量（measurement）而获得的观测值的集合；将这些观测值绘制于图形之上，它会有一个数据轴和一个**时间轴**；
  - As our world gets increasingly instrumented, sensors and systems are constantly emitting a relentless stream of time series data.
  - Weather records, economic indicators and patient health evolution metrics — all are time series data.
- 服务器指标数据、应用程序性能监控数据、网络数据等也都是时序数据；



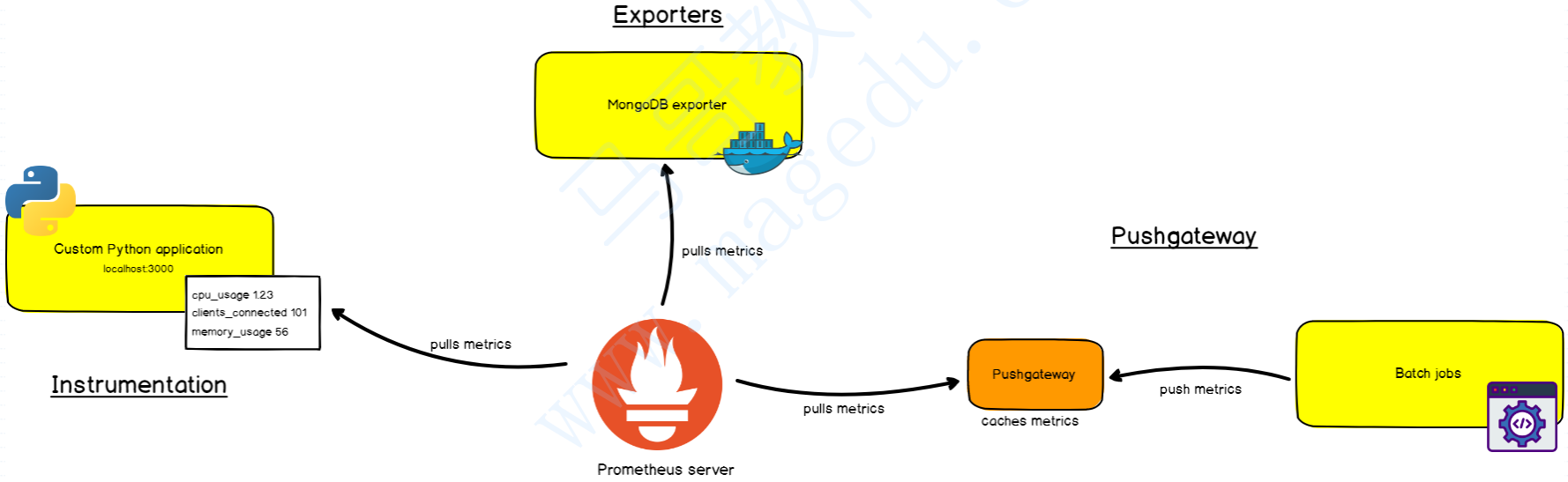
What does Prometheus do?

- 基于HTTP call，从配置文件中指定的网络端点（endpoint）上周期性获取指标数据



How does Prometheus work?

- Prometheus支持通过三种类型的途径从目标上“抓取（Scrape）”指标数据；
  - Exporters
  - Instrumentation
  - Pushgateway

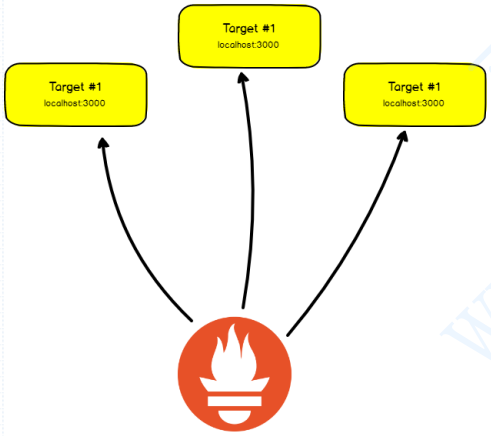


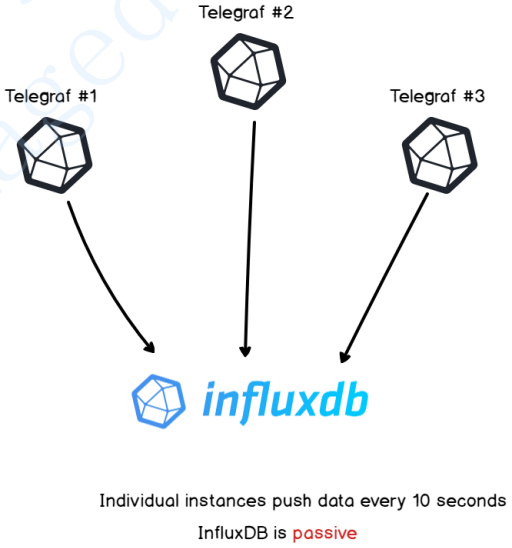
How does Prometheus work?

- Prometheus刮擦指标数据的三个途径；
  - By ‘instrumenting’ your application, meaning that your application will expose Prometheus compatible metrics on a given URL.
  - By using of the prebuilt exporters : Prometheus has an entire collection of exporters for existing technologies.
    - You can for example find prebuilt exporters for Linux machine monitoring (node exporter), for very established databases (SQL exporter or MongoDB exporter) and even for HTTP load balancers (such as the HAProxy exporter).
  - By using the Pushgateway : sometimes you have applications or jobs that do not expose metrics directly.
    - Those applications are either not designed for it (such as batch jobs for example), or you may have made the choice not to expose those metrics directly via your app.

Pull and Push

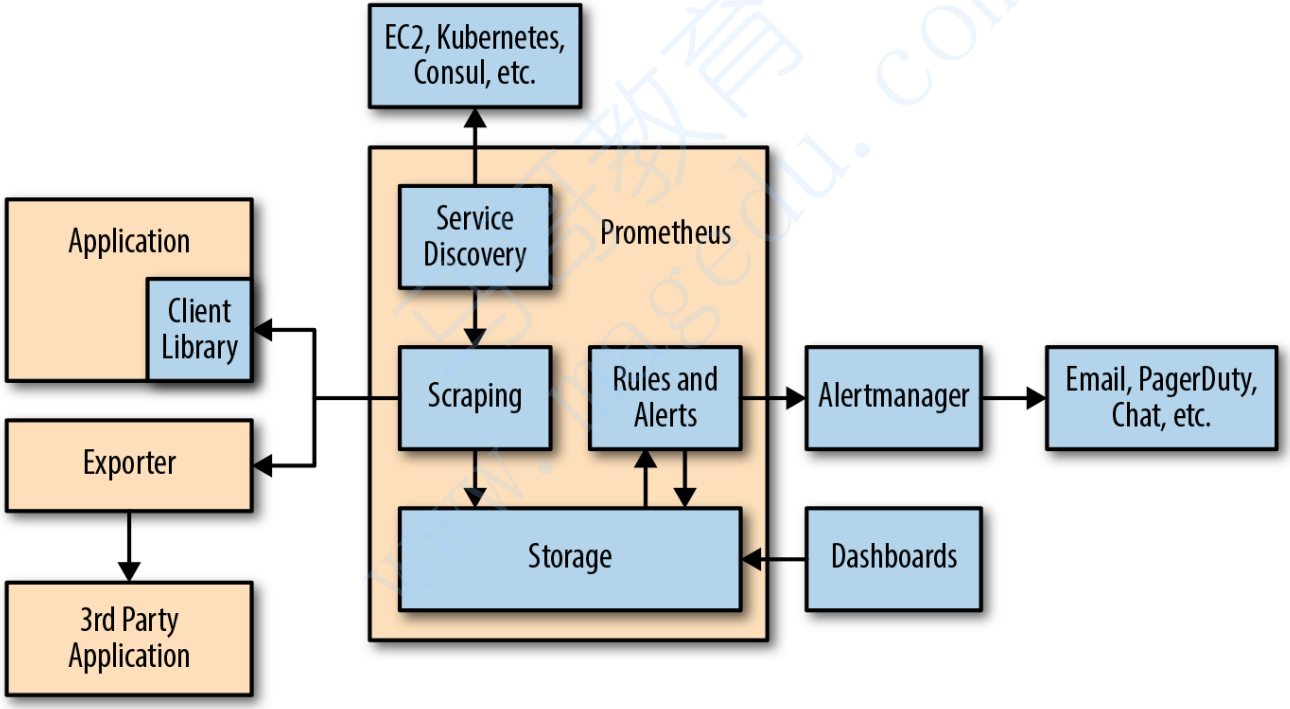
- Prometheus同其它TSDB相比有一个非常典型的特性：它主动从各Target上“拉取（pull）”数据，而非等待被监控端的“推送（push）”；
- 两个方式各有优劣，其中，Pull模型的优势在于：
  - 集中控制：有利于将配置集在Prometheus Server上完成，包括指标及采取速率等；
  - Prometheus的根本目标在于收集在Target上预先完成聚合的聚合型数据，而非一款由事件驱动的存储系统；





Prometheus的生态组件

- Prometheus负责时序型指标数据的采集及存储，但数据的分析、聚合及直观展示以及告警等功能并非由Prometheus Server所负责；

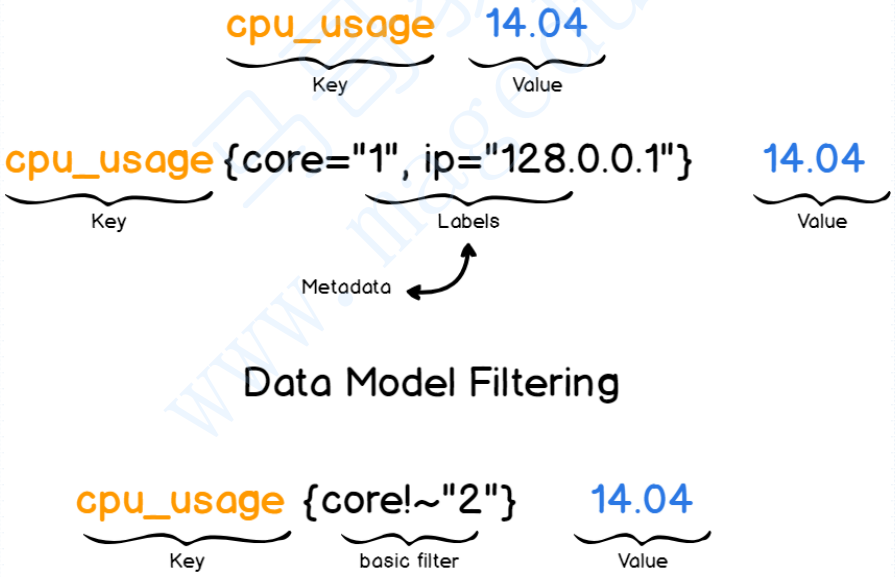


Prometheus的生态组件

- Prometheus 生态圈中包含了多个组件，其中部分组件可选
  - **Prometheus Server**: 收集和存储时间序列数据；
  - **Client Library**: 客户端库，目的在于为那些期望原生提供Instrumentation功能的应用程序提供便捷的开发途径；
  - **Push Gateway**: 接收那些通常由短期作业生成的指标数据的网关，并支持由Prometheus Server进行指标拉取操作；
  - **Exporters**: 用于暴露现有应用程序或服务（不支持Instrumentation）的指标给Prometheus Server；
  - **Alertmanager**: 从Prometheus Server接收到“告警通知”后，通过去重、分组、路由等预处理功能后以高效向用户完成告警信息发送；
  - **Data Visualization**：Prometheus Web UI （Prometheus Server内建），及Grafana等；
  - **Service Discovery**：动态发现待监控的Target，从而完成监控配置的重要组件，在容器化环境中尤为有用；该组件目前由Prometheus Server内建支持；

Prometheus数据模型

- Prometheus仅用于以“键值”形式存储时序式的聚合数据，它并不支持存储文本信息；
  - 其中的“键”称为指标（Metric），它通常意味着CPU速率、内存使用率或分区空闲比例等；
  - 同一指标可能会适配到多个目标或设备，因而它使用“标签”作为元数据，从而为Metric添加更多的信息描述纬度；
  - 这些标签还可以作为过滤器进行指标过滤及聚合运算；

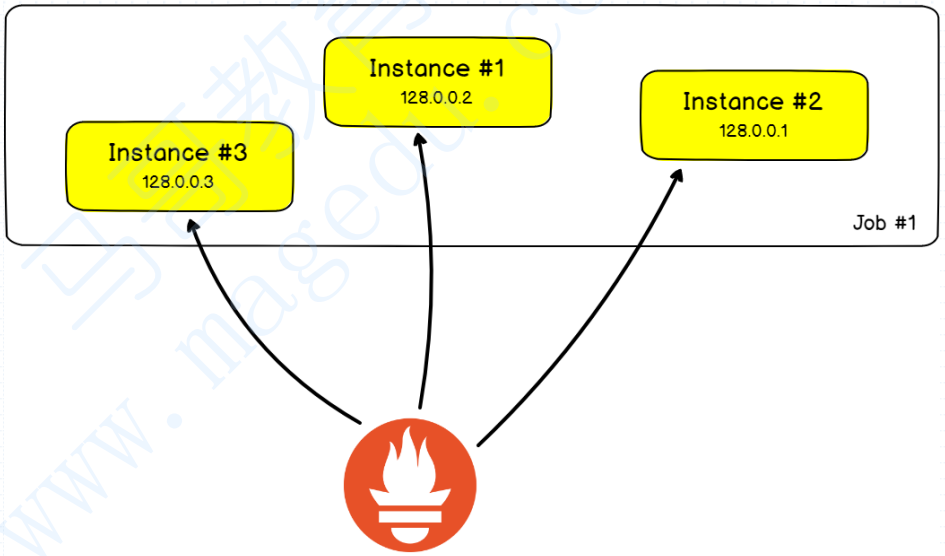


指标类型（Metric Types）

- Prometheus使用4种方法来描述监视的指标
  - Counter：计数器，用于保存单调递增型的数据，例如站点访问次数等；不能为负值，也不支持减少，但可以重置回0；
  - Gauge：仪表盘，用于存储有着起伏特征的指标数据，例如内存空闲大小等；
    - Gauge是Counter的超集；但存在指标数据丢失的可能性时，Counter能让用户确切了解指标随时间的变化状态，而Gauge则可能随时间流逝而精准度越来越低；
  - Histogram：直方图，它会在一段时间范围内对数据进行采样，并将其计入可配置的bucket之中；Histogram能够存储更多的信息，包括样本值分布在每个bucket（bucket自身的可配置）中的数量、所有样本值之和以及总的样本数量，从而Prometheus能够使用内置的函数进行如下操作：
    - 计算样本平均值：以值的总和除以值的数量；
    - 计算样本分位值：分位数有助于了解符合特定标准的数据个数；例如评估响应时长超过1秒钟的请求比例，若超过20%即发送告警等；
  - Summary：摘要，Histogram的扩展类型，但它是直接由被监测端自行聚合计算出分位数，并将计算结果响应给Prometheus Server的样本采集请求；因而，其分位数计算是由由监控端完成；

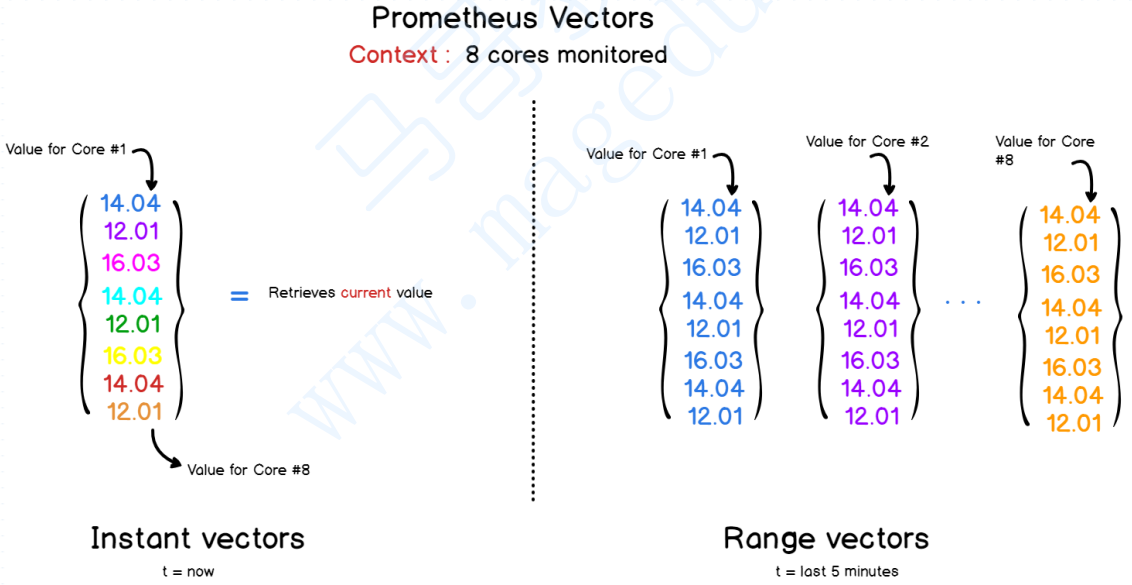
作业（Job）和实例（Instance）

- Instance：能够接收Prometheus Server数据Scrape操作的每个网络端点（endpoint），即为一个Instance（实例）；
- 通常，具有类似功能的Instance的集合称为一个Job，例如一个MySQL主从复制集群中的所有MySQL进程；



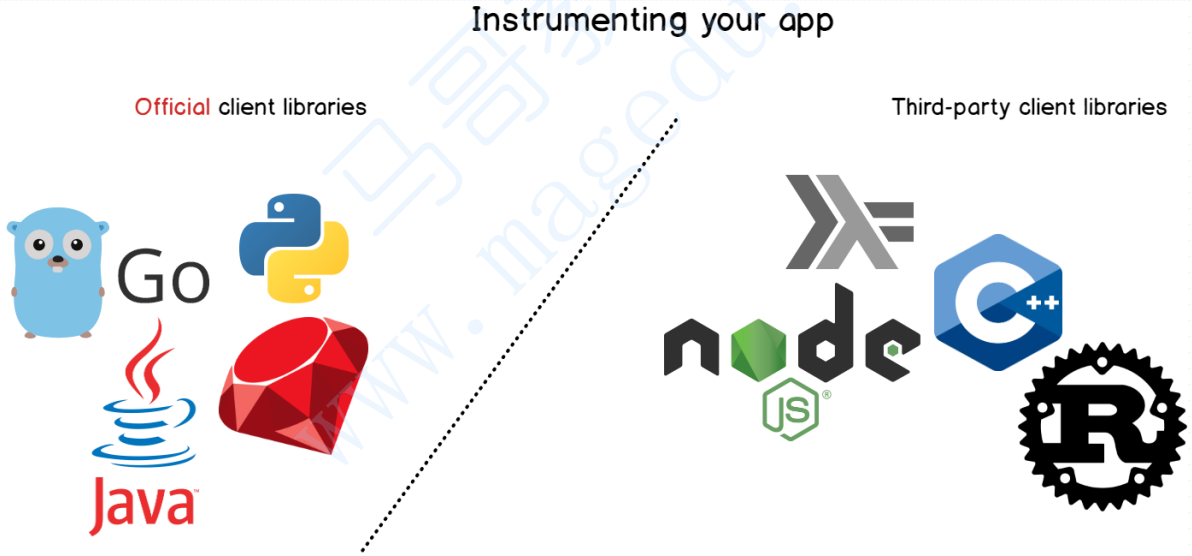
PromQL

- Prometheus提供了内置的数据查询语言**PromQL**（全称为Prometheus Query Language），支持用户进行实时的数据查询及聚合操作；
- PromQL支持处理两种**向量**，并内置提供了一组用于数据处理的**函数**
  - 即时向量：最近一次的时间戳上跟踪的数据指标；
  - 时间范围向量：指定时间范围内的所有时间戳上的数据指标；



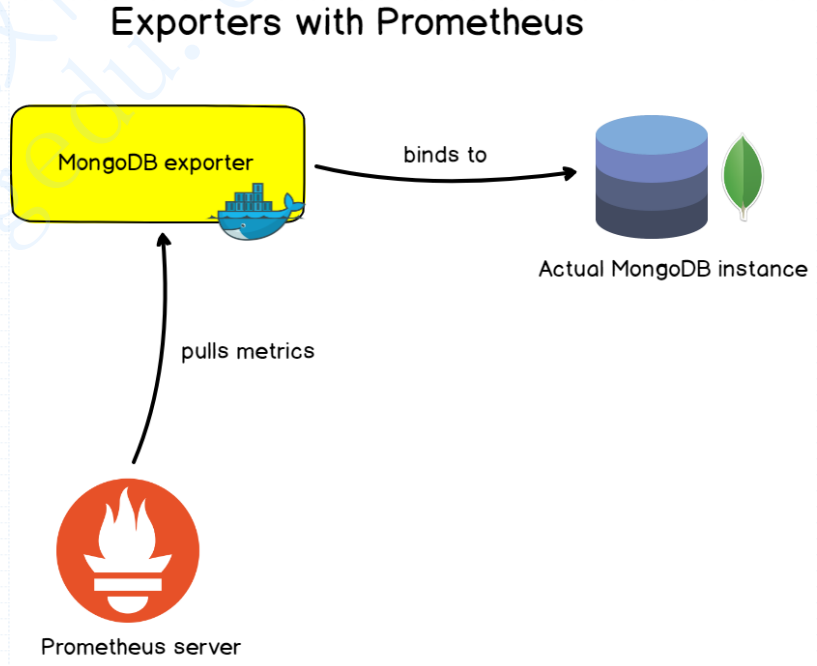
Instrumentation（程序仪表）

- 任何能够支持Scrape指标数据的应用程序都首先要具有一个测量系统；
- 在Prometheus的语境中，Instrumentation是指附加到应用程序中的那些用于暴露程序指标数据的客户端库；
  - 程序员借助于这些客户端库编写代码生成可暴露的指标数据；



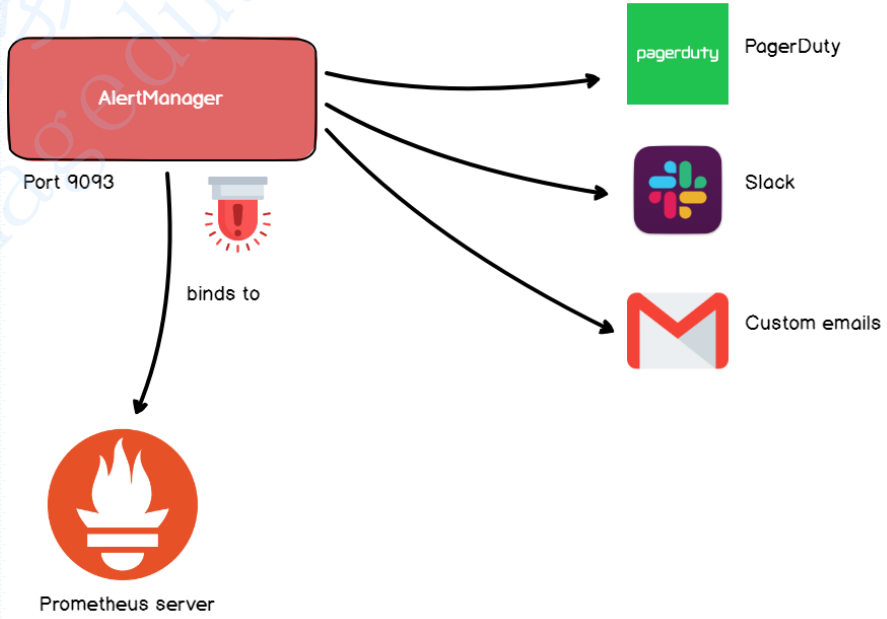
Exporters

- 对于那些未内建Instrumentation，且也不便于自行添加该类组件以暴露指标数据的应用程序来说，常用的办法是于待监控的目标应用程序外部运行一个独立指标暴露程序，该类型的程序即统称为Exporter；
- 换句话说，Exporter负责从目标应用程序上采集和聚合原始格式的数据，并转换或聚合为Prometheus格式的指标向外暴露；
- Prometheus站点上提供了大量的Exporter



Alerts

- 抓取到异常值后，Prometheus支持通过“告警（Alert）”机制向用户发送反馈或警示，以触发用户能够及时采取应对措施；
- Prometheus Server仅负责生成告警指示，具体的告警行为由另一个独立的应用程序AlertManager负责；
  - 告警指示由Prometheus Server基于用户提供的“**告警规则**”周期性计算生成；
  - Alertmanager接收到Prometheus Server发来的告警指示后，基于用户定义的**告警路由**（route）向告警**接收人**（receivers）发送告警信息；



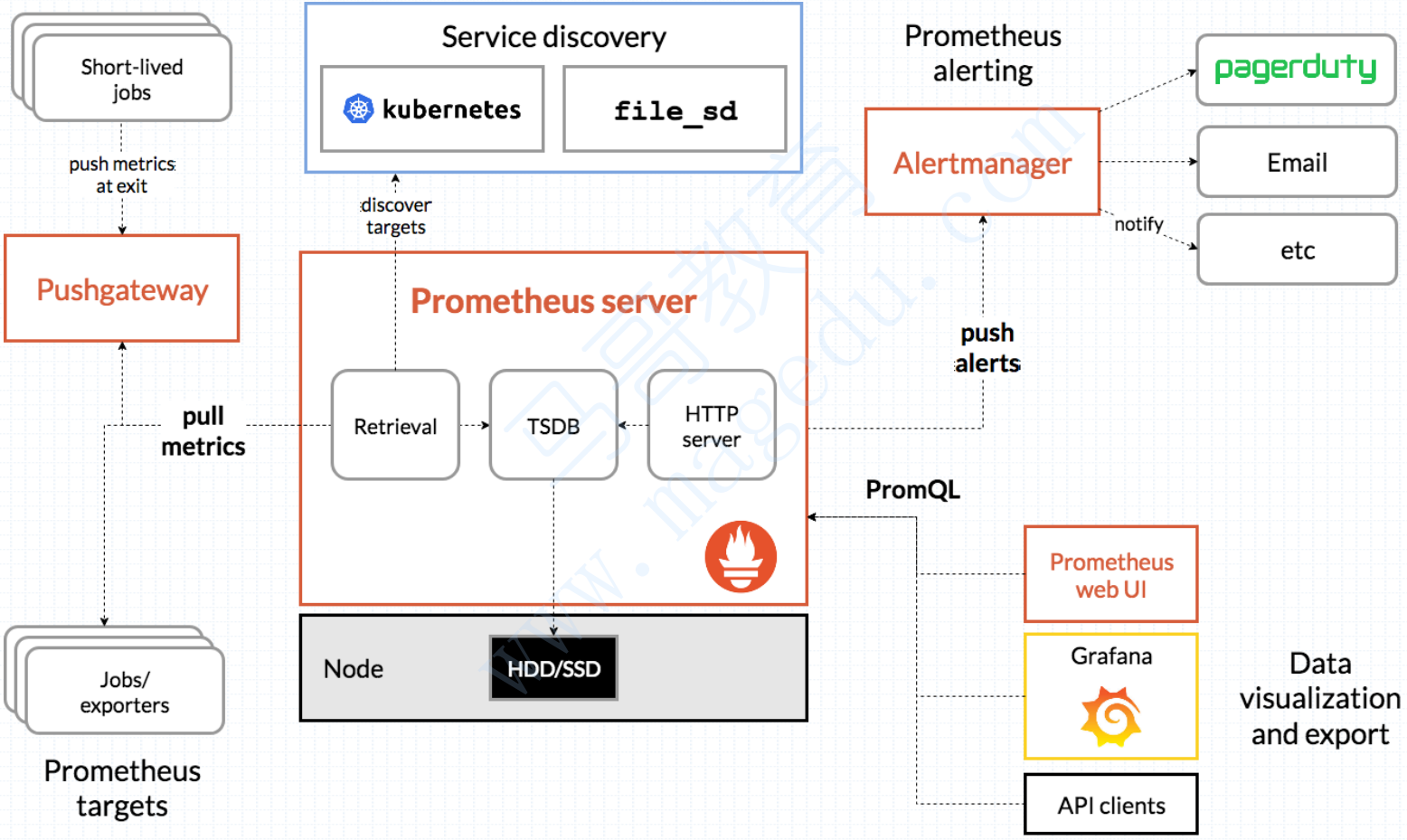
Prometheus架构及组件

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

Prometheus Architecture



Prometheus的关键组件

- 关键组件
  - Prometheus-Server：Prometheus监控系统的核心组件
  - Exporters：指标暴露器
    - node-exporter
    - mysql-exporter
  - AlertManager
  - PushGateway
- 工作模式
  - Prometheus Server基于服务发现（Service Discovery）机制或静态配置获取要监视的目标（Target），并通过每个目标上的指标exporter来采集（Scrape）指标数据
  - Prometheus Server内置了一个基于文件的时间序列存储来持久存储指标数据，用户可使用PromDash或PromQL接口来检索数据，也能够按需将告警需求发往Alertmanager完成告警内容发送
  - 一些短期运行的作业的生命周期过短，难以有效地将必要的指标数据供给到Server端，它们一般会采用推送（Push）方式输出指标数据，Prometheus借助于Pushgateway接收这些推送的数据，进而由Server端进行抓取

Prometheus的局限性

- Prometheus是一款指标监控系统，不适合存储事件及日志等；它更多地展示的是趋势性的监控，而非精准数据；
- Prometheus认为只有最近的监控数据才有查询的需要，其本地存储的设计初衷只是保存短期（例如一个月）数据，因而不支持针对大量的历史数据进行存储；
  - 若需要存储长期的历史数据，建议基于远端存储机制将数据保存于InfluxDB或OpenTSDB等系统中；
- Prometheus的集群机制成熟度不高，即便基于Thanos亦是如此；

部署Prometheus

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

安装Prometheus

- 下载预制的程序包
  - https://prometheus.io/download/

prometheus

The Prometheus monitoring system and time series database. prometheus/prometheus

2.24.1 / 2021-01-20 Release notes

File name	OS	Arch	Size	SHA256 Checksum
prometheus-2.24.1.darwin-amd64.tar.gz	darwin	amd64	62.53 MiB	73c27ba24f5b7beaf78a7bd46a6695966c6ad5f4db02866dc577aaf55b844505
prometheus-2.24.1.dragonfly-amd64.tar.gz	dragonfly	amd64	62.35 MiB	e7a1f6c102d6664b74ddd379f29a48125963628a4b6e5228dd95632da61c2537
prometheus-2.24.1.freebsd-amd64.tar.gz	freebsd	amd64	62.42 MiB	b983b713e15ff9ba6cc7fcccce67d8d73e40182d818fbf7cb40dbef2b8328d72
prometheus-2.24.1.linux-amd64.tar.gz	linux	amd64	62.47 MiB	5aec10296624449e83469ef647cb762bd4de2aa12fc91d2375c5e6be9fd049c0
prometheus-2.24.1.netbsd-amd64.tar.gz	netbsd	amd64	62.35 MiB	73395fa7528a02a54b76361d4f91bb7415b3de70b598d5935eef9afa37254187
prometheus-2.24.1.openbsd-amd64.tar.gz	openbsd	amd64	62.34 MiB	7b02f7accd1caab4ff64fa7f28198e3dfade9309efbbbe4a6fb625e18369db5d
prometheus-2.24.1.windows-amd64.zip	windows	amd64	63.66 MiB	2ef42e0d4051381062de1db3a676f45691bc73b2b3bea8dc82d2dd9ca54c7b21

使用包管理器

- CentOS上可基于yum repository安装Prometheus-Server
  - https://packagecloud.io/app/prometheus-rpm/release/search

```
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/$releasever/$basearch
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
    https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
gpgcheck=1
metadata_expire=300
```

- Ubuntu和Debian可直接使用apt命令安装

Exporters

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

客户端库

- 应用程序自己并不会直接生成指标数据，这依赖于开发人员将相关的客户端库添加至应用程序中构建出的测量系统（instrumentation system）来完成。
  - Prometheus为包括Go、Python、Java（或Scala）和Ruby等主流的编程语言提供了各自适用的客户端库，另外还有适用于Bash、C、C++、C#、Node.js、Haskell、Erlang、Perl、PHP和Rust等多种编程语言的第三方库可用；
  - 通常，三两行代码即能将客户端库整合进应用程序中实现直接测量（direct instrumentation）机制；
- 客户端库主要负责处理所有的细节类问题，例如线程安全和记账，以及生成文本格式的指标以数据响应HTTP请求等；
- 客户端库通常还会额外提供一些指标，例如CPU使用率和垃圾回收统计信息等，具体的实现则取决于库和相关的运行时环境；

Exporter基础

- 对于那些非由用户可直接控制的应用代码来说，为其添加客户端库以进行直接测量很难实现
  - 操作系统内核就是一个典型的示例，它显然不大可能易于实现添加自定义代码通过HTTP协议输出Prometheus格式的指标
  - 但这一类的程序一般都会通常某种接口输出其内在的指标，只不过这些指标可能有着特殊的格式，例如Linux内核的特有指标格式，或者SNMP指标格式等
  - 这些指标需要对它进行适当的解析和处理以转换为合用的目标格式，Exporter（指标暴露器）是完成此类转换功能的应用程序
- Exporter独立运行于要获取其测量指标的应用程序之外，负责接收来自于Prometheus Server的指标获取请求，它通过目标应用程序（真正的目标）内置的指标接口获取指标数据，并将这些指标数据转换为合用的目标格式后响应给Prometheus
  - Exporter更像是“一对一”的代理，它作为Prometheus Server的target存在，工作于应用程序的指标接口和Prometheus的文本指标格式之间转换数据格式
  - 但Exporter不存储也不缓存任何数据

部署Node Export监控系统级指标

- 下载相应的程度包文件

node_exporter

Exporter for machine metrics prometheus/node_exporter

1.0.1 / 2020-06-15 Release notes

File name	OS	Arch	Size	SHA256 Checksum
node_exporter-1.0.1.linux-386.tar.gz	linux	386	8.85 MiB	734e036a849152b185da2080eb8656c36cde862095a464cb17705ca723ea3929
node_exporter-1.0.1.linux-amd64.tar.gz	linux	amd64	9.08 MiB	3369b76cd2b0ba678b6d618deab320e565c3d93ccb5c2a0d5db51a53857768ae
node_exporter-1.0.1.linux-arm64.tar.gz	linux	arm64	8.47 MiB	017514906922fcc4b7d727655690787faed0562bc7a17aa9f72b0651cb1b47fb
node_exporter-1.0.1.linux-armv5.tar.gz	linux	armv5	8.46 MiB	38413100bfb935c59aea088a0af792134b75972eb90ab2bc6cf1c09ad3b08aea
node_exporter-1.0.1.linux-armv6.tar.gz	linux	armv6	8.46 MiB	c1d7affbc7762c478c169830c43b4c6177a761bf1d2dd715dbffa55ca772655a
node_exporter-1.0.1.linux-armv7.tar.gz	linux	armv7	8.45 MiB	e7f4427a25f1870103588e4968c7dc8c1426c00a0c029d0183a9a7afdd61357b
node_exporter-1.0.1.linux-mips.tar.gz	linux	mips	8.45 MiB	43335ccab5728b3c61ea7a0977143719c392ce13a90fa0d14169b5c10e8babd0
node_exporter-1.0.1.linux-mips64.tar.gz	linux	mips64	8.60 MiB	c0109f2f76628d2e25ea78e39d4b95100079ee859863be1471519b5e85a2fe78
node_exporter-1.0.1.linux-mips64le.tar.gz	linux	mips64le	8.35 MiB	bcba02058b9ce171b5c3b077f78f371eb7685239f113200d15787c55fb204857
node_exporter-1.0.1.linux-mipsle.tar.gz	linux	mipsle	8.28 MiB	85f0a24c07c5d8237caf36a5c68a63958280dab802b5056ff36d75563d5e5241
node_exporter-1.0.1.linux-ppc64.tar.gz	linux	ppc64	8.67 MiB	43aa5e72f5068d16eb8d33f6b729186bf558d40ec0c734746b40a16902864808
node_exporter-1.0.1.linux-ppc64le.tar.gz	linux	ppc64le	8.44 MiB	5ae6c772108c877038cd66a761e4ad93edcc8c4461204784994l2b24e7953146
node_exporter-1.0.1.linux-s390x.tar.gz	linux	s390x	9.12 MiB	2f22d1ce18969017fb32dbd285a264adf3da6252eec05f03f105cf638ec0bb06

Exporter Unit文件示例

- Exporter的Unit文件示例
  - 首先添加Prometheus用户
  - 其次，创建node_exporter.service这一Unitfile，文件内容类似如下

```
[Unit]
Description=node_exporter
Documentation=https://prometheus.io/
After=network.target
[Service]
Type=simple
User=prometheus
ExecStart=/usr/local/bin/node_exporter \
        --collector.ntp \
        --collector.mountstats \
        --collector.systemd \
        --collector.tcpstat
ExecReload=/bin/kill -HUP $MAINPID
TimeoutStopSec=20s
Restart=always
[Install]
WantedBy=multi-user.target
```

Node Exporter的指标

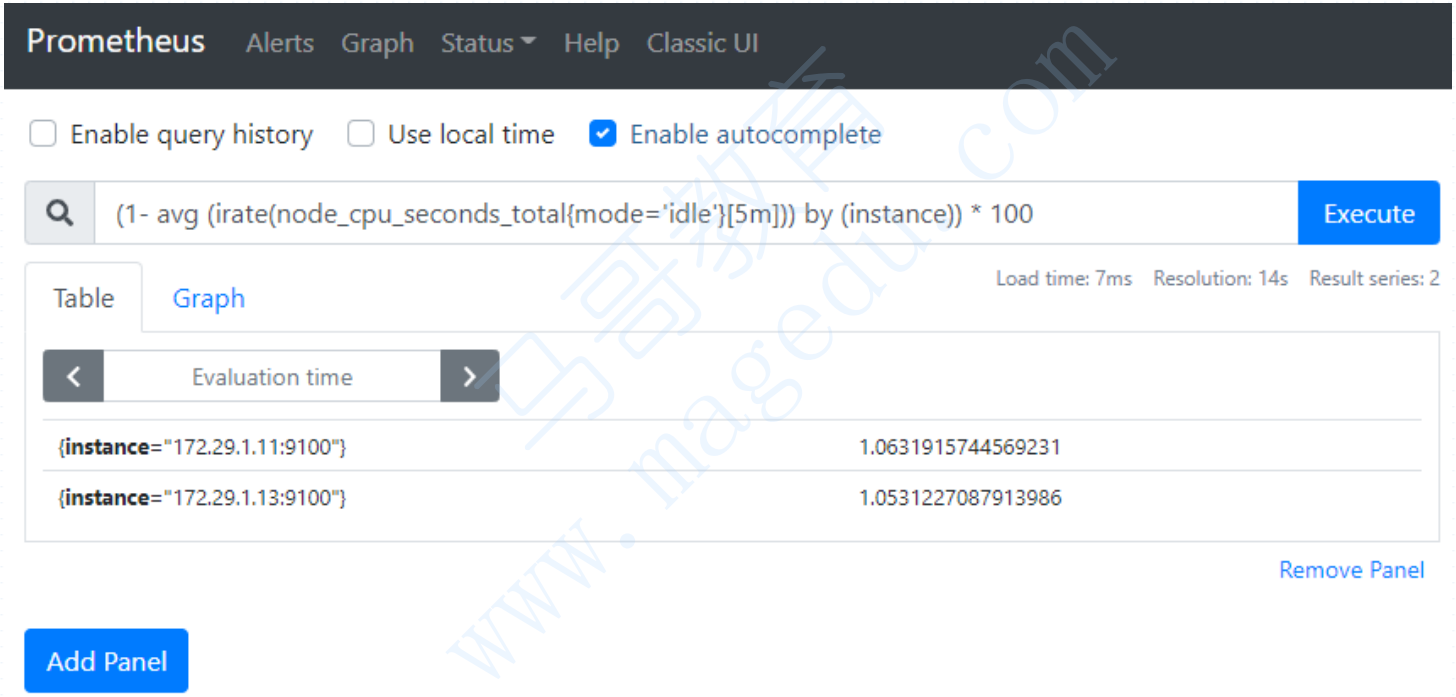
- node-export如何启用内置的指标
  - Collectors are enabled by providing a --collector.<name> flag.
  - Collectors that are enabled by default can be disabled by providing a --no-collector.<name> flag.
- 常用的各指标
  - node_cpu_seconds_total
  - node_memory_MemTotal_bytes
  - node_filesystem_size_bytes{mount_point=PATH}
  - node_system_unit_state{name=}
  - node_vmstat_pswpin：系统每秒从磁盘读到内存的字节数；
  - node_vmstat_pswpout：系统每秒钟从内存写到磁盘的字节数；
  - 更多指标介绍
    - https://github.com/prometheus/node_exporter

适用于主机监控的USE方法

- USE是使用率（Utilization）、饱和度（Saturation）和错误（Error）的缩写，由Netflix的内核和性能工程师Brendan Gregg开发；
  - USE方法可以概括为：针对每个资源，检查使用率、饱和度和错误；
    - 资源：系统的一个组件，在USE中，它指的是一个传统意义上的物理服务器组件，如CPU、内存和磁盘等；
    - 使用率：资源忙于工作的平均时间，它通常用随时间变化的百分比进行表示；
    - 饱和度：资源排队工作的指标，无法再处理额外的工作；通常用队列长度表示；
    - 错误：资源错误事件的计数；
  - 对CPU来说，USE通常意味着如下概念
    - CPU使用率随时间的百分比；
    - CPU饱和度，等待CPU的进程数；
    - 错误，通常对CPU不太有影响；
  - 对内存来说，USE的意义相似
    - 内存使用率随时间的百分比；
    - 内存饱和度，可通过监控swap进行测量；
    - 错误，通常不太关键；

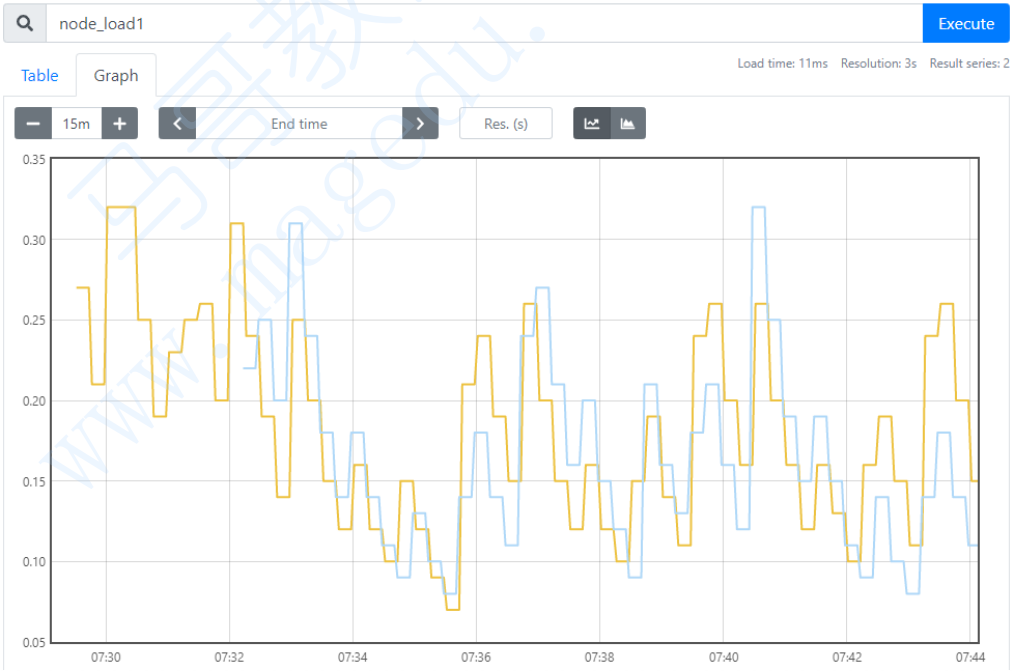
CPU使用率

- 每台主机CPU在5分钟内的平均使用率



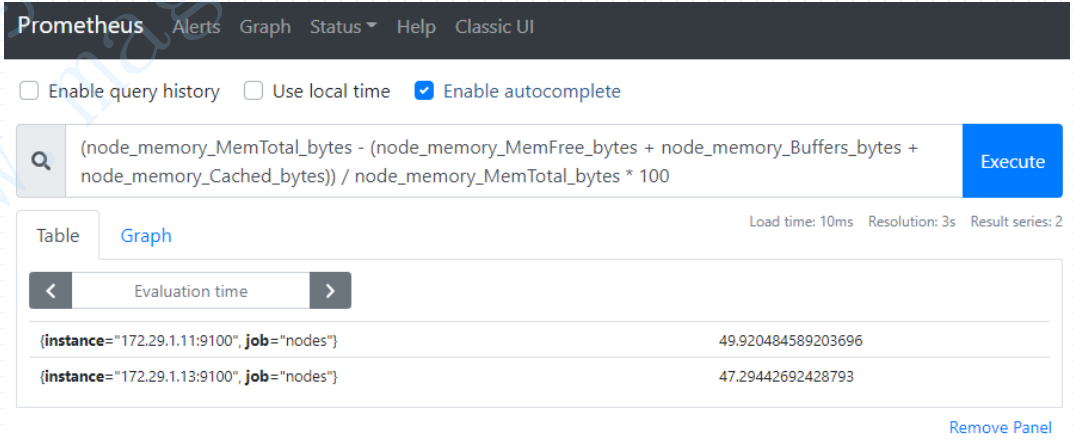
CPU的饱和度

- 跟踪CPU的平均负载就能获取到相关主机的CPU饱和度，实际上，它是将主机上的CPU数量考虑在内的一段时间内的平均运行队列长度
  - 平均负载少于CPU的数量是正常状况，而长时间内超过CPU数量则表示CPU已然饱和；
  - node_load1 > on (instance) 2 * count (node_cpu_seconds_total{mode="idle"}) by (instance)
    - 查询1分钟平均负载超过主机CPU数量两瓠的时间序列



内存使用率

- node_exporter暴露了多个以node_memory为前缀的指标，我们重点关注如下几个
  - node_memory_MemTotal_bytes
  - node_memory_MemFree_bytes
  - node_memory_Buffers_bytes
  - node_memory_Cached_bytes
- 计算使用率
  - 可用空间：上面后三个指标之和；
  - 已用空间：总空间减去可用空间；
  - 使用率：已用空间除以总空间；



MySQL exporter

- Exporter Unit File示例

```
vim /usr/lib/systemd/system/mysqld_exporter.service
[Unit]
Description=mysql_exporter
Documentation=https://prometheus.io/
After=network.target

[Service]
Type=simple
User=prometheus
ExecStart=/usr/local/mysqld_exporter/mysqld_exporter --config.my-cnf=my.cnf
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- my.cnf文件示例

```
[client]
host=127.0.0.1
user=exporter
password=password
```

- 授权exporter用户
  - mysql> GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'localhost';

PromQL基础

讲师：马永亮（马哥）

http://www.magedu.com

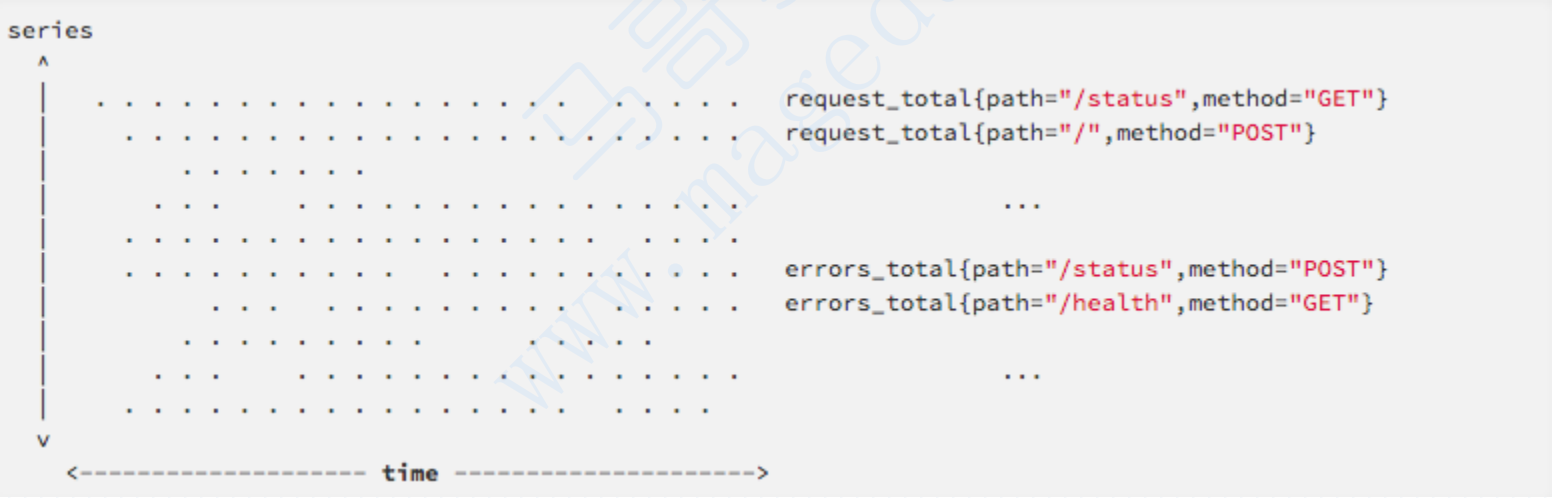
https://github.com/ikubernetes/

本节话题

- PromQL时间序列
  - 数据类型
  - 时间序列
  - 指标
- 时间序列选择器
  - 即时向量选择器
  - 范围向量选择器
- Prometheus指标类型
  - Counter
  - Gauge
  - Histogram
  - Summary

Prometheus时间序列

- 时间序列数据：按照时间顺序记录系统、设备状态变化的数据，每个数据称为一个样本；
  - 数据采集以特定的时间周期进行，因而，随着时间流逝，将这些样本数据记录下来，将生成一个离散的样本数据序列；
  - 该序列也称为向量（Vector）；而将多个序列放在同一个坐标系内（以时间为横轴，以序列为纵轴），将形成一个由数据点组成的矩阵；

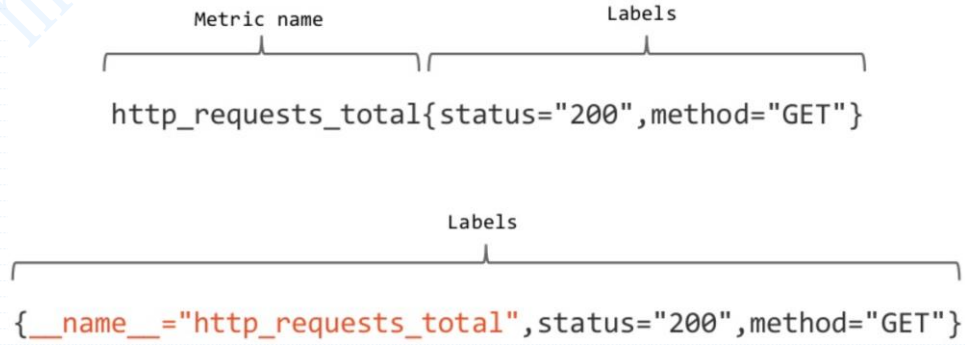


PromQL简介

- Prometheus基于指标名称（metrics name）以及附属的标签集（labelset）唯一定义一条时间序列
  - 指标名称代表着监控目标上某类可测量属性的基本特征标识
  - 标签则是这个基本特征上再次细分的多个可测量维度
- 基于PromQL表达式，用户可以针对指定的特征及其细分的纬度进行过滤、聚合、统计等运算从而产生期望的计算结果
- PromQL (Prometheus Query Language)是Prometheus Server内置数据查询语言
  - PromQL使用表达式（expression）来表述查询需求
  - 根据其使用的指标和标签，以及时间范围，表达式的查询请求可灵活地覆盖在一个或多个时间序列的一定范围内的样本之上，甚至是只包含单个时间序列的单个样本

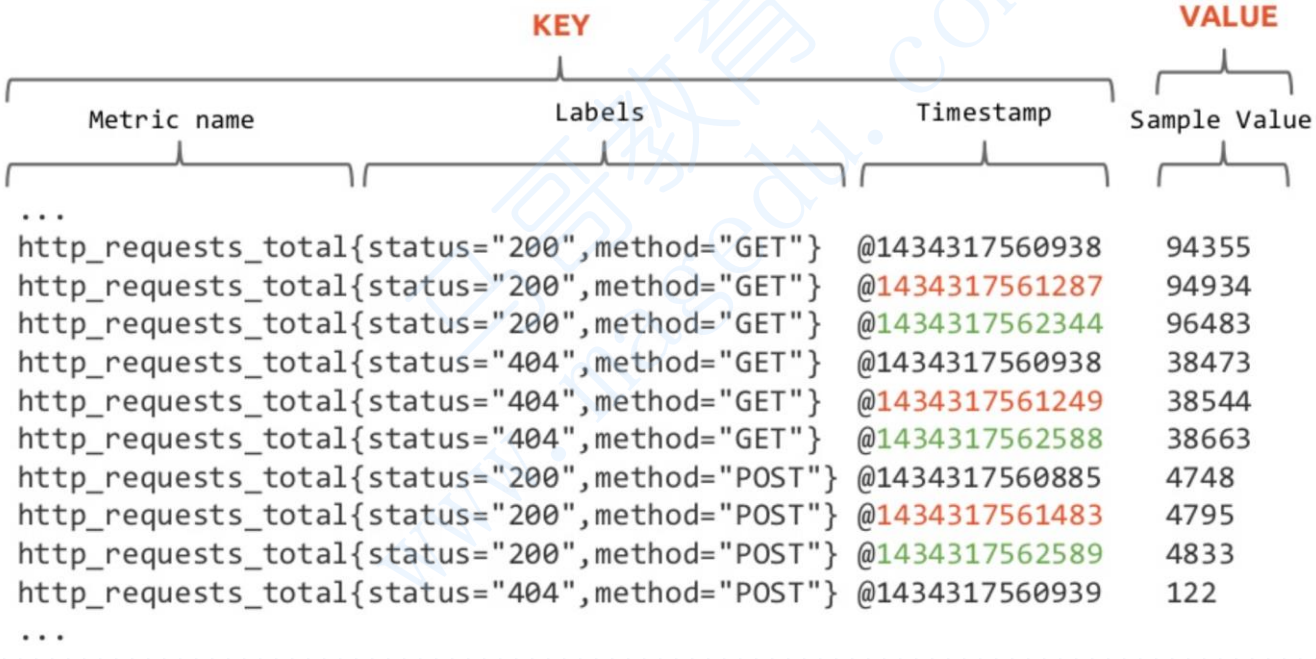
Prometheus数据模型

- Prometheus中，每个时间序列都由指标名称（Metric Name）和标签（Label）来唯一标识，格式为“<metric name>{<label name>=<label value>, …}”；
  - 指标名称：通常用于描述系统上要测定的某个特征；
    - 例如，http_requests_total表示接收到的HTTP请求总数；
    - 支持使用字母、数字、下划线和冒号，且必须能匹配RE2规范的正则表达式；
  - 标签：键值型数据，附加在指标名称之上，从而让指标能够支持多纬度特征；可选项；
    - 例如，http_requests_total{method=GET}和http_requests_total{method=POST}代表着两个不同的时间序列；
    - 标签名称可使用字母、数字和下划线，且必须能匹配RE2规范的正则表达式；
    - 以“__”为前缀的名称为Prometheus系统预留使用；
- Metric Name的表示方式有两种
  - 后一种通常用于Prometheus内部



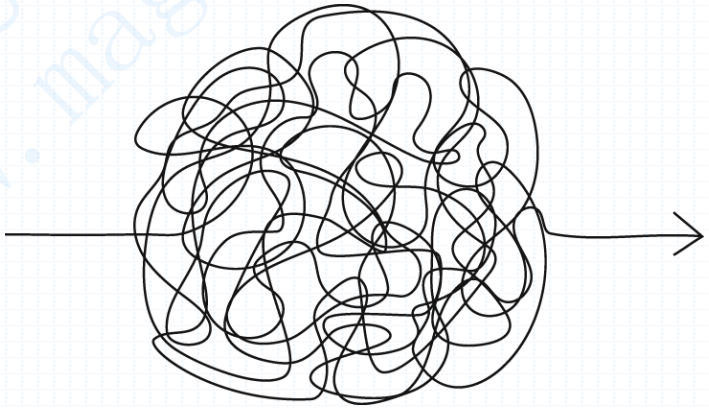
样本数据格式

- Prometheus的每个数据样本由两部分组成
  - float64格式的数据
  - 毫秒精度的时间戳



指标名称及标签使用注意事项

- 指标名称和标签的特定组合代表着一个时间序列；
  - 指标名称相同，但标签不同的组合分别代表着不同的时间序列；
  - 不同的指标名称自然更是代表着不同的时间序列；
- PromQL支持**基于定义的指标维度进行过滤和聚合**
  - 更改任何标签值，包括添加或删除标签，都会创建一个新的时间序列
  - 应该尽可能地保持标签的稳定性，否则，则很可能创建新的时间序列，更甚者会生成一个动态的数据环境，并使得监控的数据源难以跟踪，从而导致建立在该指标之上的图形、告警及记录规则变得无效

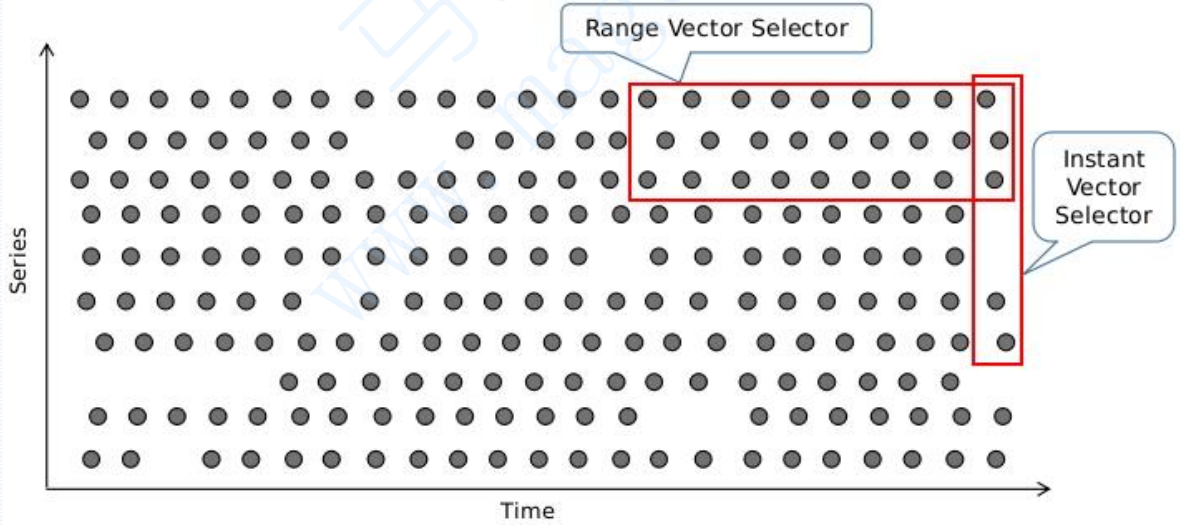


PromQL的数据类型

- PromQL的表达式中支持4种数据类型
  - 即时向量（Instant Vector）：特定或全部的时间序列集合上，具有相同时间戳的一组样本值称为即时向量；
  - 范围向量（Range Vector）：特定或全部的时间序列集合上，在指定的同一时间范围内的所有样本值；
  - 标量（Scalar）：一个浮点型的数据值；
  - 字符串（String）：支持使用单引号、双引号或反引号进行引用，但反引号中不会对转义字符进行转义；

时间序列选择器（Time series Selectors）

- PromQL的查询操需要针对有限个时间序列上的样本数据进行，挑选出目标时间序列是构建表达式时最为关键的一步
  - 用户可使用向量选择器表达式来挑选出给定指标名称下的所有时间序列或部分时间序列的即时（当前）样本值或至过去某个时间范围内的样本值，前者称为即时向量选择器，后者称为范围向量选择器
    - 即时向量选择器（Instant Vector Selectors）：返回0个、1个或多个时间序列上在给定时间戳（instant）上的各自的一个样本，该样本也可称为即时样本；
    - 范围向量选择器（Range Vector Selectors）：返回0个、1个或多个时间序列上在给定时间范围内的各自的一组样本；



向量表达式使用要点

- 表达式的返回值类型亦是即时向量、范围向量、标题或字符串4种数据类型其中之一，但是，有些使用场景要求表达式返回值必须满足特定的条件，例如
  - 需要将返回值绘制成图形时，仅支持即时向量类型的数据；
  - 对于诸如rate一类的速率函数来说，其要求使用的却又必须是范围向量型的数据；
- 由于范围向量选择器的返回的是范围向量型数据，它不能用于表达式浏览器中图形绘制功能，否则，表达式浏览器会返回“Error executing query: invalid expression type "range vector" for range query, must be Scalar or instant Vector”一类的错误
  - 事实上，范围向量选择几乎总是结合速率类的函数rate一同使用

即时向量选择器

- 即时向量选择器由两部分组成；
  - 指标名称：用于限定特定指标下的时间序列，即负责过滤指标；可选；
  - 匹配器（Matcher）：或称为标签选择器，用于过滤时间序列上的标签；定义在{}之中；可选；
- 显然，定义即时向量选择器时，以上两个部分应该至少给出一个；于是，这将存在以下三种组合；
  - 仅给定指标名称，或在标签名称上使用了空值的匹配器：返回给定的指标下的所有时间序列各自的即时样本；
    - 例如，http_requests_total和http_requests_total{}的功能相同，都是用于返回http_requests_total指标下各时间序列的即时样本；
  - 仅给定匹配器：返回所有符合给定的匹配器的所有时间序列上的即时样本；
    - 注意：这些时间序列可能会有着不同的指标名称；
    - 例如，{job=".*", method="get"}
  - 指标名称和匹配器的组合：返回给定的指定下的，且符合给定的标签过滤器的所有时间序列上的即时样本；
    - 例如，http_requests_total{method="get"}

匹配器（Matcher）

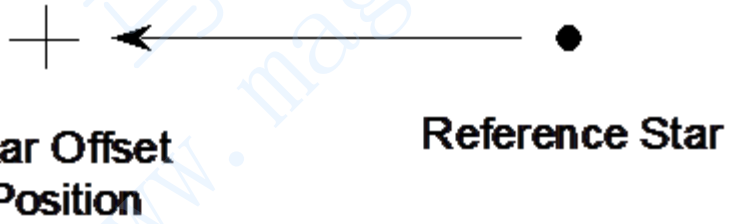
- 匹配器用于定义标签过滤条件，目前支持如下4种匹配操作符；
  - =: Select labels that are exactly equal to the provided string.
  - !=: Select labels that are not equal to the provided string.
  - =~: Select labels that regex-match the provided string.
  - !~: Select labels that do not regex-match the provided string.
- 注意事项
  - 匹配到空标签值的匹配器时，所有未定义该标签的时间序列同样符合条件；
    - 例如，http_requests_total{env= ""}，则该指标名称上所有未使用该标签（env）的时间序列也符合条件，比如时间序列http_requests_total{method ="get"}；
  - 正则表达式将执行完全锚定机制，它需要匹配指定的标签的整个值；
  - 向量选择器至少要包含一个指标名称，或者至少有一个不会匹配到空字符串的匹配器；
    - 例如，{job=""}为非法的选择器；
  - 使用“__name__”做为标签名称，还能够对指标名称进行过滤；
    - 例如，{__name__=~"http_requests_.*"}能够匹配所有以“ http_requests_ ”为前缀的所有指标；

范围向量选择器

- 同即时向量选择器的唯一不同之处在于，范围向量选择器需要在表达式后紧跟一个方括号[]来表达需在时间时序上返回的样本所处的时间范围；
  - 时间范围：以当前时间为基准时间点，指向过去一个特定的时间长度；例如[5m]便是指过去5分钟之内；
- 时间格式：一个整数后紧跟一个时间单位，例如“5m”中的“m”即是时间单位；
  - 可用的时间单位有ms（毫秒）、s（秒）、m（分钟）、h（小时）、d（天）、w（周）和y（年）；
  - 必须使用整数时间，且能够将多个不同级别的单位进行串联组合，以时间单位由大到小为顺序，例如1h30m，但不能使用1.5h；
- 需要注意的是，范围向量选择器返回的是一定时间范围内的数据样本，虽然不同时间序列的数据抓取时间点相同，但它们的时间戳并不会严格对齐；
  - 多个Target上的数据抓取需要分散在抓取时间点前后一定的时间范围内，以均衡Prometheus Server的负载；
  - 因而，Prometheus在趋势上准确，但并非绝对精准；

偏移量修改器

- 默认情况下，即时向量选择器和范围向量选择器都以当前时间为基准时间点，而偏移量修改器能够修改该基准；
- 偏移量修改器的使用方法是紧跟在选择器表达式之后使用“offset”关键字指定
  - “http_requests_total offset 5m”，表示获取以http_requests_total为指标名称的所有时间序列在过去5分钟之时的即时样本；
  - “http_requests_total[5m] offset 1d”，表示获取距此刻1天时间之前的5分钟之内的所有样本；



PromQL的指标类型

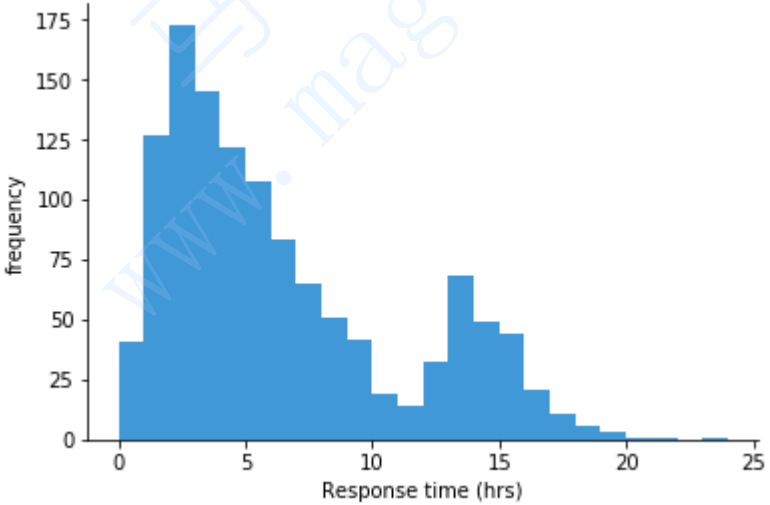
- PromQL有四个指标类型，它们主要由Prometheus的客户端库使用
  - Counter：计数器，单调递增，除非重置（例如服务器或进程重启）；
  - Gauge：仪表盘，可增可减的数据；
  - Histogram：直方图，将时间范围内的数据划分成不同的时间段，并各自评估其样本个数及样本值之和，因而可计算出分位数；
    - 可用于分析因异常值而引起的平均值过大的问题；
    - 分位数计算要使用专用的histogram_quantile函数；
  - Summary：类似于Histogram，但客户端会直接计算并上报分位数；
- Prometheus Server并不使用类型信息，而是将所有数据展平为时间序列

Counter和Gauge

- 通常，Counter的总数并没有直接作用，而是需要借助于rate、topk、increase和irate等函数来生成样本数据的变化状况（增长率）；
  - rate(http_requests_total[2h])，获取2小内，该指标下各时间序列上的http总请求数的增长速率；
  - topk(3, http_requests_total)，获取该指标下http请求总数排名前3的时间序列；
  - irate(http_requests_total[2h])，高灵敏度函数，用于计算指标的瞬时速率；
    - 基于样本范围内的最后两个样本进行计算，相较于rate函数来说，irate更适用于短期时间范围内的变化速率分析；
- Gauge用于存储其值可增可减的指标的样本数据，常用于进行求和、取平均值、最小值、最大值等聚合计算；也会经常结合PromQL的predict_linear和delta函数使用；
  - predict_linear(v range-vector, t, scalar)函数可以预测时间序列v在t秒后的值，它通过线性回归的方式来预测样本数据的Gauge变化趋势；
    -
  - delta(v range-vector)函数计算范围向量中每个时间序列元素的第一个值与最后一个值之差，从而展示不同时间点上的样本值的差值；
    - delta(cpu_temp_celsius{host="web01.magedu.com"}[2h])，返回该服务器上的CPU温度与2小时之前的差异；

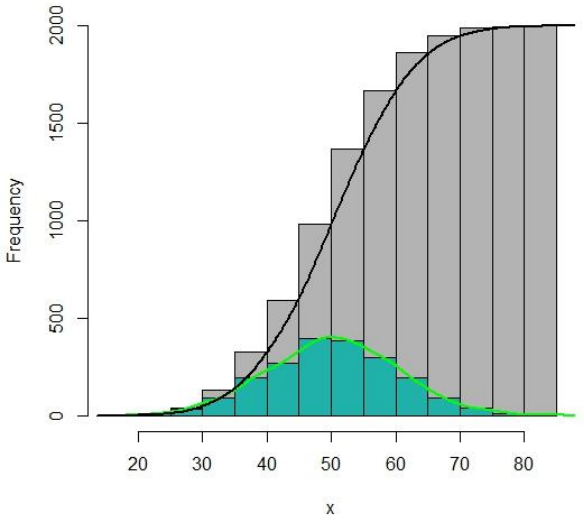
Histogram

- Histogram是一种对数据分布情况的图形表示，由一系列高度不等的长条图（bar）或线段表示，用于**展示单个测度的值的分布**
  - 它一般用横轴表示某个指标维度的数据取值区间，用纵轴表示样本统计的频率或频数，从而能够以二维图的形式展现数值的分布状况
  - 为了构建Histogram，首先需要将值的范围进行分段，即将所有值的整个可用范围分成一系列连续、相邻（相邻处可以是等同值）但不重叠的间隔，而后统计每个间隔中有多少值
  - 从统计学的角度看，分位数不能被聚合，也不能进行算术运算；



Histogram

- 对于Prometheus来说，Histogram会在一段时间范围内对数据进行采样（通常是请求持续时长或响应大小等），并将其计入可配置的bucket（存储桶）中
  - Histogram事先将特定测度可能的取值范围分隔为多个样本空间，并通过对落入bucket内的观测值进行计数以及求和操作
  - 与常规方式略有不同的是，Prometheus取值间隔的划分采用的是累积（Cumulative）区间间隔机制，即每个bucket中的样本均包含了其前面所有bucket中的样本，因而也称为累积直方图
    - 可降低Histogram的维护成本
    - 支持粗略计算样本值的分位数
    - 单独提供了_sum和_count指标，从而支持计算平均值



Histogram

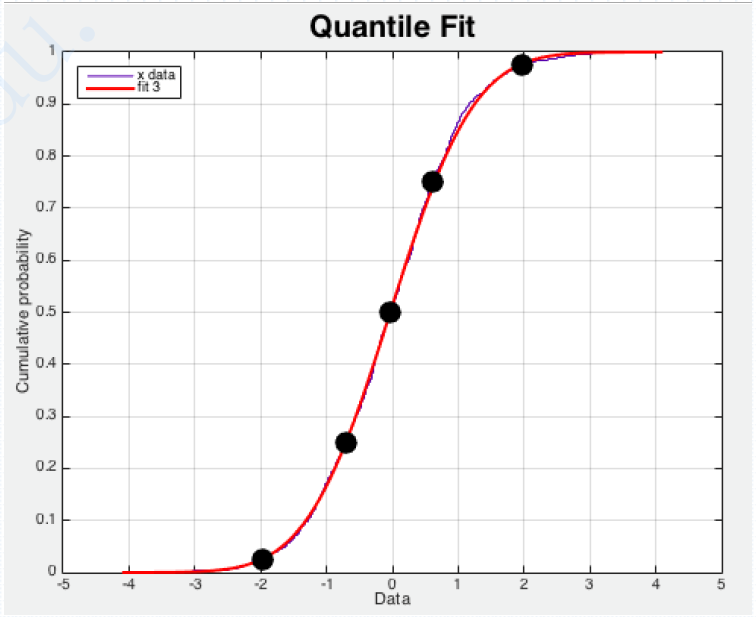
- Histogram类型的每个指标有一个基础指标名称<basename>，它会提供多个时间序列：
  - <basename>_bucket{le="<upper inclusive bound>"}：观测桶的上边界（upper inclusive bound），即样本统计区间，最大区间（包含所有样本）的名称为<basename>_bucket{le="+Inf"}；
  - <basename>_sum：所有样本观测值的总和；
  - <basename>_count：总的观测次数，它自身本质上是一个Counter类型的指标；
- 累积间隔机制生成的样本数据需要额外使用内置的histogram_quantile()函数即可根据Histogram指标来计算相应的分位数（quantile），即某个bucket的样本数在所有样本数中占据的比例
  - histogram_quantile()函数在计算分位数时会假定每个区间内的样本满足线性分布状态，因而它的结果仅是一个预估值，并不完全准确；
  - 预估的准确度取决于bucket区间划分的粒度；粒度越大，准确度越低；

Summary

- 指标类型是客户端库的特性，而Histogram在客户端仅是简单的桶划分和分桶计数，分位数计算由Prometheus Server基于样本数据进行估算，因而其结果未必准确，甚至不合理的bucket划分会导致较大的误差；
- Summary是一种类似于Histogram的指标类型，但它在客户端于一段时间内（默认为10分钟）的每个采样点进行统计，计算并存储了分位数数值，Server端直接抓取相应值即可；
- 但Summary不支持sum或avg一类的聚合运算，而且其分位数由客户端计算并生成，Server端无法获取客户端未定义的分位数，而Histogram可通过PromQL任意定义，有着较好的灵活性；

Summary

- 对于每个指标，Summary以指标名称<basename>为前缀，生成如下几个个指标序列
  - <basename>{quantile="<φ>"}，其中φ是分位点，其取值范围是(0 ≤φ≤ 1)；计数器类型指标；如下是几种典型的常用分位点；
    - 0、0.25、0.5、0.75和1几个分位点；
    - 0.5、0.9和0.99几个分位点；
    - 0.01、0.05、0.5、0.9和0.99几个分位点；
  - <basename>_sum，抓取到的所有样本值之和；
  - <basename>_count，抓取到的所有样本总数；



PromQL进阶

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

Agenda

- 聚合运算
  - 聚合运算表达式
  - 聚合函数
- 二元运算
  - 算术运算
  - 比较运算
  - 集合/逻辑运算
- 向量匹配
  - 一对一匹配
  - 一对多/多对一匹配
- HTTP API
-

Prometheus的聚合函数

- 一般说来，单个指标的价值不大，监控场景中往往需要联合并可视化一组指标，这种联合机制即是指“聚合”操作，例如，将计数、求和、平均值、分位数、标准差及方差等统计函数应用于时间序列的样本之上生成具有统计学意义的结果等；
- 对查询结果事先按照某种分类机制进行分组（groupby）并将查询结果按组进行聚合计算也是较为常见的需求，例如分组统计、分组求平均值、分组求和等；
- 聚合操作由**聚合函数**针对一组值进行计算并返回单个值或少量几值作为结果
  - Prometheus内置提供的11个聚合函数也称为聚合运算符
  - 这些运算符**仅支持**应用于**单个即时向量**的元素，其返回值也是具有少量元素的新向量或标量
  - 这些聚合运行符既可以基于向量表达式返回结果中的时间序列的所有标签维度进行分组聚合，也可以仅基于指定的标签维度分组后再进行分组聚合

聚合表达式

- PromQL中的聚合操作语法格式可采用如下面两种格式之一
  - <aggr-op>([parameter,] <vector expression>) [without|by (<label list>)]
  - <aggr-op> [without|by (<label list>)] ([parameter,] <vector expression>)
- 分组聚合：先分组、后聚合
  - without：从结果向量中删除由without子句指定的标签，未指定的那部分标签则用作分组标准；
  - by：功能与without刚好相反，它仅使用by子句中指定的标签进行聚合，结果向量中出现但未被by子句指定的标签则会被忽略；
    - 为了保留上下文信息，使用by子句时需要显式指定其结果中原本出现的job、instance等一类的标签
- 事实上，各函数工作机制的不同之处也仅在于计算操作本身，PromQL对于它们的执行逻辑相似；

11个聚合函数

- sum()：对样本值求和；
- avg()：对样本值求平均值，这是进行指标数据分析的标准方法；
- count()：对分组内的时间序列进行数量统计；
- stddev()：对样本值求标准差，以帮助用户了解数据的波动大小（或称之为波动程度）；
- stdvar()：对样本值求方差，它是求取标准差过程中的中间状态；
- min()：求取样本值中的最小者；
- max()：求取样本值中的最大者；
- topk()：逆序返回分组内的样本值最大的前k个时间序列及其值；
- bottomk()：顺序返回分组内的样本值最小的前k个时间序列及其值；
- quantile()：分位数用于评估数据的分布状态，该函数会返回分组内指定的分位数的值，即数值落在小于等于指定的分位区间的比例；
- count_values()：对分组内的时间序列的样本值进行数量统计；

二元运算符（Binary Operators）

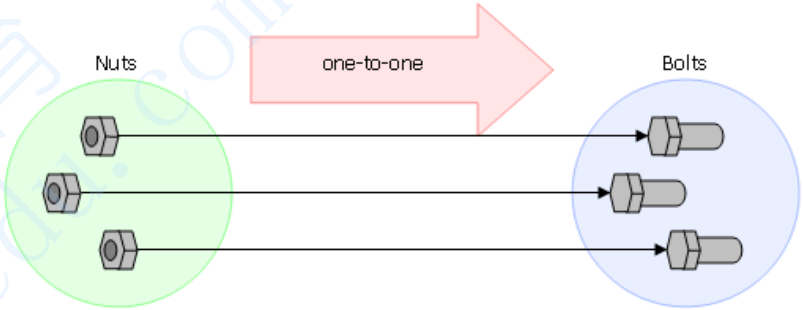
- PromQL支持基本的算术运算和逻辑运算，这类运算支持使用操作符连接两个操作数，因而也称为二元运算符或二元操作符；
  - 支持的运算
    - 两个标量间运算；
    - 即时向量和标量间的运算：将运算符应用于向量上的每个样本；
    - 两个即时向量间的运算：遵循向量匹配机制；
  - 将运算符用于两个即时向量间的运算时，可基于**向量匹配**模式（Vector Matching）定义其运算机制；
- 算术运算
  - 支持的运算符：+（加）、-（减）、*（乘）、/（除）、%（取模）和^（幂运算）；
- 比较运算
  - 支持的运算符：==（等值比较）、!=（不等）、>、<、>=和<=（小于等于）；
- 逻辑/集合运算
  - 支持的运算符：and（并且）、or（或者）和unless（除了）；
  - 目前，该运算仅允许在两个即时向量间进行，尚不支持标量参与运算；

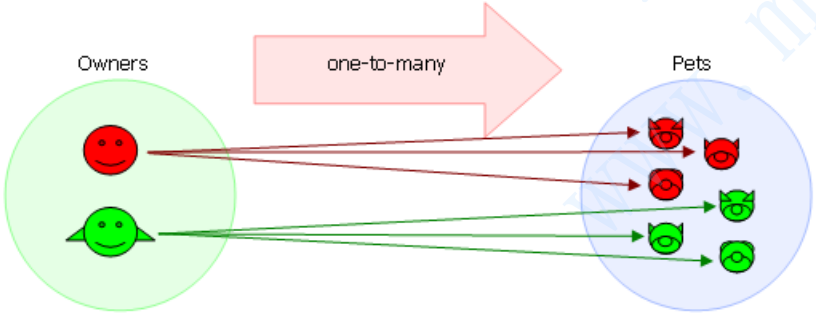
二元运算符优先级

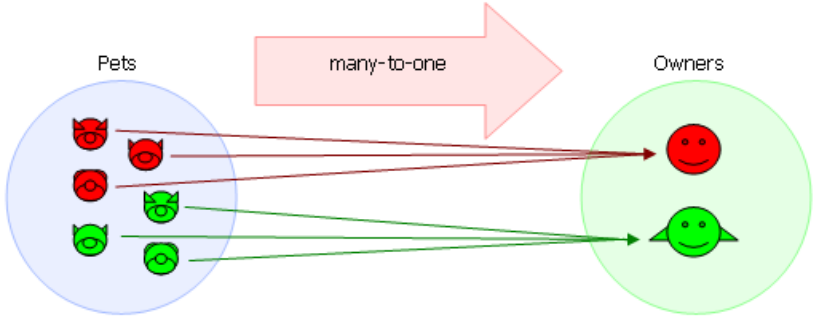
- Prometheus的复杂运算中，二元运算符存在如下给定次序中所示的由高到低的优先级
  - ^
  - *, /, %
  - +, -
  - ==, !=, <=, <, >=, >
  - and, unless
  - or
- 具有相同优先级的运算符满足结合律（左结合），但幂运算除外，因为它是右结合机制；
- 可以使用括号()改变运算次序；

向量匹配

- 即时向量间的运算是PromQL的特色之一；运算时，PromQL为会左侧向量中的每个元素找到匹配的元素，其匹配行为有两种基本类型
  - 一对一 （One-to-One）
  - 一对多或多对一 （Many-to-One, One-to-Many）







向量一对一匹配

- 即时向量的一对一匹配
  - 从运算符的两边表达式所获取的即时向量间依次比较，并找到唯一匹配（标签完全一致）的样本值；
  - 找不到匹配项的值则不会出现在结果中；
- 匹配表达式语法
  - <vector expr> <bin-op> ignoring(<label list>) <vector expr>
  - <vector expr> <bin-op> on(<label list>) <vector expr>
    - ignore：定义匹配检测时要忽略的标签；
    - on：定义匹配检测时只使用的标签；
  - 例如，rate(http_requests_total{status_code=~"5.*"}[5m]) > .1 * rate(http_requests_total[5m])
    - 左侧会生成一个即时向量，它计算出5xx响应码的各类请求的增长速率；
      - ✓ 除了status_code标签外，该指标通常还有其它标签；于是，status_code的值为500的标签同其它标签的每个组合将代表一个时间序列，其相应的即时样本即为结果向量的一个元素；
    - 右侧会生成一个即时向量，它计算出所有标签组合所代表的各类请求的增长速率；
    - 计算时，PromQL会在操作符左右两侧的结果元素中找到标签完全一致的元素进行比较；
    - 其意义为，计算出每类请求中的500响应码在该类请求中所占的比例；

一对一匹配示例

Example input:

```
method_code:http_errors:rate5m{method="get", code="500"}  24
method_code:http_errors:rate5m{method="get", code="404"}  30
method_code:http_errors:rate5m{method="put", code="501"}  3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get"}  600
method:http_requests:rate5m{method="del"}  34
method:http_requests:rate5m{method="post"} 120
```

Example query:

```
method_code:http_errors:rate5m{code="500"} / ignoring(code) method:http_requests:rate5m
```

This returns a result vector containing the fraction of HTTP requests with status code of 500 for each method, as measured over the last 5 minutes. Without `ignoring(code)` there would have been no match as the metrics do not share the same set of labels. The entries with methods `put` and `del` have no match and will not show up in the result:

```
{method="get"}  0.04            //  24 / 600
{method="post"} 0.05            //   6 / 120
```

向量一对多/多对一匹配

- 一对多/多对一匹配
  - “一”侧的每个元素，可与“多”侧的多个元素进行匹配；
  - 必须使用group_left或group_right明确指定哪侧为“多”侧；
- 匹配表达式语法
  - <vector expr> <bin-op> ignoring(<label list>) group_left(<label list>) <vector expr>
  - <vector expr> <bin-op> ignoring(<label list>) group_right(<label list>) <vector expr>
  - <vector expr> <bin-op> on(<label list>) group_left(<label list>) <vector expr>
  - <vector expr> <bin-op> on(<label list>) group_right(<label list>) <vector expr>

多对一匹配示例

Example query:

```
method_code:http_errors:rate5m / ignoring(code) group_left method:http_requests:rate5m
```

In this case the left vector contains more than one entry per `method` label value. Thus, we indicate this using `group_left`. The elements from the right side are now matched with multiple elements with the same `method` label on the left:

```
{method="get", code="500"}  0.04            //  24 / 600
{method="get", code="404"}  0.05            //  30 / 600
{method="post", code="500"} 0.05            //   6 / 120
{method="post", code="404"} 0.175           //  21 / 120
```

Many-to-one and one-to-many matching are advanced use cases that should be carefully considered. Often a proper use of `ignoring(<labels>)` provides the desired outcome.

Service Discovery

讲师：马永亮（马哥）

http://www.magedu.com

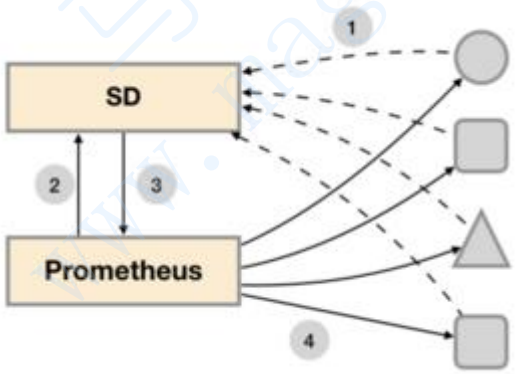
https://github.com/ikubernetes/

Agenda

- Prometheus指标抓取的生命周期
  - 发现 -> 配置 -> relabel -> 指标数据抓取 -> metrics relabel
- Prometheus的服务发现
  - 基于文件的服务发现；
  - 基于DNS的服务发现；
  - 基于API的服务发现：Kubernetes、Consul、Azure、……
- 重新标记
  - target重新打标
  - metric重新打标

Prometheus为何要进行服务发现？

- Prometheus Server的数据抓取工作于Pull模型，因而，它必需要事先知道各Target的位置，然后才能从相应的Exporter或Instrumentation中抓取数据
  - 对于小型的系统环境来说，通过static_configs指定各Target便能解决问题，这也是最简单的配置方法；
    - 每个Targets用一个网络端点（ip:port）进行标识；
  - 对于中大型的系统环境或具有较强动态性的云计算环境来说，静态配置显然难以适用；
  - 因此，Prometheus为此专门设计了一组服务发现机制，以便于能够基于服务注册中心（服务总线）自动发现、检测、分类可被监控的各Target，以及更新发生了变动的Target；



指标抓取的生命周期

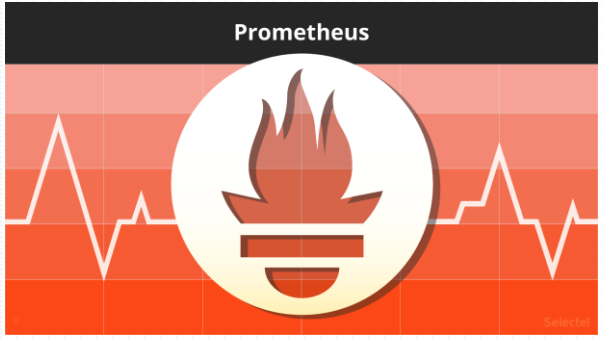
- 下图展示了Prometheus上进行指标抓取的简单生命周期
  - 在每个scrape_interval期间，Prometheus都会检查执行的作业（Job）；
  - 这些作业首先会根据Job上指定的发现配置生成target列表，此即服务发现过程；
    - 服务发现会返回一个Target列表，其中包含一组称为元数据的标签，这些标签都以“__meta_”为前缀；
    - 服务发现还会根据目标配置来设置其它标签，这些标签带有“__”前缀和后缀，包括“__scheme__”、“__address__”和“__metrics_path__”，分别保存有target支持使用协议(http或https，默认为http）、target的地址及指标的URI路径（默认为/metrics）；
    - 若URI路径中存在任何参数，则它们的前缀会设置为“__param_”
    - 这些目标列表和标签会返回给Prometheus，其中的一些标签也可以配置中被覆盖；
  - 配置标签会在抓取的生命周期中被重复利用以生成其他标签，例如，指标上的instance标签的默认值就来自于__address__标签的值；

指标抓取的生命周期

- 对于发现的各目标，Prometheus提供了可以重新标记(relabel) 目标的机会
  - 它定义在job配置段的relabel_config配置中，常用于实现如下功能
    - 将来自服务发现的元数据标签中的信息附加到指标的标签上；
    - 过滤目标；
- 这之后，便是数据抓取、以及指标返回的过程；
- 抓取而来的指标在保存之前，还允许用户对指标重新打标并过滤的方式；
  - 它定义在job配置段的metric_relabel_configs配置中，常用于实现如下功能
    - 删除不必要的指标；
    - 从指标中删除敏感或不需要的标签；
    - 添加、编辑或修改指标的标签值或标签格式；

Prometheus可集成的服务发现机制

- 不同场景中，服务注册中心的指代也会有所不同
  - 公有或私有IaaS云自身保存有平台上的所有资源信息，其API Server便可作为Prometheus的服务发现媒介；
    - azure、ec2、digitalocean、gce、hetzner、
  - Prometheus也可以集成到多种不同的开源服务发现工具上，以动态发现需要监控的目标；
    - Consul、Eureka Zookeeper Serverset或Airbnb Nerve等
  - Prometheus也可以很好地集成到Kubernetes平台上，通过其API Server动态发现各类被监控的Pod（容器集）、Service、Endpoint、Ingress和Node对象；
    - 它也支持基于dockerswarm和marathon两款编排工具进行服务发现；
  - Prometheus还支持基于DNS或文件的动态发现机制；



基于文件的服务发现

- 基于文件的服务发现是仅仅略优于静态配置的服务发现方式，它不依赖于任何平台或第三方服务，因而也是最为简单和通用的实现方式；
  - Prometheus Server定期从文件中加载Target信息
    - 文件可使用JSON和YAML格式，它含有定义的Target列表，以及可选的标签信息；
    - 下面第一个配置，能够将prometheus默认的静态配置转换为基于文件的服务发现时所需的配置；

```
- targets:
  - localhost:9090
  labels:
    app: prometheus
    job:  prometheus
```

```
- targets:
  - localhost:9100
  labels:
    app: node-exporter
    job: node
```

  - 这些文件可由另一个系统生成，例如Puppet、Ansible或Saltstack等配置管理系统，也可能是由脚本基于CMDB定期查询生成；

基于文件的服务发现

- 发现target的配置，定义在配置文件的job之中

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
  - static_configs:
    - targets:

rule_files:

scrape_configs:
  - job_name: ‘prometheus'
    file_sd_configs:
    - files:                              # 指定要加载的文件列表；
      - targets/prometheus*.yaml          # 文件加载支持glob通配符；
      refresh_interval: 2m                # 每隔2分钟重新加载一次文件中定义的Targets，默认为5m；

  - job_name: ‘nodes'
    file_sd_configs:
    - files:
      - targets/node*.yaml
      refresh_interval: 2m
```

基于DNS的服务发现

- 基于DNS的服务发现针对一组DNS域名进行定期查询，以发现待监控的目标
  - 查询时使用的DNS服务器由/etc/resolv.conf文件指定；
  - 该发现机制依赖于A、AAAA和SRV资源记录，且仅支持该类方法，尚不支持RFC6763中的高级DNS发现方式；

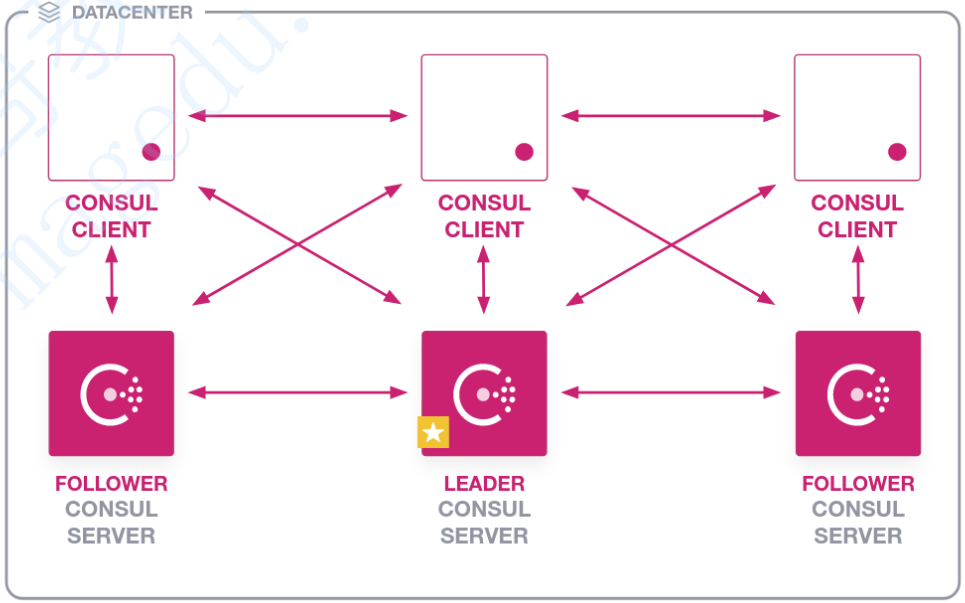
```
# A list of DNS domain names to be queried.
names:
  [ - <string> ]

# The type of DNS query to perform. One of SRV, A, or AAAA.
[ type: <string> | default = 'SRV' ]

# The port number used if the query type is not SRV.
[ port: <int>]

# The time after which the provided names are refreshed.
[ refresh_interval: <duration> | default = 30s ]
```

此处要指定SRV资源记录的名称，例如“_prometheus._tcp.magedu.com”

  - 元数据标签：
    - __meta_dns_name
    - __meta_dns_srv_record_target
    - __meta_dns_srv_record_port

Consul基础

- Consul简介
  - 一款基于golang开发的开源工具，主要面向分布式，服务化的系统提供服务注册、服务发现和配置管理的功能
  - 提供服务注册/发现、健康检查、Key/Value存储、多数据中心和分布式一致性保证等功能
- 部署（以开发模式为例）
  - 下载地址
    - https://www.consul.io/downloads/
  - 安装
    - unzip consul_1.9.2_linux_amd64.zip -d /usr/local/bin/
  - 启动开发者模式
    - mkdir -pv /consul/data/
    - consul agent -dev -ui -data-dir=/consul/data/ \ -config-dir=/etc/consul/ -client=0.0.0.0
  - 正常应该使用server模式



Consul基础（2）

启动Consul时常用的关键参数

- `server` - Providing this flag specifies that you want the agent to start in server mode.
- `-bootstrap-expect` - This tells the Consul server how many servers the datacenter should have in total. All the servers will wait for this number to join before bootstrapping the replicated log, which keeps data consistent across all the servers. Because you are setting up a one-server datacenter, you'll set this value to `1`.
- `-node` - Each node in a datacenter must have a unique name. By default, Consul uses the hostname of the machine, but you'll manually override it, and set it to `agent-one`.
- `-bind` - This is the address that this agent will listen on for communication from other Consul members. It must be accessible by all other nodes in the datacenter. If you don't set a bind address Consul will try to listen on all IPv4 interfaces and will fail to start if it finds multiple private IPs. Since production servers often have multiple interfaces, you should always provide a bind address. In this case it is `172.20.20.10`, which you specified as the address of the first VM in your Vagrantfile.
- `data-dir` - This flag tells Consul agents where they should store their state, which can include sensitive data like ACL tokens for both servers and clients. In production deployments you should be careful about the permissions for this directory.
- `config-dir` - This flag tells consul where to look for its configuration. You will set it to a standard location: `/etc/consul.d`.

Consul基础(3)

- 查看集群成员
  - 使用consul members命令

```
~]# consul members
Node                  Address         Status  Type    Build  Protocol  DC   Segment
master01.magedu.com   172.29.1.1:8301  alive   server  1.9.2  2         dc1  <all>
```

  - 通过HTTP API进行
    - ~]# curl localhost:8500/v1/catalog/nodes
- 在Server上添加其它agent
  - consul join [options] address ...
- 在agent主机上，设置该agent离开集群并关闭agent
  - consul leave

Consul基础(4)

- 在Consul上注册Service
  - 在配置目录中添加文件，在文件中定义单个Service，或者在文件中定义多个Services

```
{
 "service": {
  "id": "node_exporter",
  "name": "node01",
  "address": "172.29.1.11",
  "port": 9100,
  "tags": ["nodes"],
  "checks": [{
   "http": "http://172.29.1.11:9100/metrics",
   "interval": "5s"
  }]
 }
}
```

```
{
 "services": [{
  "id": "node_exporter-node01",
  "name": "node01",
  "address": "172.29.1.11",
  "port": 9100,
  "tags": ["nodes"],
  "checks": [{
   "http": "http://172.29.1.11:9100/metrics",
   "interval": "5s"
  }]
 },
 {
  "ID": "node_exporter-node02",
  "Name": "node02",
  "Address": "172.29.1.12",
  "Port": 9100,
  "Tags": ["nodes"],
  "Checks": [{
   "http": "http://172.29.1.12:9100/metrics",
   "interval": "5s"
  }]
 }]
}
```

而后，运行consul reload命令，让consul重新加载配置信息；
或者，直接基于HTTP API提交注册的服务信息即可；

~# curl -X PUT --data @nodes.json http://localhost:8500/v1/agent/service/register

注销Service，要使用consul service deregister命令，或者通过deregister API完成；

基于Consul的服务发现

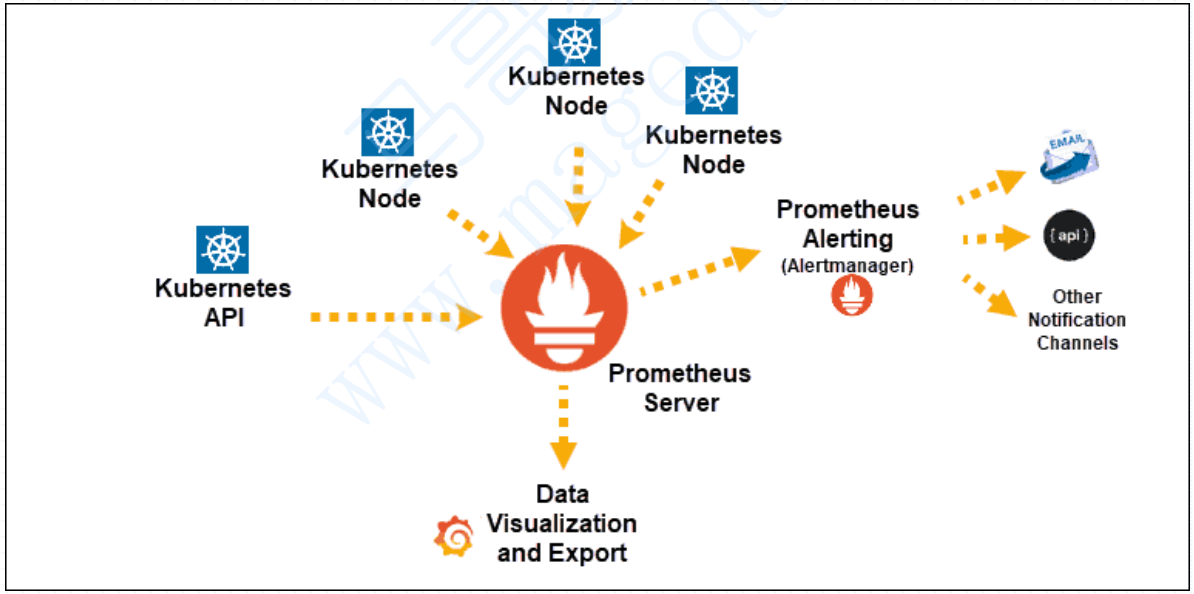
- Prometheus的Consul服务发现机制将通过Consul的Catalog API来发现target；
  - 在发现的target上执行relabel时，它支持使用如下meta标签
    - __meta_consul_address: the address of the target
    - __meta_consul_dc: the datacenter name for the target
    - __meta_consul_health: the health status of the service
    - __meta_consul_metadata_<key>: each node metadata key value of the target
    - __meta_consul_node: the node name defined for the target
    - __meta_consul_service_address: the service address of the target
    - __meta_consul_service_id: the service ID of the target
    - __meta_consul_service_metadata_<key>: each service metadata key value of the target
    - __meta_consul_service_port: the service port of the target
    - __meta_consul_service: the name of the service the target belongs to
    - __meta_consul_tagged_address_<key>: each node tagged address key value of the target
    - __meta_consul_tags: the list of tags of the target joined by the tag separator

consul_sd_configs配置段的可用参数


```
# The information to access the Consul API. It is to be defined as the Consul documentation requires.
[ server: <host> | default = "localhost:8500" ]
[ token: <secret> ]
[ datacenter: <string> ]
[ scheme: <string> | default = "http" ]
[ username: <string> ]
[ password: <secret> ]

tls_config:
  [ <tls_config> ]

# A list of services for which targets are retrieved. If omitted, all services  are scraped.
services:
  [ - <string> ]

# An optional list of tags used to filter nodes for a given service. Services must contain all tags in the list.
tags:
  [ - <string> ]

# Node metadata key/value pairs to filter nodes for a given service.
[ node_meta:
  [ <string>: <string> ... ] ]

# The string by which Consul tags are joined into the tag label.
[ tag_separator: <string> | default = , ]

# Allow stale Consul results (see https://www.consul.io/api/features/consistency.html). Will reduce load on Consul.
[ allow_stale: <boolean> | default = true ]

# The time after which the provided names are refreshed. On large setup it might be a good idea to increase this value because the catalog will change all the time.
[ refresh_interval: <duration> | default = 30s ]
```

基于Kubernetes API的服务发现

- 基于Kubernetes API的服务发现机制，支持将API Server中Node、Service、Endpoint、Pod和Ingress等资源类型下相应的各资源对象视作target，并持续监视相关资源的变动；
  - Node、Service、Endpoint、Pod和Ingress资源分别由各自的发现机制进行定义
  - 负责发现每种类型资源对象的组件，在Prometheus中称为一个“role”
  - 同时支持在集群上基于DaemonSet控制器部署node-exporter后发现各节点



Node资源发现

- Prometheus的node role将Kubernetes集群中的每个节点视作一个target，这些节点都监听着kubelet使用的端口；
  - node role依次检索节点规范上的NodeInternalIP、NodeInternalIP、NodeExternalIP、NodeLegacyHostIP和NodeHostName字段，并将发现的第一个地址作为目标地址（__address__）；
  - 可用的meta标签有如下几个：
    - __meta_kubernetes_node_name: The name of the node object.
    - __meta_kubernetes_node_label_<labelname>: Each label from the node object.
    - __meta_kubernetes_node_labelpresent_<labelname>: true for each label from the node object.
    - __meta_kubernetes_node_annotation_<annotationname>: Each annotation from the node object.
    - __meta_kubernetes_node_annotationpresent_<annotationname>: true for each annotation from the node object.
    - __meta_kubernetes_node_address_<address_type>: The first address for each node address type, if it exists.
  - 节点上instance标签的值取自从API Server中发现的节点的名称；

Pod资源发现

- Prometheus的pod role负责发现Kubernetes集群上的每个Pod资源并暴露其容器为target；
  - 把Pod上声明的每个端口视作一个target；
  - 会为未指定端口的容器创建“无端口”类型的target，以便于用户通过relabel机制手动添加端口
  - 可用的部分metadata标签如下
    - __meta_kubernetes_namespace: The namespace of the pod object.
    - __meta_kubernetes_pod_name: The name of the pod object.
    - __meta_kubernetes_pod_ip: The pod IP of the pod object.
    - __meta_kubernetes_pod_label_<labelname>: Each label from the pod object.
    - __meta_kubernetes_pod_annotation_<annotationname>: Each annotation from the pod object.
    - __meta_kubernetes_pod_ready: Set to true or false for the pod's ready state.
    - __meta_kubernetes_pod_phase: Set to Pending, Running, Succeeded, Failed or Unknown in the lifecycle.
    - __meta_kubernetes_pod_node_name: The name of the node the pod is scheduled onto.
    - __meta_kubernetes_pod_host_ip: The current host IP of the pod object.
    - __meta_kubernetes_pod_uid: The UID of the pod object.
    - ……

Service资源发现

- Prometheus的service role负责发现Kubernetes集群上的每个Service资源；
  - 把Service上声明的每个端口视作一个target；
  - 特别适用于对Service进行黑盒监控的场景；
  - target地址为Service的DNS名称及相应的端口；
  - 可用的部分meta标签如下
    - __meta_kubernetes_namespace: The namespace of the service object.
    - __meta_kubernetes_service_cluster_ip: The cluster IP address of the service. (Does not apply to services of type ExternalName)
    - __meta_kubernetes_service_external_name: The DNS name of the service. (Applies to services of type ExternalName)
    - __meta_kubernetes_service_label_<labelname>: Each label from the service object.
    - __meta_kubernetes_service_annotation_<annotationname>: Each annotation from the service object.
    - __meta_kubernetes_service_name: The name of the service object.
    - __meta_kubernetes_service_port_name: Name of the service port for the target.
    - __meta_kubernetes_service_port_protocol: Protocol of the service port for the target.
    - __meta_kubernetes_service_type: The type of the service.

Endpoint资源发现

- Prometheus的endpoint role从各Endpoint资源中发现目标；
  - 它把endpoint上的每个端口都视作一个单独的target；
  - 若endpoint的后端工作负载是Pod，则会把该Pod上其它未绑定到endpoint的端口同样视作一个单独的目标；
  - 可用的部分meta标签如下
    - __meta_kubernetes_namespace: The namespace of the endpoints object.
    - __meta_kubernetes_endpoints_name: The names of the endpoints object.
    - 对于通过附加在Endpoint资源上的端口发现的各target，还以如下meta标签
      - ✓ __meta_kubernetes_endpoint_hostname: Hostname of the endpoint.
      - ✓ __meta_kubernetes_endpoint_node_name: Name of the node hosting the endpoint.
      - ✓ __meta_kubernetes_endpoint_ready: Set to true or false for the endpoint's ready state.
      - ✓ __meta_kubernetes_endpoint_port_name: Name of the endpoint port.
      - ✓ __meta_kubernetes_endpoint_port_protocol: Protocol of the endpoint port.
      - ✓ __meta_kubernetes_endpoint_address_target_kind: Kind of the endpoint address target.
      - ✓ __meta_kubernetes_endpoint_address_target_name: Name of the endpoint address target.
    - 若发现的Endpoint资源属于某个Service，则相关Service的元标签也会添加在该Endpoint上；
    - 若发现的Endpoint资源的后端端点是Pod，则相关Pod的元标签也会添加在该Endpoint上；

Ingress资源发现

- Prometheus的ingress role负责从API Server中发现Ingress资源；
  - 它把Ingress资源上的每个path视作一个target；
  - 特别适用于对Ingress进行黑盒监控的场景；
  - 相关的地址被设定为Ingress资源上相关host字段的值；
  - 可用的部署meta标签如下
    - __meta_kubernetes_namespace: The namespace of the ingress object.
    - __meta_kubernetes_ingress_name: The name of the ingress object.
    - __meta_kubernetes_ingress_label_<labelname>: Each label from the ingress object.
    - __meta_kubernetes_ingress_annotation_<annotationname>: Each annotation from the ingress object.
    - __meta_kubernetes_ingress_scheme: Protocol scheme of ingress, https if TLS config is set. Defaults to http.
    - __meta_kubernetes_ingress_path: Path from ingress spec. Defaults to /.

基于Kubernetes发现机制的部分配置参数


```
# The API server addresses. If left empty, Prometheus is assumed to run inside of the cluster and will discover API servers automatically and use the pod's
# CA certificate and bearer token file at /var/run/secrets/kubernetes.io/serviceaccount/.
[ api_server: <host> ]

# The Kubernetes role of entities that should be discovered. One of endpoints, service, pod, node, or ingress.
role: <string>

# Optional authentication information used to authenticate to the API server.
#  Note that `basic_auth`, `bearer_token`和`bearer_token_file` 等认证方式互斥;

[ bearer_token: <secret> ]
[ bearer_token_file: <filename> ]

# TLS configuration.
tls_config:
  # CA certificate to validate API server certificate with.
  [ ca_file: <filename> ]

# Certificate and key files for client cert authentication to the server.
  [ cert_file: <filename> ]
  [ key_file: <filename> ]

  # ServerName extension to indicate the name of the server.
  [ server_name: <string> ]

# Optional namespace discovery. If omitted, all namespaces are used.
namespaces:
  names:
    [ - <string> ]
```

- 配置示例参考：https://github.com/iKubernetes/k8s-prom/blob/master/prometheus/prometheus-cfg.yaml

对target重新打标

- 对target重新打标是在数据抓取之前动态重写target标签的强大工具，在每个数据抓取配置中，可以定义多个relabel步骤，它们将按照定义的顺序依次执行；
- 对于发现的每个target，Prometheus默认会执行如下操作
  - job的标签设定为其所属的job_name的值；
  - __address__标签的值为该target的套接字地址“<host>:<port>”
  - instance标签的值为__address__的值；
  - __scheme__标签的值为抓取该target上指标时使用的协议（http或https）；
  - __metrics_path__标签的值为抓取该target上的指标时使用URI路径，默认为/metrics；
  - __param_<name>标签的值为传递的URL参数中第一个名称为<name>的参数的值；
- 重新标记期间，还可以使用该target上以“__meta_”开头的元标签；
  - 各服务发现机制为其target添加的元标签会有所不同；
- 重新标记完成后，该target上以“__”开头的所有标签都会被移除；
  - 若在relabel的过程中需要临时存储标签值，则要使用__tmp标签名称为前缀进行保存，以避免同Prometheus的内建标签冲突；

relabel_config


```
# The source labels select values from existing labels. Their content is concatenated
# using the configured separator and matched against the configured regular expression
# for the replace, keep, and drop actions.
[ source_labels: '[' <labelname> [, ...] ']' ]

# Separator placed between concatenated source label values.
[ separator: <string> | default = ; ]

# Label to which the resulting value is written in a replace action.
# It is mandatory for replace actions. Regex capture groups are available.
[ target_label: <labelname> ]

# Regular expression against which the extracted value is matched.
[ regex: <regex> | default = (.*) ]

# Modulus to take of the hash of the source label values.
[ modulus: <int> ]

# Replacement value against which a regex replace is performed if the
# regular expression matches. Regex capture groups are available.
[ replacement: <string> | default = $1 ]

# Action to perform based on regex matching.
[ action: <relabel_action> | default = replace ]
```

relabel_configs

- <relabel_action>字段用于定义重新标记的行为，其可用取值如下
  - 替换标签值
    - replace：首先将source_labels中指定的各标签的值进行串连，而后将regex字段中的正则表达式对源**标签值**进行匹配判定，若匹配，则将target_label字段中指定的标签的值替换为replacement字段中保存的值；
      - ✓ replacement可按需引用保存regex中的某个“分组模式”匹配到的值；默认保存整个regex匹配到的内容；
      - ✓ 进行值替换时，replacememnt字段中指定标签的值也支持以分组格式进行引用；
    - hashmod：将target_label的值设置为一个hash值，该hash则由modules字段指定的hash模对块对source_labels上各标签的串连值进行hash计算生成；
  - 删除指标：该处的每个指标名称对应一个target
    - keep：regex**不能**匹配到target上的source_labels上的**各标签的串连值**时，则删除该target；
    - drop：regex**能**够匹配到target上的source_labels上的**各标签的串连值**时，则删除该target；
  - 创建或删除标签
    - labelmap：将regex对所有的**标签名**进行匹配判定，而后将匹配到的标签的值赋给replacement字段指定的标签名之上；通常用于取出匹配的标签名的一部分生成新标签；
    - labeldrop：将regex对所有的标签名进行匹配判定，能够匹配到的标签将从该target的标签集中删除；
    - labelkeep：将regex对所有的标签名进行匹配判定，不能匹配到的标签将从该target的标签集中删除；
- 注意：要确保在labeldrop或labelkeep操作后，余下的标签集依然能惟一标识该指标

relabel示例之replace

- 下面示例，将三个源标签的值接顺序串联后，由指定的正则表达式进行模式匹配，而后由replacement引用模式匹配的结果，并加以改造后，将其赋值给endpoint标签；

```
- job_name: 'nodes'
  file_sd_configs:
  - files:
    - targets/prometheus/node*.yaml

  relabel_configs:
  - source_labels:
    - __scheme__
    - __address__
    - __metrics_path__
    regex: "(http|https)(.*)"
    separator: ""
    target_label: "endpoint"
    replacement: "${1}://${2}"
    action: replace
```

- 生成的结果如下图

nodes (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	app="node-exporter" endpoint="http://localhost:9100/metrics" instance="localhost:9100" job="node"	647.000ms	13.940ms	

relabel示例之labelmap

- 下面的示例，将regex指定的模式对target上的所有标签进行匹配判定，对于匹配到的标签名，它将以该标签名中匹配的部分为前缀，指定的“_name”为后缀生成新的标签名，而新标签的值则与其原标签的值相同；

```
- job_name: 'nodes'
  file_sd_configs:
  - files:
    - targets/prometheus/node*.yaml

  relabel_configs:
  - regex: "(job|app)"
    replacement: ${1}_name
    action: labelmap
```

- 生成的结果如下图所示

nodes (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	app="node-exporter" app_name="node-exporter" endpoint="http://localhost:9100/metrics" instance="localhost:9100" job="node" job_name="node"	829.000ms	50.047ms	

对抓取到的metric重新打标

- 对metric重新打标是在数据抓取之后动态重写metric标签的工具，在每个数据抓取配置中，可以定义多个metric relabel的步骤，它们将按照定义的顺序依次执行；
  - 删除不必要的指标；
  - 从指标中删除敏感或不需要的标签；
  - 添加、编辑或修改指标的标签值或标签格式；
- 对metric重新打标的配置格式与target重新打标的格式相同，但前者要定义在专用的metric_relabel_configs字段中；
- 但是要注意的是，更改或添加标签会创建新的时间序列；
  - 应该明确地使用各个标签，并尽可能保持不变，以避免创建出一个动态的数据环境；
  - 标签是时间序列的唯一性约束，删除标签并导致时间序列重复时，可能会导致系统出现问题；

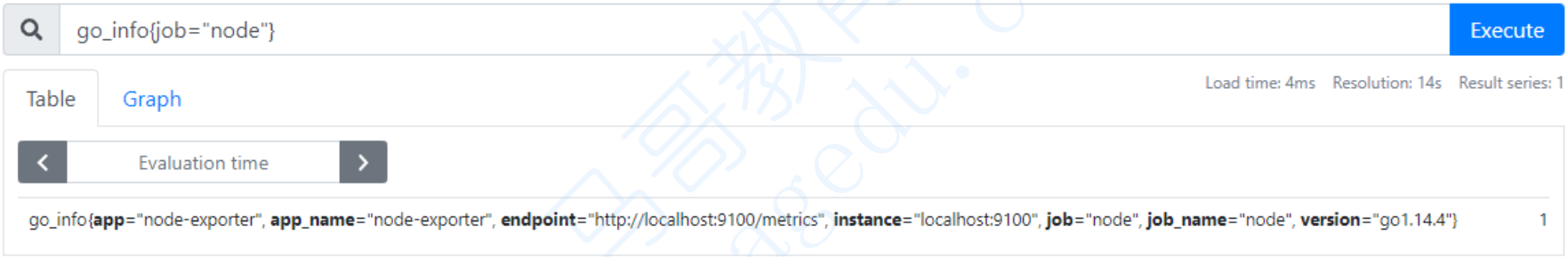
metric relabel示例之删除指标

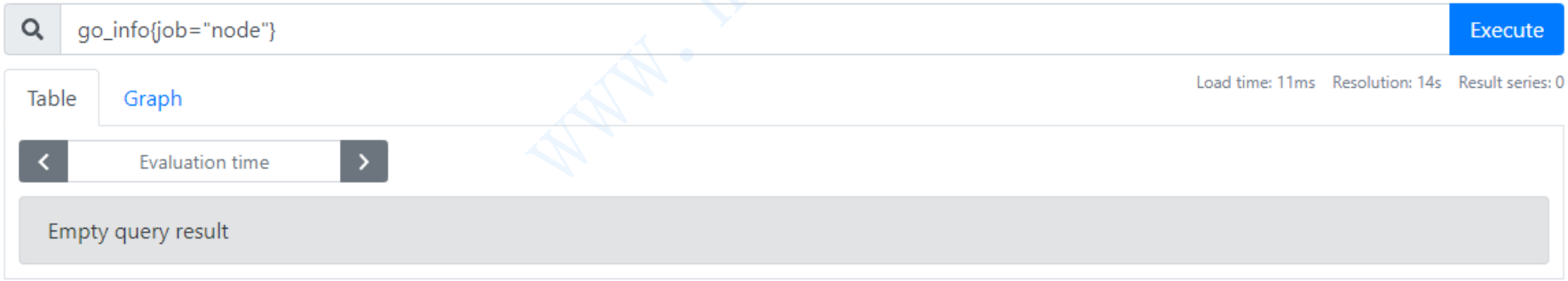
- 在source_labels字段上，通过指标上元标签“__name__”引用指标名称，而后由regex进行匹配判定，可使用drop action删除匹配的指标，或使用keep action仅保留匹配的指标；
- 下面的示例，用于在相应的job上，在发现的各target之上，删除以“go_info”为前名称前缀的指标

```
- job_name: 'nodes'
  file_sd_configs:
  - files:
    - targets/prometheus/node*.yaml

  metric_relabel_configs:
  - source_labels:
    - __name__
    regex: "go_info.*"
    action: drop
```

metric relabel示例之删除指标

- 下面的第一个图为执行go_info为前缀的指标删除之前的查询结果，而第二个图则是删除相关指标之后的查询结果；
  - 提示：若删除的指标此前曾由Prometheus抓取并存储过相关的样本数据，则删除操作的需要经过一定的时长后才会反映至查询结果中；





查询持久化及可视化

讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

查询持久化

- 在Prometheus的表达式浏览器进行的查询会生成的新的数据序列，但其结果仅会临时保存于Prometheus Server上；
- 在样本数据量较大、工作较为繁忙的Prometheus Server上，对于那些查询频率较高且运算较为复杂的查询来说，实时查询可能会存在一定程度的响应延迟；
- 记录规则（Recording rule）能够预先运行频繁用到或计算消耗较大的表达式，并将其结果保存为一组新的时间序列；
  - 记录规则是定义在Prometheus配置文件中的查询语句，由Server加载后，它能够于以类似批处理任务的方式在后台周期性的执行并记录查询结果；
  - 客户端只需要查询由记录规则生成的结果序列上的样本数据即可，速度远快于实时查询；
  - 常用于跨多个时间序列生成聚合数据，或者计算消耗较大的查询等场景中；
  - 多见于同可视化工具结合使用的需求中，也可用于生成可产生告警信息的时间序列；
- 告警规则（Alert rule）是另一种定义在Prometheus配置文件中的PromQL表达式，它通常是一个基于查询语句的布尔表达式，该表达式负责触发告警；
  - 告警规则中用的查询语句较为复杂时，可将其保存为记录规则，而后通过查询该记录规则生成的时间序列来参与比较，从而避免实时查询导致的较长时间延迟；

配置Recording rule

- 记录规则将生成新的时间序列，因而其名称必须是规范的指标名称格式；
- 记录规则必须定义在规则组（rule group）中，各规则按给定的顺序依次运行；
- 规则组的定义语法如下：

```
groups:
  [ - <rule_group> ]
```

<rule_group>

```
# The name of the group. Must be unique within a file.
name: <string>

# How often rules in the group are evaluated.
[ interval: <duration> | default = global.evaluation_interval ]

rules:
  [ - <rule> ... ]
```

配置Recording rule

- 记录规则的定义语法如下

<rule>

The syntax for recording rules is:

```
# The name of the time series to output to. Must be a valid metric name.
record: <string>

# The PromQL expression to evaluate. Every evaluation cycle this is
# evaluated at the current time, and the result recorded as a new set of
# time series with the metric name as given by 'record'.
expr: <string>

# Labels to add or overwrite before storing the result.
labels:
  [ <labelname>: <labelvalue> ]
```

Recording rule使用示例

- Recording rule 示例，通常要保存于单独的文件中，如rules/recording_rules.yaml

```
groups:
- name: custom_rules
  interval: 5s
  rules:
  - record: instance:node_cpu:avg_rate5m
    expr: 100 - avg(irate(node_cpu_seconds_total{job="node", mode="idle"}[5m])) by (instance) * 100

  - record: instace:node_memory_MemFree_percent
    expr: 100 - (100 * node_memory_MemFree_bytes / node_memory_MemTotal_bytes)

  - record: instance:root:node_filesystem_free_percent
    expr: 100 * node_filesystem_free_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"}
```

- 而后在prometheus.yml中通过rule_files加载该文件

```
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
……

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  - "rules/*.yaml“
……
```

Recording rule使用示例

- Web UI中的/rules路径下可展示加载的所有rules

Rules

custom_rules — 10.630s ago — 2.270ms

Rule	State	Error	Last Evaluation	Evaluation Time
record: instance:node_cpu:avg_rate5m expr: 100 - avg by(instance) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100	OK		10.631s ago	0.914ms
record: instace:node_memory_MemFree_percent expr: 100 - (100 * node_memory_MemFree_bytes / node_memory_MemTotal_bytes)	OK		10.630s ago	0.546ms
record: instance:root:node_filesystem_free_percent expr: 100 * node_filesystem_free_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"}	OK		10.630s ago	0.782ms

- 相关指标的使用方式与常规指标并无不同

instance:root:node_filesystem_free_percent — Execute

Table | Graph — Load time: 6ms Resolution: 7s Result series: 1

Evaluation time

instance:root:node_filesystem_free_percent{**app**="node-exporter", **app_name**="node-exporter", **device**="/dev/sda3", **endpoint**="http://localhost:9100/metrics", **fstype**="xfs", **instance**="localhost:9100", **job**="node", **job_name**="node", **mountpoint**="/"} 81.30091928228302

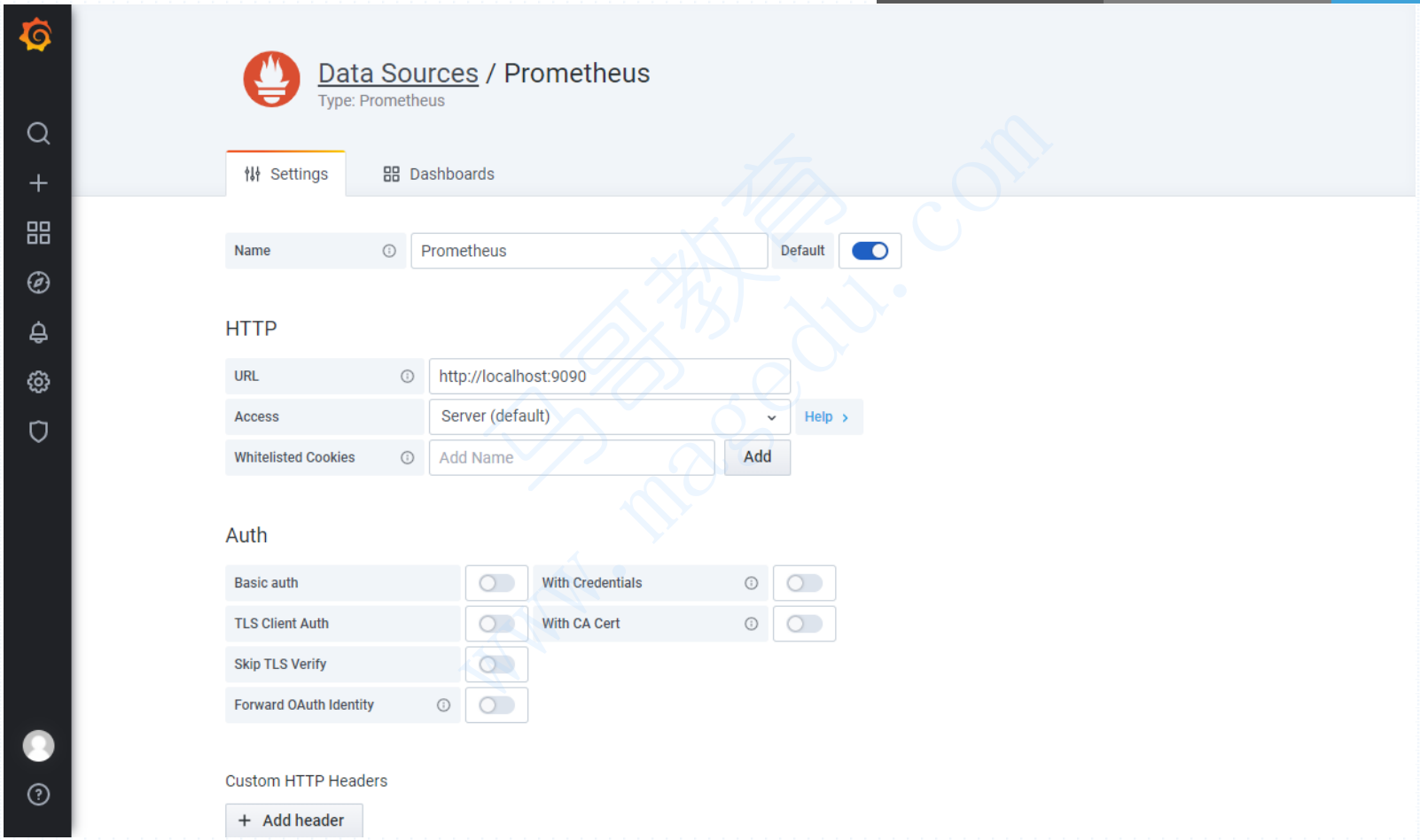
Grafana

- Grafana是一款基于go语言开发的通用可视化工具，支持从多种不同的数据源加载并展示数据，可作为其数据源的部分存储系统如下所示
  - TSDB：Prometheus、IfluxDB、OpenTSDB和Graphit
  - 日志和文档存储：Loki和ElasitchSearch
  - 分布式请求跟踪：Zipkin、Jaeger和Tempo
  - SQL DB: MySQL、PostgreSQL和Microsoft SQL Server
  - …
- Grafana基础
  - 默认监听于TCP协议的3000端口，支持集成其他认证服务，且能够通过/metrics输出内建指标；
  - 几个基本概念
    - 数据源（Data Source）：提供用于展示的数据的存储系统；
    - 仪表盘（Dashboard）：组织和管理数据的可视化面板（Panel）；
    - 团队和用户：提供了面向企业组织层级的管理能力；

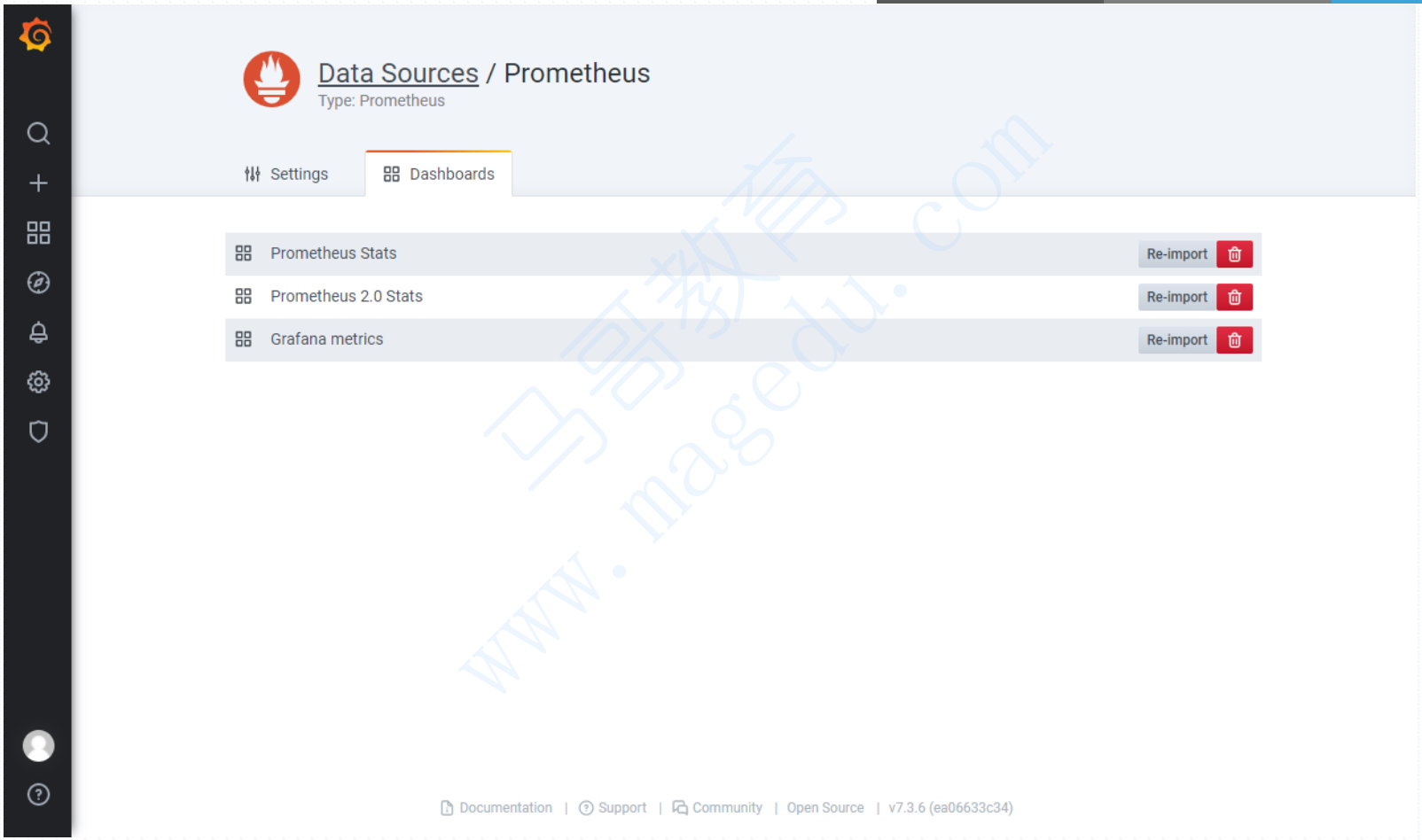
部署Grafana

- Ubuntu/Debian系统上的部署步骤
  - ~# apt-get install -y adduser libfontconfig1
  - ~# VERSION=7.3.6
  - ~# wget https://mirrors.huaweicloud.com/grafana/${VERSION}/grafana_${VERSION}_amd64.deb
  - ~# dpkg -i grafana_${VERSION}_amd64.deb
- RHEL/CentOS系统上的部署步骤
  - ~]# VERSION=7.3.6
  - ~]# wget https://mirrors.huaweicloud.com/grafana/${VERSION}/grafana-${VERSION}-1.x86_64.rpm
  - ~]# yum install grafana-7.3.6-1.x86_64.rpm
- Docker容器运行方式
  - ~]# VERSION=7.3.6
  - ~]# docker run -d --name=grafana -p 3000:3000 grafana/grafana
  - ~]# docker run -d --name=grafana -p 3000:3000 grafana/grafana:${VERSION}-ubuntu

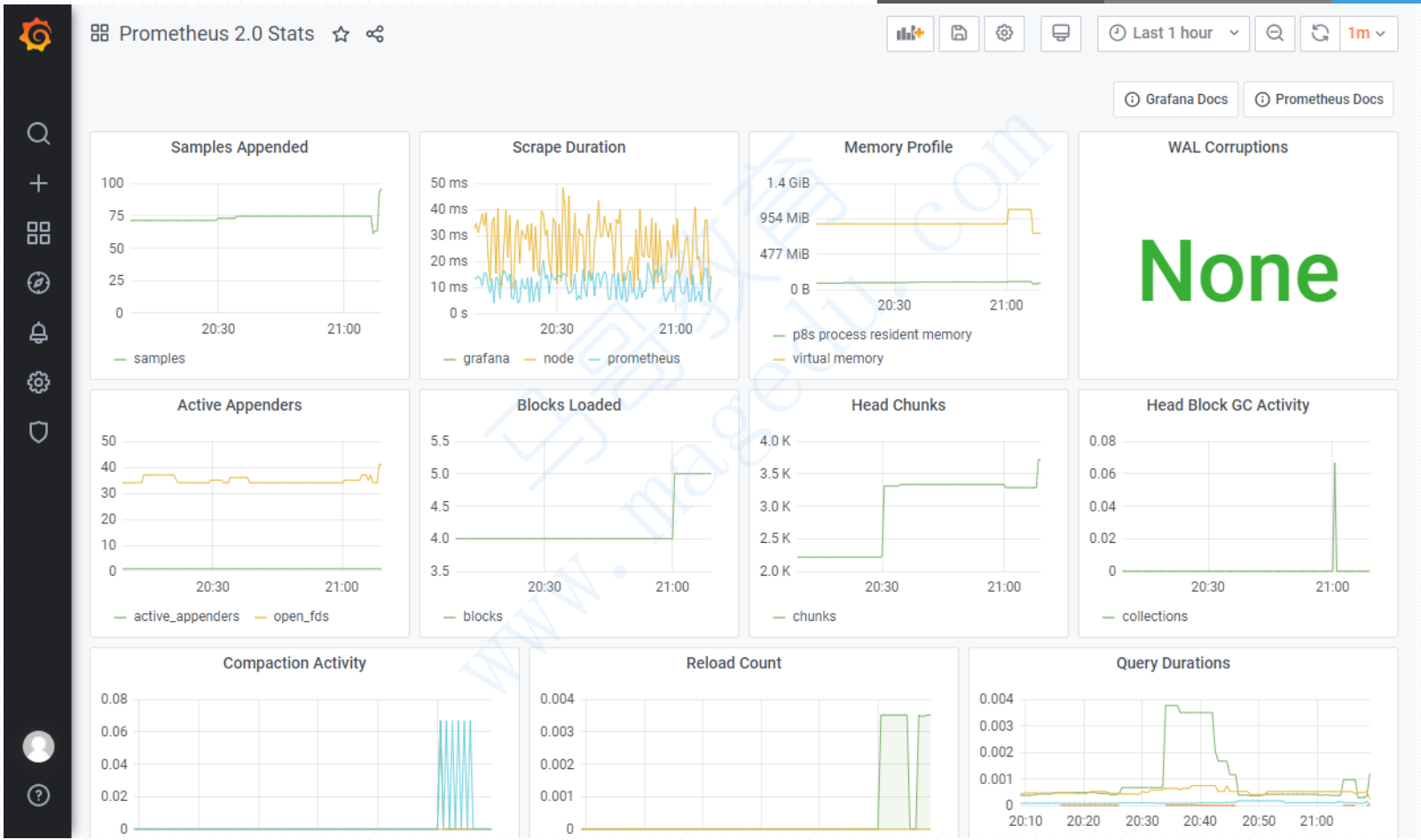
Grafana的数据源



导入内建的Dashboard



Grafana Dashboard示例



监控Grafana

- Grafana直接提供了Instrumetation，指标暴露的路径为/metrics

Alerts

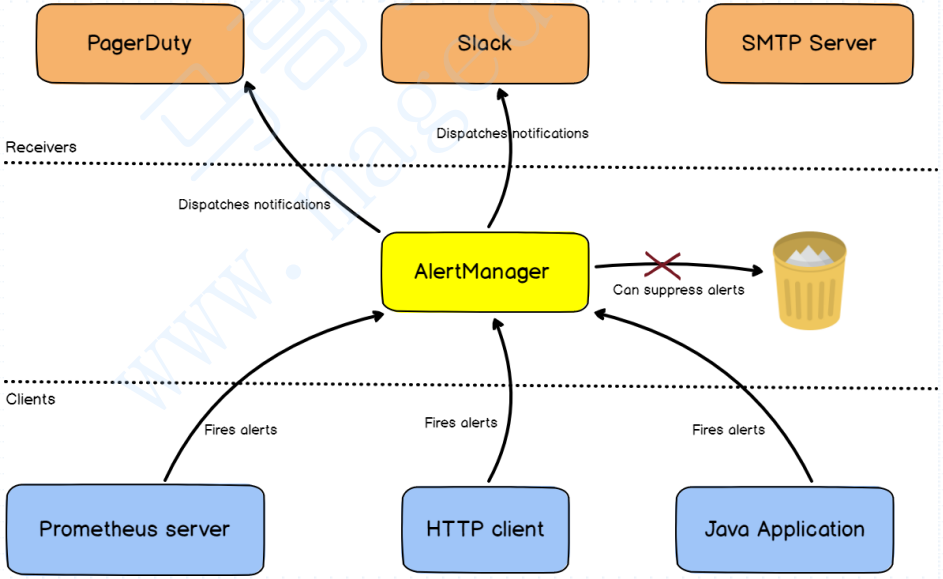
讲师：马永亮（马哥）

http://www.magedu.com

https://github.com/ikubernetes/

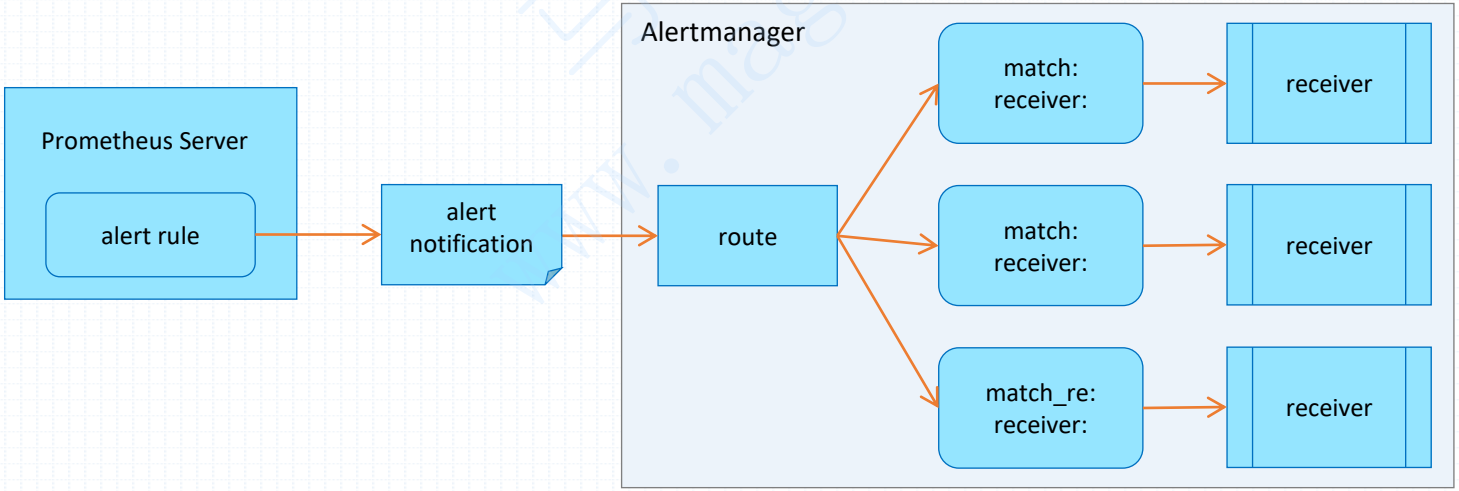
告警功能概述

- Prometheus对指标的收集、存储同告警能力分属于Prometheus Server和AlertManager两个独立的组件，前者仅负责基于“告警规则”生成告警通知，具体的告警操作则由后者完成；
  - Alertmanager负责处理由客户端发来的告警通知
    - 客户端通常是Prometheus Server，但它也支持接收来自其它工具的告警；
    - Alertmanager对告警通知进行分组、去重后，根据路由规则将其路由到不同的receiver，如Email、短信或PagerDuty等；



Prometheus监控系统的告警逻辑

- 首先要配置Prometheus成为Alertmanager的告警客户端；
  - 反过来，Alertmanager也是应用程序，它自身同样应该纳入prometheus的监控目标；
- 配置逻辑
  - 在Alertmanager上定义receiver，他们通常是能够基于某个媒介接收告警消息的特定用户；
    - Email、Wechat、Pagerduty、Slack和Webhook等是为常见的发送告警信息的媒介；
    - 在不同的媒介上，代表告警消息接收人的地址表示方式也会有所不同；
  - 在Alertmanager上定义路由规则（route），以便将收到的告警通知按需分别进行处理；
  - 在Prometheus上定义**告警规则**生成告警通知，发送给Alertmanager；



Alertmanager

- 除了基本的告警通知能力外，Altermanager还支持对告警进行去重、分组、抑制、静默和路由等功能；
  - 分组（Grouping）：将相似告警合并为单个告警通知的机制，在系统因大面积故障而触发告警潮时，分组机制能避免用户被大量的告警噪声淹没，进而导致关键信息的隐没；
  - 抑制（Inhibition）：系统中某个组件或服务故障而触发告警通知后，那些依赖于该组件或服务的其它组件或服务可能也会因此而触发告警，抑制便是避免类似的级联告警的一种特性，从而让用户能将精力集中于真正的故障所在；
  - 静默（Silent）：是指在一个特定的时间窗口内，即便接收到告警通知，Alertmanager也不会真正向用户发送告警信息的行为；通常，在系统例行维护期间，需要激活告警系统的静默特性；
  - 路由（route）：用于配置Alertmanager如何处理传入的特定类型的告警通知，其基本逻辑是根据路由匹配规则的匹配结果来确定处理当前告警通知的路径和行为；

部署Alertmanager

- Altermanager是一个独立的go二进制程序，需要独立部署及维护；
  - 按需下载相应的程序包，展开至特定的目录下即可运行；例如：
    - ~]# tar xf alertmanager-0.21.0.linux-amd64.tar.gz -C /usr/local/
    - ~]# ln -s /usr/local/alertmanager-0.21.0.linux-amd64 /usr/local/alertmanager

alertmanager

Prometheus Alertmanager prometheus/alertmanager

0.21.0 / 2020-06-16 Release notes

File name	OS	Arch	Size	SHA256 Checksum
alertmanager-0.21.0.darwin-386.tar.gz	darwin	386	23.98 MiB	1d2af6bcebf6de204bca2ed650133883eb2f35844000299248ab9efce77e1b77
alertmanager-0.21.0.darwin-amd64.tar.gz	darwin	amd64	24.26 MiB	b508eb6f5d1b957abbee92f9c37deb99a3da3311572bed34357ac8acbb75e510
alertmanager-0.21.0.dragonfly-amd64.tar.gz	dragonfly	amd64	24.43 MiB	8531f3b02d788df4fabe333dc75a558cd92c51c4c8280d5a27dbaa0efb390f66
alertmanager-0.21.0.freebsd-386.tar.gz	freebsd	386	23.95 MiB	bd554804e0240b1cba8d3f9f1bb5cfde6f1c815ec3b77e83e617808ca0983420
alertmanager-0.21.0.freebsd-amd64.tar.gz	freebsd	amd64	24.48 MiB	1fe3f1fcdebb055d464d24865743f996843166bb7c8f3cea114887f5bb283b2d
alertmanager-0.21.0.freebsd-armv6.tar.gz	freebsd	armv6	22.86 MiB	3cc30810003e21d06f1f5534a284bacf077d04bc8f86e4b82f6b189d45d248df
alertmanager-0.21.0.freebsd-armv7.tar.gz	freebsd	armv7	22.85 MiB	dec886a05913e210a8b0f109e713b6f394d2afddb5395da045a8823d2d7f9c83
alertmanager-0.21.0.linux-386.tar.gz	linux	386	24.01 MiB	cfa6090466b530ea7501fcded4a186516cac22c149cda1f50dd125177badff9d
alertmanager-0.21.0.linux-amd64.tar.gz	linux	amd64	24.52 MiB	9ccd863937436fd6bfe650e22521a7f2e6a727540988eef515dde208f9aef232
alertmanager-0.21.0.linux-arm64.tar.gz	linux	arm64	22.97 MiB	1107870a95d51e125881f44aebbc56469e79f5068bd96f9eddff38e7a5ebf423

Alertmanager基础配置示例

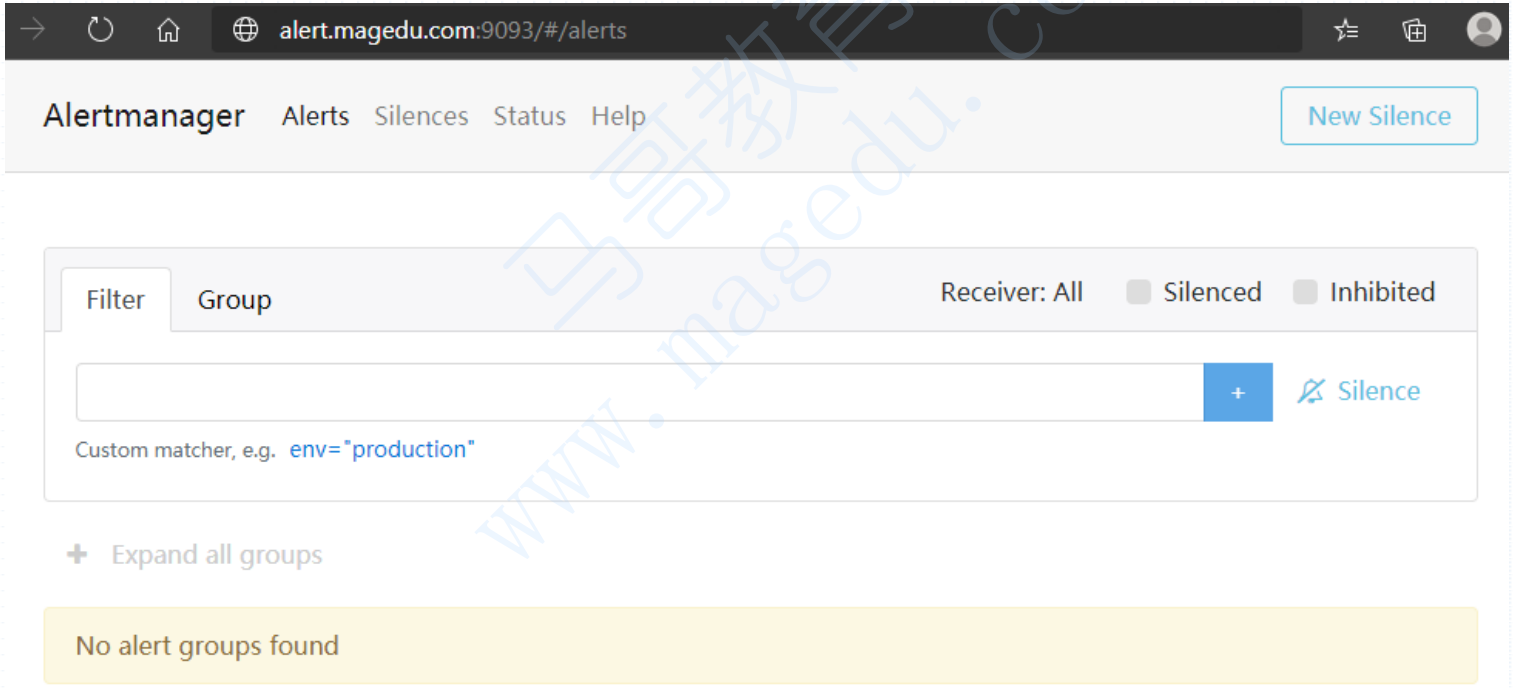
- Alertmanager的配置文件遵循YAML格式，通常给予一个类似如下的基础配置，即可启动相关的进行；

```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@magedu.com'
  smtp_require_tls: false

templates:
- 'templates/*.yaml'

route:
  receiver: email-receiver

receivers:
- name: 'email-receiver'
  email_configs:
  - to: ‘root@magedu.com‘, ‘root@localhost’
```

  - global配置段用于定义全局配置、templates配置段负责定义告警模板文件、route配置段用于指定如何处理传入的告警，示例中仅定义了基本的路由信息（默认路由），而receiver配置段则定义了告警信息的接收器，每个接收器都应该有其具体的定义；
  - 提示：上面配置中用于处理和发送告警信息的媒介是运行于本地节点上的SMTP服务，因而需要安装配置postfix一类的程序以完成测试；

启动Alertmanager

- 将示例中的配置保存在文件中，即可启动alertmanager：
  - ~]# ./alertmanager --config.file=alertmanager.yml
  - 内建的HTTP服务默认监听于TCP协议的9093端口，且/metrics路径暴露了其内建的指标；



组合Prometheus与Altermanager

- 调用的Alertmanager实例的配置信息定义在顶级的alerting配置段中，它同样支持静态配置和动态发现的机制，其配置方式类似于被监控的target的定义；
  - 静态配置示例

```
# Alertmanager configuration
alerting:
 alertmanagers:
 - static_configs:
   - targets:
     - alert.magedu.com:9093
```

  - 动态配置示例，右侧为基于文件发现时，其用到的配置示例；

```
# Alertmanager configuration
alerting:
 alertmanagers:
 - file_sd_configs:
   - files:
     - "targets/alertmanager*.yaml"
```

```
# Alertmanager Target file
- targets:
  - alert.magedu.com:9093
  labels:
    app: alertmanager
    job: alertmanager
```

  - 还应该将Alertmanager程序作为监控目标，例如将如下配置添加到scrape_configs中；

```
- job_name: 'alertmanager'
  file_sd_configs:
  - files:
    - targets/prometheus/alertmanager*.yaml
```

配置Prometheus加载告警规则文件

- 类似于记录规则，告警规则（Alerting rule）也定义在独立的文件中，而后由Prometheus在rule_files配置段中加载；
  - 尽管可以将所有的告警规则定义在同一个文件中，但出于便捷管理之目的，建议将其按功能分别放置于不同的文件中；
  - rule_files配置段加载告警规则的文件，同样可以使用文件名通配机制；
  - 下面的配置示例中，我们假设所有的规则文件放置于专用的alert_rules目录下，均以“.yaml”为文件名后缀，于是一条文件加载配置项即可加载相应目录下所有的告警规则文件；

```
rule_files:
  - "rules/*.yaml"
  - “alert_rules/*.yaml“      # 专用于加载告警规则相关的文件；
```

添加告警规则

- 根据配置，在alert_rules目录中所有以“.yaml”为后缀的文件均被识别为告警规则文件；下面便是一条告警规则示例
  - 若某个Instance的up指标的值转为0持续超过1分钟后，将触发告警；

```
groups:
- name: AllInstances
  rules:
  - alert: InstanceDown
    # Condition for alerting
    expr: up == 0
    for: 1m
    # Annotation - additional informational labels to store more information
    annotations:
      title: 'Instance down'
      description: Instance has been down for more than 1 minute.'
    # Labels - additional labels to be attached to the alert
    labels:
      severity: 'critical'
```

告警规则

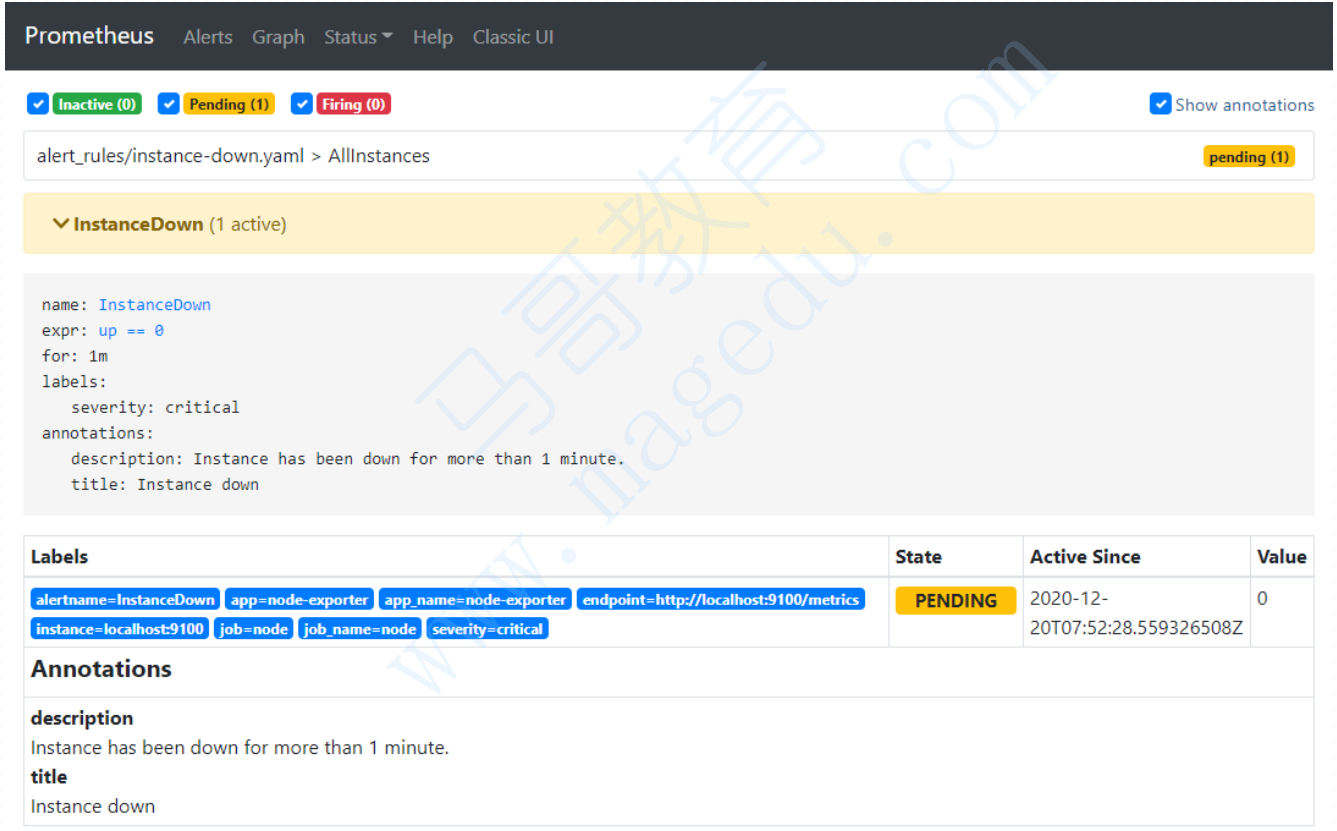
- 类似于记录规则，有着类似或相关联功能的告警规则同样可以组织为group，从而为规则名称提供“名称空间”；一个组内的每个告警规则必须有个名称，且在该组内其名称必须惟一；
  - alert：告警规则的名称；
  - expr：基于PromQL表达式的告警触发条件（布尔表达式），用于计算是否有时间序列可以满足该条件；可以使用由Recording rule定义的指标；
  - for：控制在触发告警之前，测试表达式的值必须为true的时长；
    - 表达式值为true，但其持续时间未能满足for定义的时长时，相关的告警状态为pending；
    - 满足该时长之后，相关的告警将被触发，并转为firing状态；
    - 表达式的值为false时，告警将处于inactive状态；
  - labels：告警规则被激活时，相关时间序列上的所有标签都会添加到生成告警实例之上，而labels则允许用户在告警上附加其它自定义的标签；该类标签值支持**模板化**；
    - 告警的名称及其标签即为告警的标识，因而，这类似于时间序列的标识机制；
  - annotations：附加在告警之上的注解信息，其格式类似于标签，但不能被用于标识告警实例；经常用于存储告警摘要，且其值支持模板化；

触发告警

- Prometheus以一个固定的周期来评估所有告警规则，其值由evaluate_interval参数定义，默认为15s；在每个评估周期内，Prometheus会计算每条告警规则中的布尔表达式并更新告警状态；
- 未使用for子句的告警，会自动从Inactive状态转为Firing，它只需要一个评估周期便能触发；而带有for子句的告警状态将先转为Pending，而后才到Firing，因而至少需要两个评估周期才能触发；
  - 处于Pending状态的告警，在其持续时长满足for子句的定义之前，若布尔表达式的值转回了false，则告警状态将转回Inactive；
  - 由此可见，经由Pending再到Firing的转换，可以确保告警更有效，且不会来回浮动；
- Prometheus将为Pending和Firing状态中的每个告警创建指标，该指标名称为ALERT，其值固定为1，并且在告警处于Pending和Firing状态期间存在；在此之后，它将不接收任何更新，直到过期；

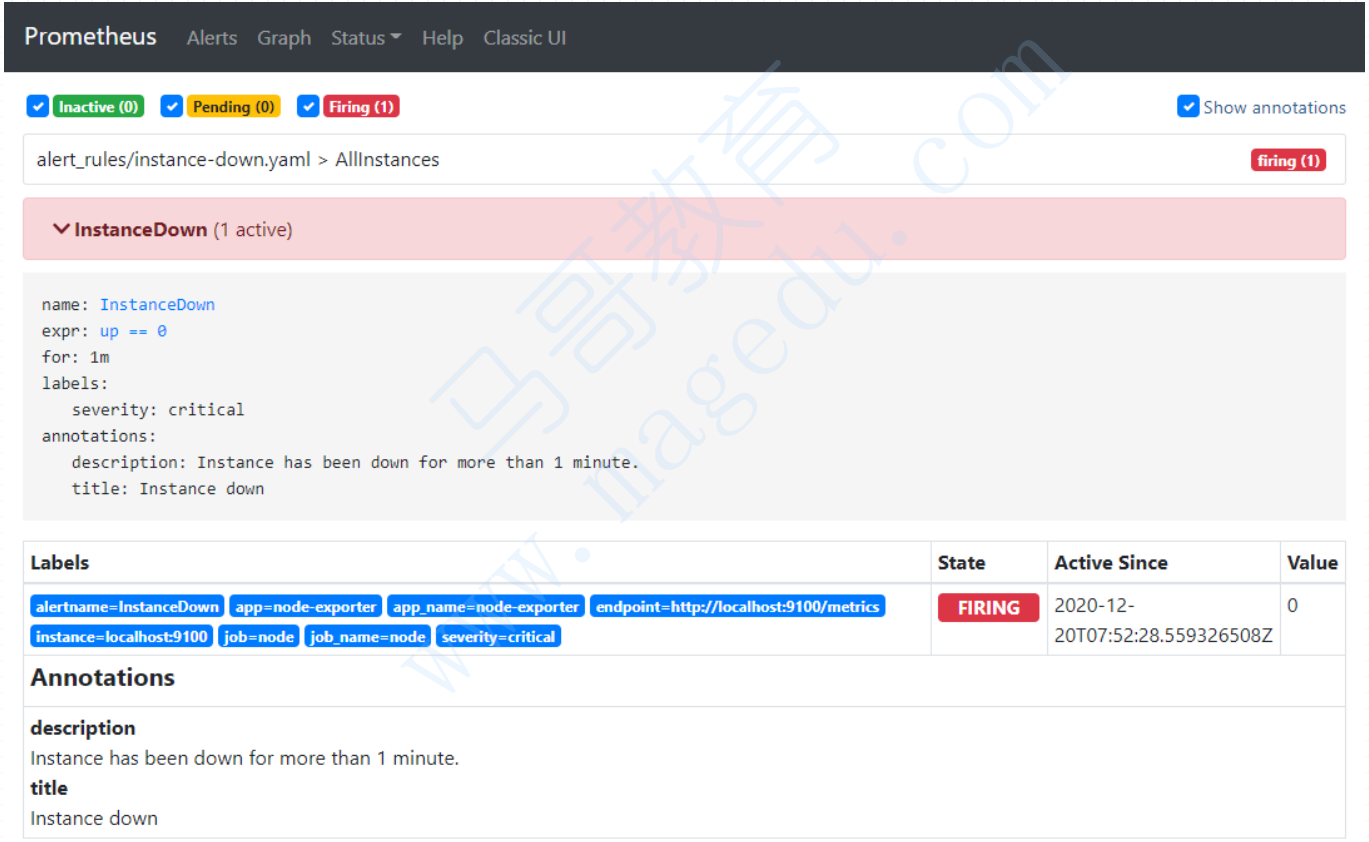
触发告警

- Pending状态的告警



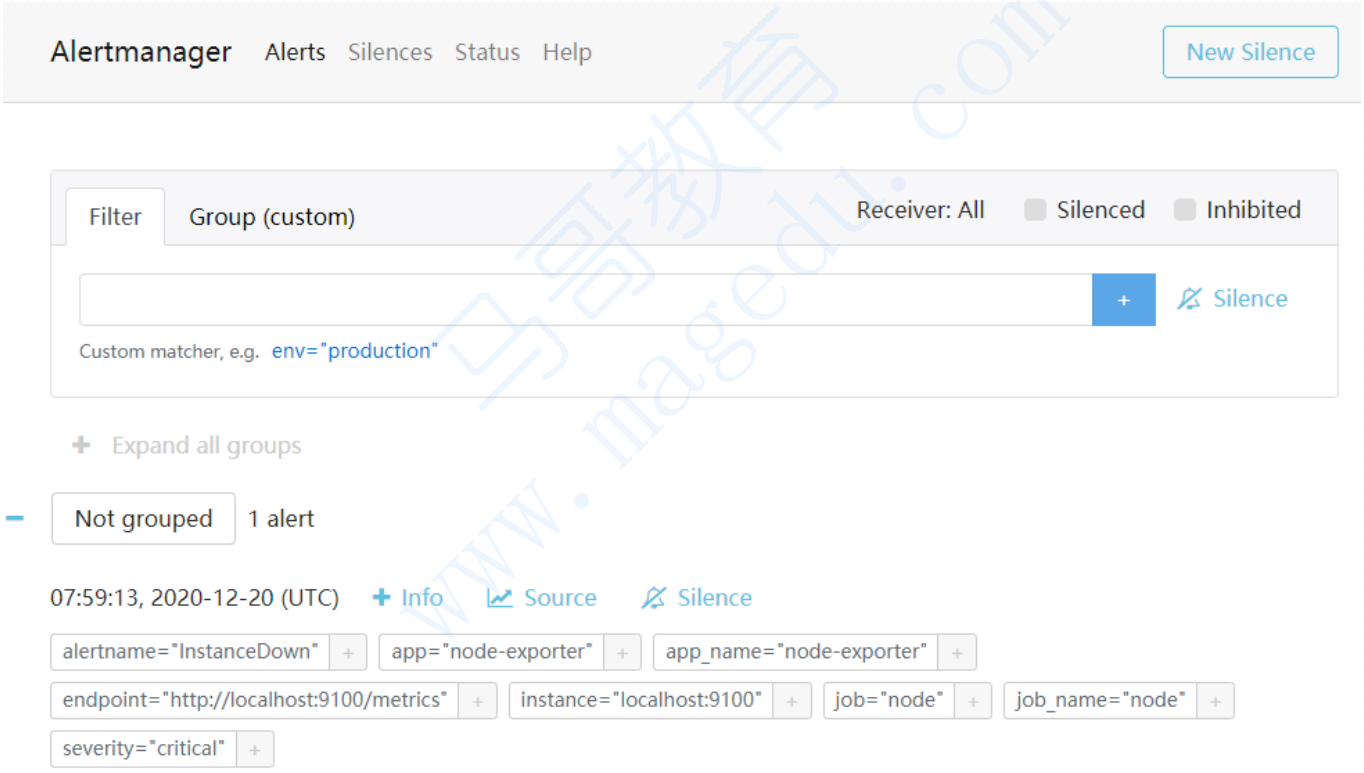
触发告警

- 已发出的告警，告警将被Alertmanager接收到



Alertmanager上的告警信息

- 告警被触后，便可在Alertmanager的Web UI上看到相关的信息，而且也将被告警信息也将被发送到目标email



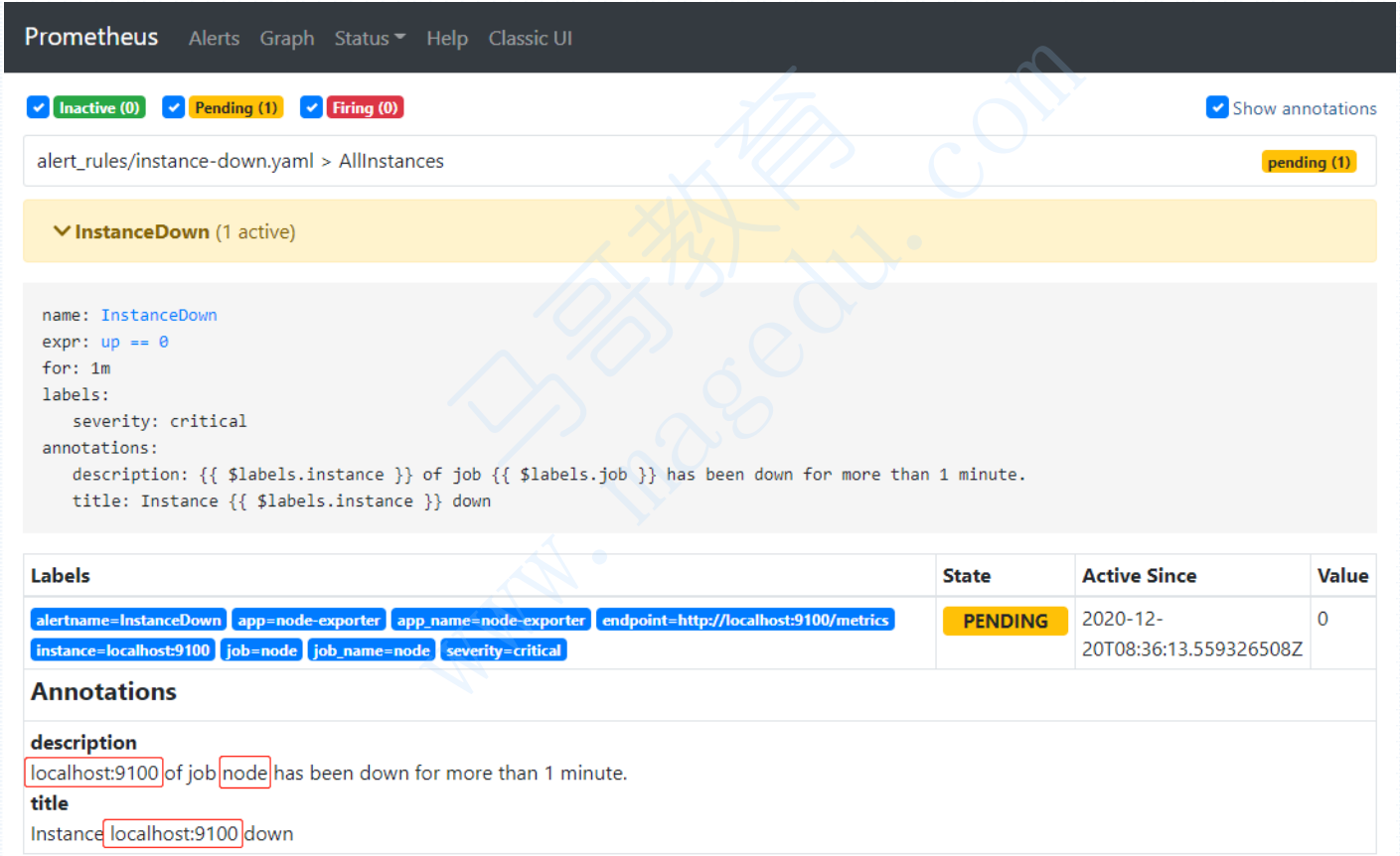
告警模板

- Prometheus支持在告警中使用模板；
  - 告警模板是指在告警中的标签和注解上引用时间序列的标签和样本值的方法；
  - 它使用标准的Go语法，并暴露一些包含时间标签和值的变量；
    - 标签引用：{{ $label.<label_name> }}
    - 指标样本值引用：{{ $value }}
  - 若要在description注解中引用触发告警的时间序列上的instance和job标签的值，可分别使用“{{ $label.instance }}”和“{{ $label.job }}”；例如，我们可以把此前的告警规则改为如下模板格式

```
groups:
- name: AllInstances
  rules:
  - alert: InstanceDown
    # Condition for alerting
    expr: up == 0
    for: 1m
    # Annotation - additional informational labels to store more information
    annotations:
      title: 'Instance {{ $labels.instance }} down'
      description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute.'
    # Labels - additional labels to be attached to the alert
    labels:
      severity: 'critical'
```

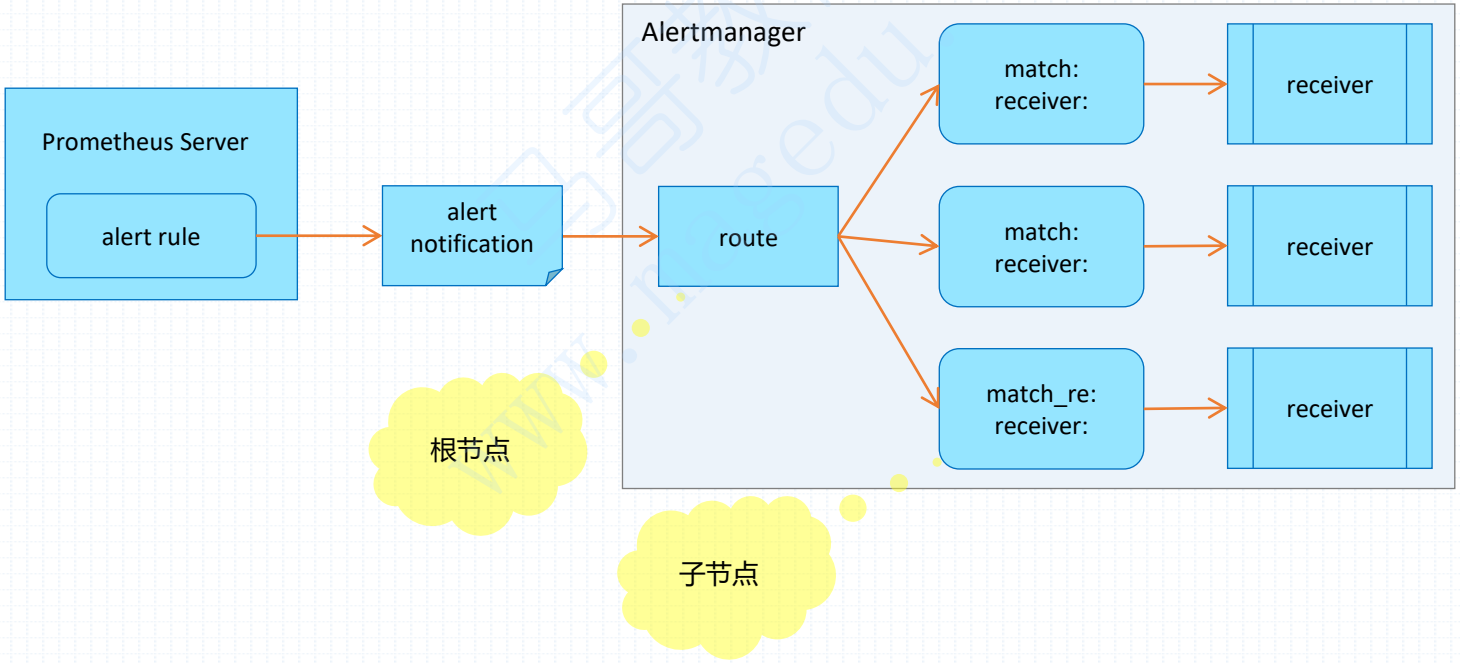
告警模板

- 经渲染后的模板字符替换的结果如图所示



告警路由

- Alertmanager的route配置段支持定义“树”状路由表，入口位置称为根节点，每个子节点可以基于匹配条件定义出一个独立的路由分支；
  - 所有告警都将进入路由根节点，而后进行子节点遍历；
  - 若路由上的continue字段的值为false，则遇到第一个匹配的路由分支后即终止；否则，将继续匹配后续的子节点；



告警路由

route配置段的可用配置参数

```
[ receiver: <string> ]
# 分组时使用的标签，默认情况下，所有的告警都组织在一起，而一旦指定分组标签，则Alertmanager将按这些标签进行分组；
[ group_by: '[' <labelname>, ... ']' ]

# Whether an alert should continue matching subsequent sibling nodes.
[ continue: <boolean> | default = false ]

# A set of equality matchers an alert has to fulfill to match the node.
match:
  [ <labelname>: <labelvalue>, ... ]

# A set of regex-matchers an alert has to fulfill to match the node.
match_re:
  [ <labelname>: <regex>, ... ]

# 发出一组告警通知的初始等待时长；允许等待一个抑制告警到达或收集属于同一组的更多初始告警，通常是0到数分钟；
[ group_wait: <duration> | default = 30s ]

# 发送关于新告警的消息之前，需要等待多久；新告警将被添加到已经发送了初始通知的告警组中；一般在5分钟或以上；
[ group_interval: <duration> | default = 5m ]

# 成功发送了告警后再次发送告警信息需要等待的时长，一般至少为3个小时；
[ repeat_interval: <duration> | default = 4h ]

# 子路由配置
routes:
  [ - <route> ... ]
```

告警路由示例

- 第一步，在Prometheus上新增一些告警规则，例如下面的两个

```
groups:
- name: nodes_alerts
  rules:
  - alert: DiskWillFillIn12Hours
    expr: predict_linear(node_filesystem_free_bytes{mountpoint="/"}[1h], 12*3600)
    for: 1m
    labels:
      severity: critital
    annotations:
      discription: Disk on {{ $label.instance }} will fill in approximately 12 hours.

- name: prometheus_alerts
  rules:
  - alert: PrometheusConfigReloadFailed
    expr: prometheus_config_last_reload_successful == 0
    for: 3m
    labels:
      serverity: warning
    annotations:
      description: Reloading Prometheus configuration has failed on {{ $labels.instance }}.
```

告警路由示例

- 第二步，添加路由信息

```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@magedu.com'
  smtp_require_tls: false

templates:
- 'templates/*.yaml'

route:
  group_by: [‘instance’]
  group_wait: 30s
  group_interval: 2m
  repeat_interval: 5m
  receiver: email-receiver
  routes:
  - match:
      severity: critical
    reciver: support-team
  - match_re:
      severity: ^(warning|critical)$
    receiver: email-receiver
```

```
receivers:
- name: 'email-receiver'
  email_configs:
  - to: ‘root@magedu.com‘, ‘root@localhost’
- name: ‘support-team'
  email_configs:
  - to: ‘root@magedu.com‘, ‘root@localhost’
```

Receiver

Inhibit Rule

关于马哥教育

- http://www.magedu.com
- https://github.com/ikubernetes

Thank You!