

作者：光大科技-吴涛

## 一、概述

随着容器化的大力发展，容器云平台已经基本由Kubernetes作为统一的容器管理方案。当我们使用Kubernetes进行容器化管理时，传统监控工具如Zabbix无法对Kubernetes做到统一有效的全面监控，全面监控Kubernetes也就成为我们需要探索的问题。使用容器云监控，旨在全面监控Kubernetes集群、节点、服务、实例的统计数据，验证集群是否正常运行并创建相应告警。本章旨在于介绍容器云平台监控的架构设计及优化。

## 二、价值和意义

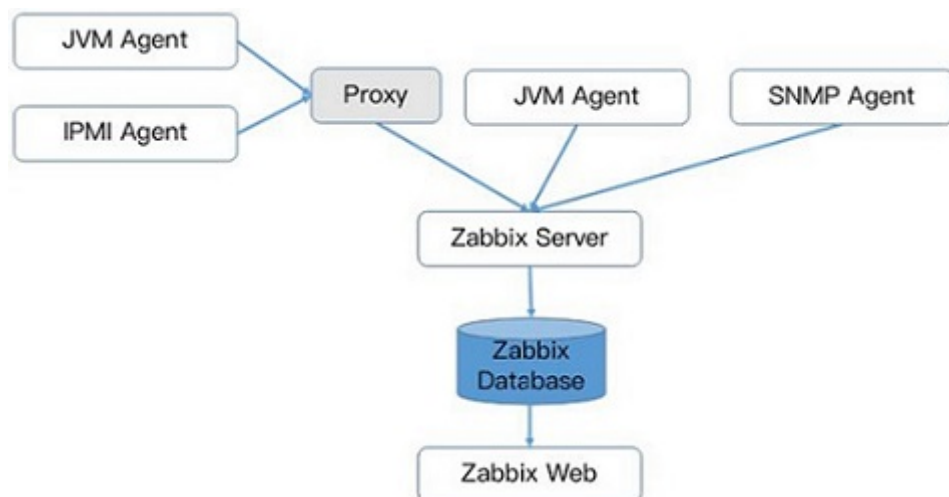
监控是运维体系中是非常重要的组成部分，通过监控可以实时掌握系统运行状态，对故障提前预警，以及历史状态的回放，还可以通过监控数据为系统的容量规划提供辅助决策，为系统性能优化提供真实的用户行为和体验。为容器云提供良好的监控环境是保证容器服务的高可靠性、高可用性和高性能的重要部分，通过对本章的学习，能够快速认识当前容器环境下都有哪些监控方案，并对主流的监控方案有一个系统的了解和认识。

## 三、监控方案选型

### 3.1 容器云监控方案有哪些

#### (1) Zabbix

Zabbix是由Alexei Vladishev开源的分布式监控系统，支持多种采集方式和采集客户端，同时支持SNMP、IPMI、JMX、Telnet、SSH等多种协议，它将采集到的数据存放到数据库中，然后对其进行分析整理，如果符合告警规则，则触发相应的告警。

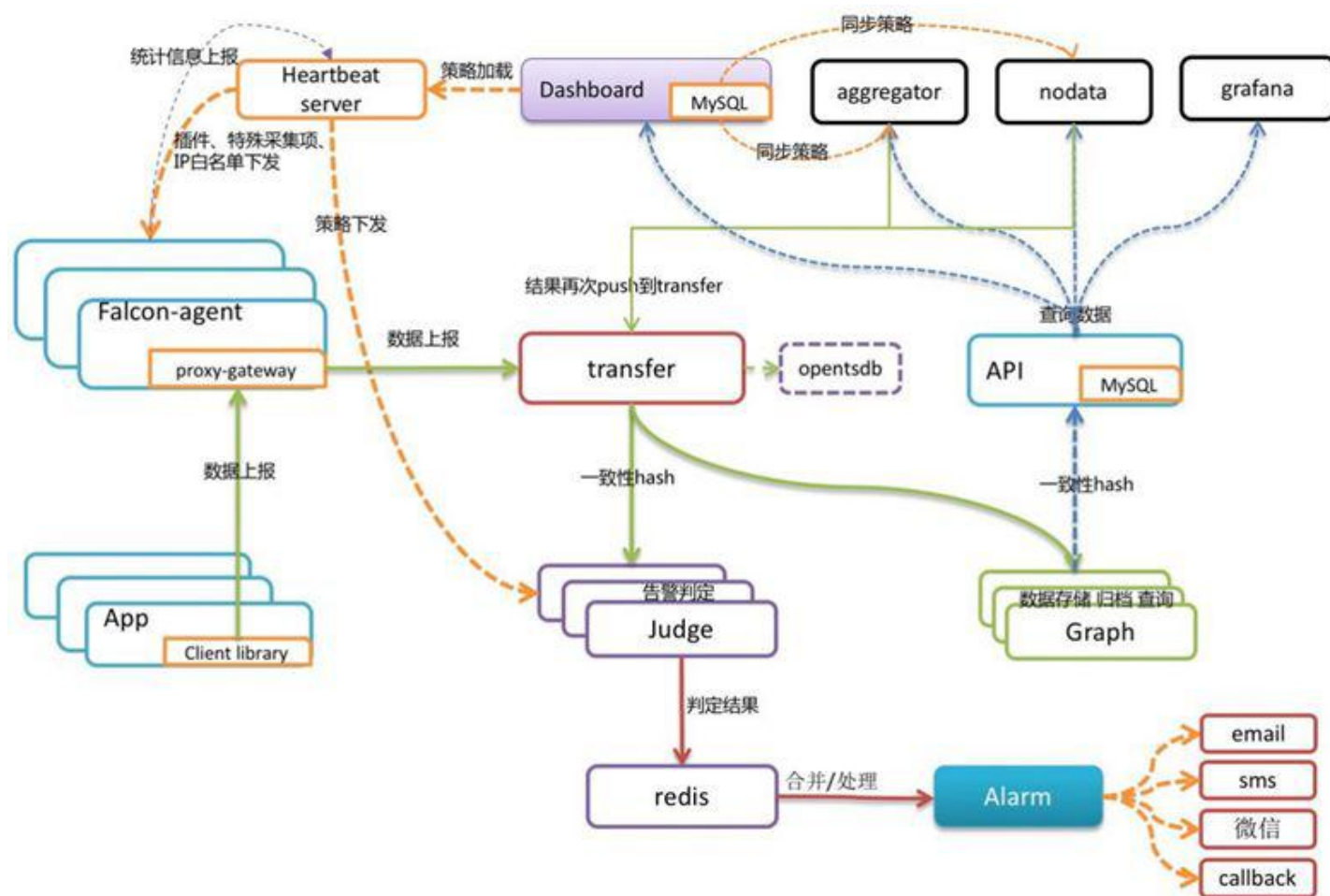


Zabbix核心组件主要是Agent和Server，其中Agent主要负责采集数据并通过主动或者被动的方式采集数据发送到Server/Proxy，除此之外，为了扩展监控项，Agent还支持执行自定义脚本。Server主要负责接收Agent发送的监控信息，并进行汇总存储，触发告警等。

Zabbix Server将收集的监控数据存储到Zabbix Database中。Zabbix Database支持常用的关系型数据库，如MySQL、PostgreSQL、Oracle等，默认是MySQL，并提供Zabbix Web页面（PHP编写）数据查询。

Zabbix由于使用了关系型数据存储时序数据，所以在监控大规模集群时常常在数据存储方面捉襟见肘。所以从Zabbix 4.2版本后开始支持TimescaleDB时序数据库，不过目前成熟度还不高。

## (2) Open-Falcon



Open-Falcon是小米开源的企业级监控工具，用Go语言开发而成，包括小米、滴滴、美团等在内的互联网公司都在使用它，是一款灵活、可扩展并且高性能的监控方案，主要组件包括了：

1) Falcon-agent是用Go语言开发的Daemon程序，运行在每台Linux服务器上，用于采集主机上的各种指标数据，主要包括CPU、内存、磁盘、文件系统、内核参数、Socket连接等，目前已经支持200多项监控指标。并且，Agent支持用户自定义的监控脚本。

2) Heartbeat server简称HBS心跳服务，每个Agent都会周期性地通过RPC方式将自己的状态上报给HBS，主要包括主机名、主机IP、Agent版本和插件版本，Agent还会从HBS获取自己需要执行的采集任务和自定义插件。

3) Transfer负责接收Agent发送的监控数据，并对数据进行整理，在过滤后通过一致性Hash算法发送到Judge或者Graph。

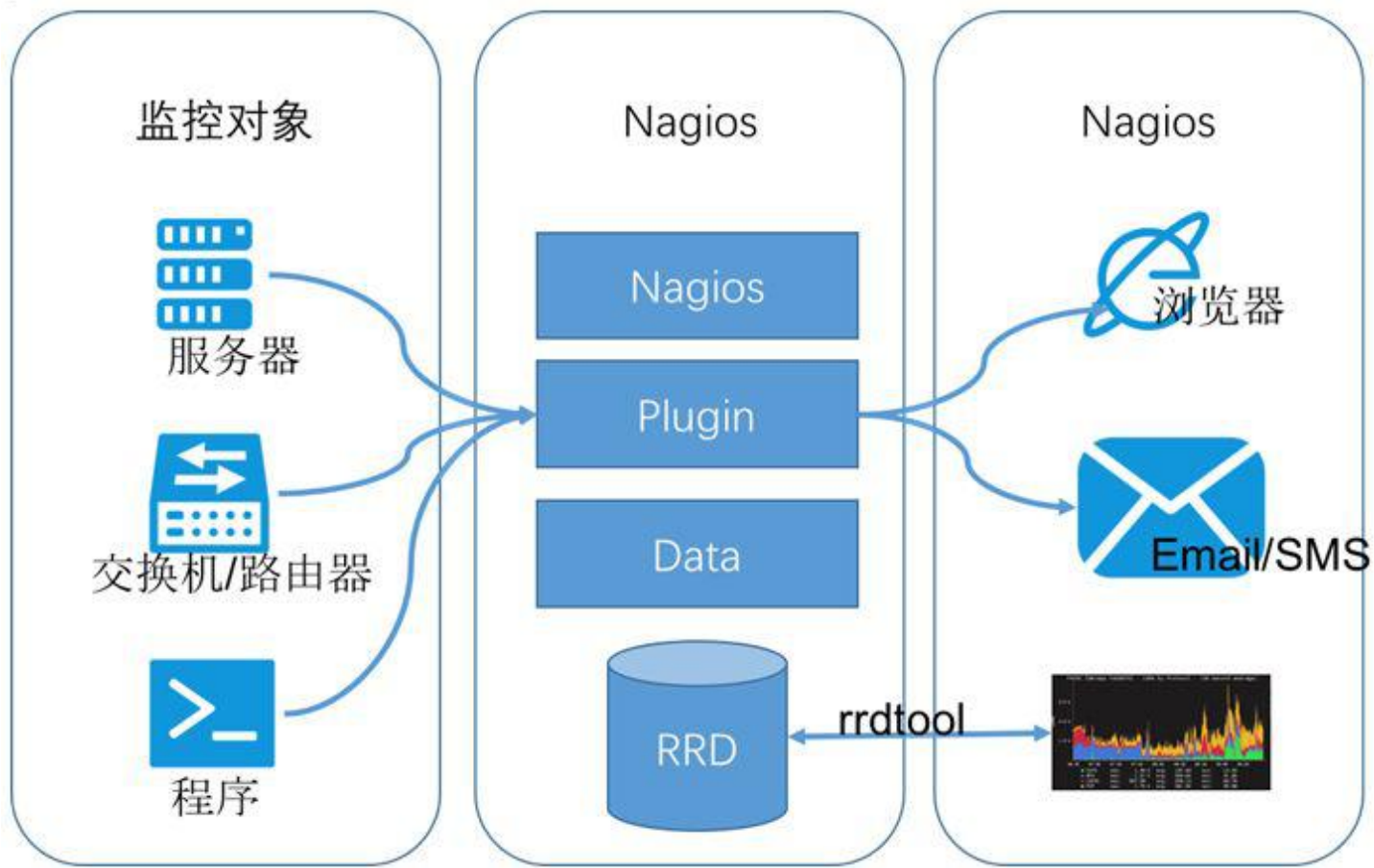
4) Graph是基于RRD的数据上报、归档、存储组件。Graph在收到数据以后，会以rrdtool的数据

归档方式来存储，同时提供RPC方式的监控查询接口。

5) Judge告警模块，Transfer转发到Judge的数据会触发用户设定的告警规则，如果满足，则会触发邮件、微信或者回调接口。这里为了避免重复告警引入了Redis暂存告警，从而完成告警的合并和抑制。

6) Dashboard是面向用户的监控数据查询和告警配置界面。

(3) Nagios



Nagios原名为NetSaint，由Ethan Galstad开发并维护。Nagios是一个老牌监控工具，由C语言编写而成，主要针对主机监控（CPU、内存、磁盘等）和网络监控（SMTP、POP3、HTTP和NNTP等），当然也支持用户自定义的监控脚本。

它还支持一种更加通用和安全的采集方式NREP（Nagios Remote Plugin Executor），它首先在远端启动一个NREP守护进程，用于在远端主机上面运行检测命令，在Nagios服务端用check nrep的plugin插件通过SSL对接到NREP守护进程执行相应的监控行为。相比SSH远程执行命令的方式，这种方式更加安全。

(4) Prometheus

Prometheus 是一个很受欢迎的开源监控和警报工具包，继Kubernetes之后成为第二个正式加入CNCF基金会的项目，2017年底发布了基于全新存储层的2.0版本，能更好地与容器云平台配合。能实现docker status、cAdvisor的监控功能，并且Prometheus原生支持Kubernetes监控，具有Kubernetes对象服务发现能力，Kubernetes的核心组件也提供了Prometheus的采集接口。单个Prometheus可以每秒抓取10万的metrics，能满足一定规模下k8s集群的监控需求，并且具备良好的查询能力，提供数据查询语言 PromQL，PromQL 提供了大量的数据计算函数，大部分情况下用户都可以直接通过PromQL 从 Prometheus 里查询到需要的聚合数据。

## 3.2 方案对比并确定

通过以下方面对上述监控方案进行对比：

1) 从开发语言上看，为了应对高并发和快速迭代的需求，监控系统的开发语言已经慢慢从C语言转移到Go。不得不说，Go凭借简洁的语法和优雅的开发，在Java占据业务开发，C占领底层开发的情况下，准确定位中间件开发需求，在当前开源中间件产品中被广泛应用。

2) 从系统成熟度上看，Zabbix和Nagios都是老牌的监控系统，系统功能比较稳定，成熟度较高。而Prometheus和Open-Falcon都是最近几年才诞生的，虽然功能还在不断迭代更新，但站在巨人的肩膀之上，在架构设计上借鉴了很多老牌监控系统的经验。

3) 从系统扩展性方面看，Zabbix和Open-Falcon都可以自定义各种监控脚本，并且Zabbix不仅可以做到主动推送，还可以做到被动拉取，Prometheus则定义了一套监控数据规范，并通过各种exporter扩展系统采集能力。

4) 从数据存储方面来看，Zabbix采用关系数据库保存，这极大限制了Zabbix采集的性能，Nagios和Open-Falcon都采用RDD数据存储，Open-Falcon还加入了一致性hash算法分片数据，并且可以对接到OpenTSDB，而Prometheus自研一套高性能的时序数据库，在V3版本可以达到每秒千万级别的数据存储，通过对接第三方时序数据库扩展历史数据的存储。

5) 从配置复杂度上看，Prometheus只有一个核心server组件，一条命令便可以启动，相比而言，其他系统配置相对麻烦，尤其是Open-Falcon。

6) 从社区活跃度上看，目前Zabbix和Nagios的社区活跃度比较低，尤其是Nagios，Open-Falcon虽然也比较活跃，但基本都是国内的公司参与，Prometheus在这方面占据绝对优势，社区活跃度最高，并且受到CNCF的支持，后期的发展值得期待。

7) 从容器支持角度看，由于Zabbix和Nagios出现得比较早，当时容器还没有诞生，自然对容器的支持也比较差。Open-Falcon虽然提供了容器的监控，但支持力度有限。Prometheus的动态发现机制，不仅可以支持swarm原生集群，还支持Kubernetes容器集群的监控，是目前容器监控最好解决方案。

Zabbix在传统监控系统中，尤其是在服务器相关监控方面，占据绝对优势。而Nagios则在网络监控方面有广泛应用，伴随着容器的发展，Prometheus开始成为主导及容器监控方面的标配，并且在未来可见的时间内被广泛应用。总体来说，对比各种监控系统的优劣，Prometheus可以说是目前监控领域最锋利的“瑞士军刀”了。

## 四、基于prometheus的容器云平台监控架构设计

### 4.1 prometheus介绍

Prometheus是由SoundCloud开发的开源监控报警系统和时序列数据库。于2016年加入Cloud Native Computing Foundation，作为继Kubernetes之后的第二个托管项目，具有强大的数据采集、数据存储、数据展示、告警等功能，天生完美支持kubernetes。

主要特点：

- (1) 多维数据模型：通过度量名称和键值对标识的时间序列数据
- (2) PromSQL：一种灵活的查询语言，可以利用多维数据完成复杂的查询
- (3) 不依赖分布式存储，单个服务器节点可直接工作



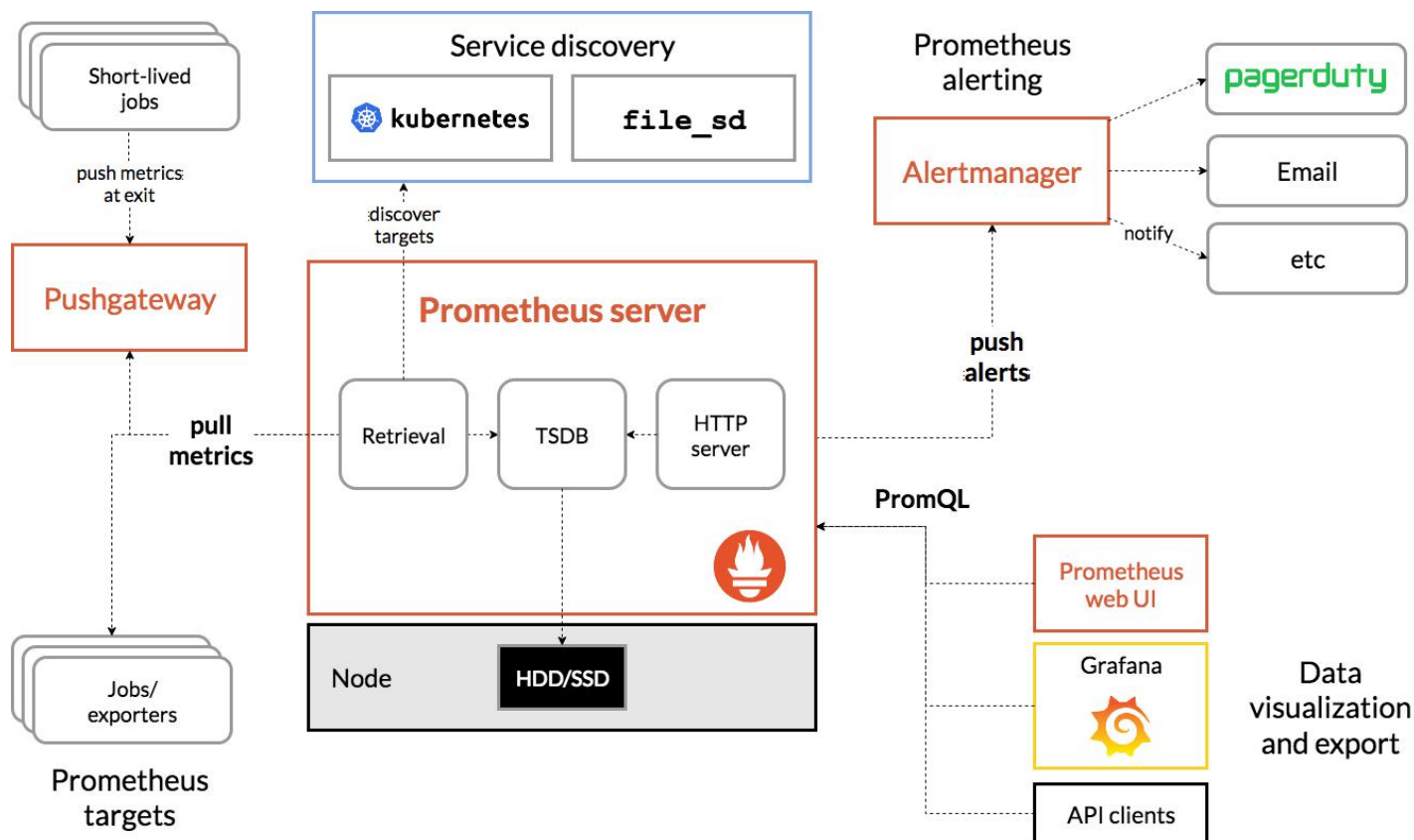
- (4) 基于HTTP的pull方式采集时间序列数据
- (5) 推送时间序列数据通过PushGateway组件支持
- (6) 通过服务发现或静态配置发现目标
- (7) 多种图形模式及仪表盘支持

组件:

- (1) Prometheus生态包括了很多组件，它们中的一些是可选的:
- (2) Prometheus主服务器，用于抓取和存储时间序列数据
- (3) 用于检测应用程序代码的客户端库
- (4) 用于支持短声明周期的push网关
- (5) 针对HAProxy, StatsD, Graphite, Snmp等服务的特定exporters
- (6) 警告管理器
- (7) 各种支持工具

多数Prometheus组件是Go语言写的，这使得这些组件很容易编译和部署。

## 4.2 架构设计

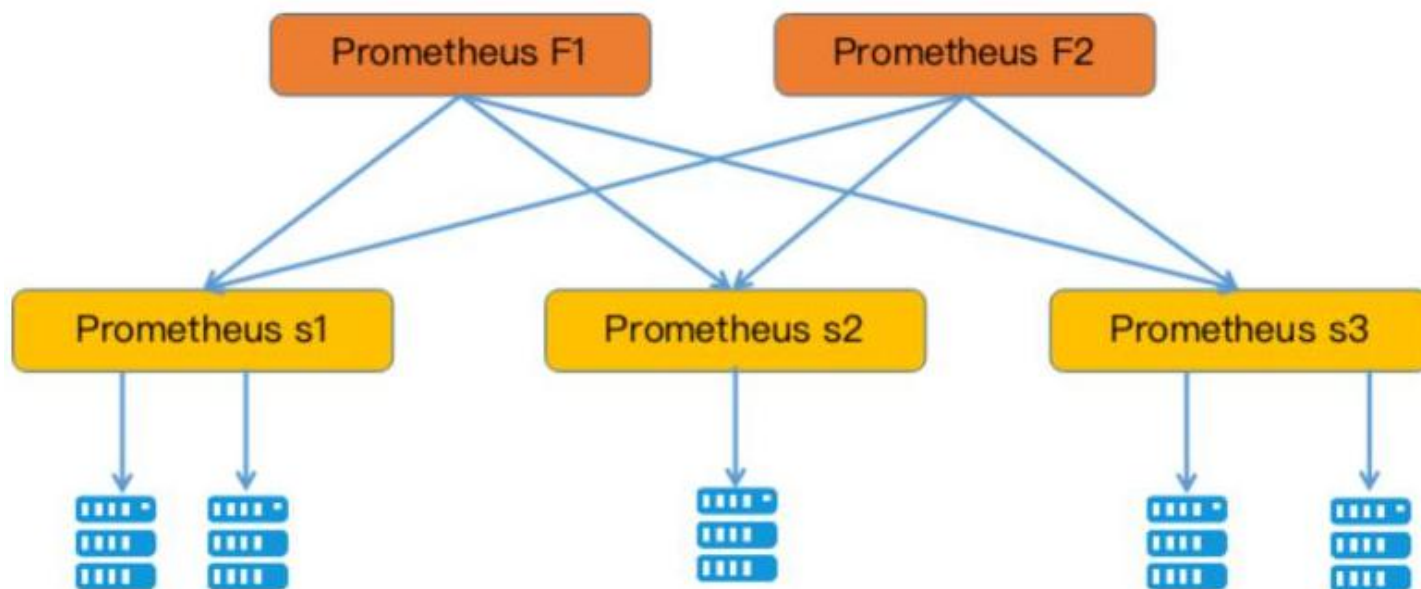


(1) 图片左侧是各种数据源主要是各种符合Prometheus数据格式的exporter，除此之外为了支持推动数据类型的Agent，可以通过Pushgateway组件，将Push转化为Pull。Prometheus甚至可以从其它的Prometheus获取数据，组建联邦集群。Prometheus的基本原理是通过HTTP周期性抓取被监控组件的状态，任意组件只要提供对应的HTTP接口并且符合Prometheus定义的数据格式，就可以接入Prometheus监控。

(2) 图片上侧是服务发现，Prometheus支持监控对象的自动发现机制，从而可以动态获取监控对象。

(3) 图片中间是Prometheus Server, Retrieval模块定时拉取数据, 并通过Storage模块保存数据。PromQL为Prometheus提供的查询语法, PromQL模块通过解析语法树, 调用Storage模块查询接口获取监控数据。

(4) 图片右侧是告警和页面展现, Prometheus将告警推送到alertmanger, 然后通过alertmanger对告警进行处理并执行相应动作。数据展现除了Prometheus自带的webui, 还可以通过grafana等组件查询Prometheus监控数据。



Prometheus支持使用联邦集群的方式, 对Prometheus进行扩展。如图, 在每个集群(数据中心)部署单独的Prometheus, 用于采集当前集群(数据中心)监控数据, 并由上层的Prometheus负责聚合多个集群(数据中心)的监控数据。这里部署多个联邦节点是为了实现高可用。

联邦集群的核心在于每一个Prometheus都包含一个用于获取当前实例中监控样本的接口/federate。对于联邦Prometheus而言, 无论是从其他的Prometheus实例还是Exporter实例中获取数据实际上并没有任何差异。

适用场景:

Prometheus在记录纯数字时间序列方面表现非常好。既适用于面向服务器等硬件指标的监控, 也适用于高动态的面向服务架构的监控。对于现流行的微服务, Prometheus的多维度数据收集和数据筛选查询语言也是非常的强大。Prometheus是为服务的可靠性而设计的, 当服务出现故障时, 可以快速定位和诊断问题。搭建过程对硬件和服务没有很强的依赖关系。

不适用场景:

(1) 如果需要100%的准确度(例如按请求计费), Prometheus不是一个好的选择, 因为收集的数据可能不够详细和完整。在这种情况下, 最好使用其他系统来收集和分析数据以进行计费, 并使用Prometheus进行其余监控。

(2) Prometheus只针对性能和可用性监控, 默认并不具备日志监控等功能。

(3) Prometheus认为只有最近的监控数据才有查询的需要, 本地存储的设计初衷只是保持短期(一个月)的数据, 并非针对大量的历史数据的存储。如果需要报表之类的历史数据, 则建议使用远端存储。

(4) Prometheus没有定义单位, 需要使用者自己去区分或者事先定义好监控数据单位。

## 4.3 监控点有哪些

从容器云平台本身的业务需求分析来看，至少应该通过Prometheus获取到以下监控数据：性能指标：

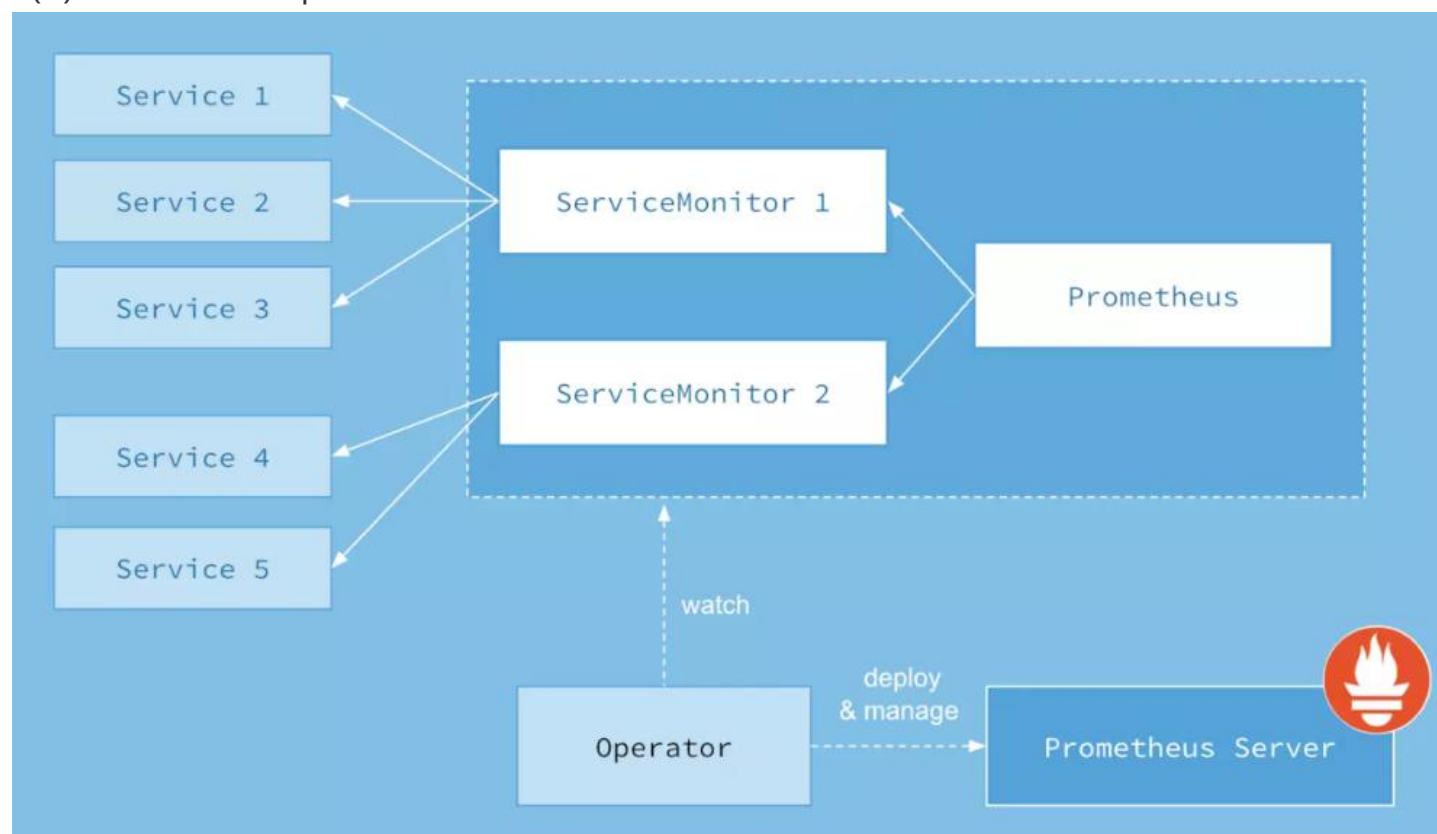
- (1) 容器相关的性能指标数据
- (2) Pod相关的性能指标数据
- (3) 主机Node节点相关的性能指标数据

服务健康状态：

- (1) Kubernetes集群服务的健康状态
- (2) Kubernetes服务组件的健康状态
- (3) 主机Node节点的健康状态
- (4) docker服务进程的健康状态
- (5) Deployment相关的健康状态
- (6) Pod的健康状态

## 4.4 重要组件介绍

### (1) Prometheus-Operator



上图是Prometheus-Operator官方提供的架构图，其中Operator是最核心的部分，作为一个控制器，它会去创建Prometheus、ServiceMonitor、AlertManager以及PrometheusRule4个CRD资源对象，然后会一直监控并维持这4个资源对象的状态。

- ①Prometheus：Prometheus Deployment定义
- ②ServiceMonitor：Prometheus监控对象的定义
- ③Alertmanager：Alertmanager Deployment定义

#### ④PrometheusRule：告警规则

这样，我们在集群中监控数据就变成了直接去操作 Kubernetes 集群的资源对象，方便很多了。一个 ServiceMonitor 可以通过 labelSelector 的方式去匹配一类 Service，Prometheus 也可以通过 labelSelector 去匹配多个ServiceMonitor。

#### (2) Prometheus Server

Prometheus Server是Prometheus组件中的核心部分，负责实现对监控数据的获取，存储以及查询。Prometheus Server可以通过静态配置管理监控目标，也可以配合使用Service Discovery的方式动态管理监控目标，并从这些监控目标中获取数据。其次Prometheus Server需要对采集到的监控数据进行存储，其本身就是一个时序数据库，将采集到的监控数据按照时间序列的方式存储在本地磁盘当中。Prometheus Server对外提供了自定义的PromQL语言，实现对数据的查询以及分析。

Prometheus Server内置的Express Browser UI，通过这个UI可以直接通过PromQL实现数据的查询以及可视化。Prometheus Server的联邦集群能力可以使其从其他的Prometheus Server实例中获取数据，因此在大规模监控的情况下，可以通过联邦集群以及功能分区的方式对Prometheus Server进行扩展。

Prometheus支持两种类型的规则：记录规则和警报规则。要在Prometheus中包含规则，需要创建一个包含必要规则语句的文件，并让Prometheus通过Prometheus配置文件中的rule\_files字段加载规则文件。

##### ①Recording rules（记录规则）

Recording rules可以预先计算经常需要或计算量大的表达式，并将其结果保存为一组新的时间序列。这样，查询预先计算的结果通常比每次需要原始表达式查询要快得多。这对于dashboards特别有用，dashboards每次刷新时都需要重复查询相同的表达式。

##### ②Alerting rules（告警规则）

告警规则可以基于Prometheus表达式定义警报条件，并将有关触发告警的通知发送到外部服务。每当告警表达式在给定时间点生成一个或多个向量元素时，警报将计为这些元素的标签集的活动状态。

并且Prometheus 提供了多种服务发现方式：

- azure\_sd\_configs
- consul\_sd\_configs
- dns\_sd\_configs
- ec2\_sd\_configs
- openstack\_sd\_configs
- file\_sd\_configs
- kubernetes\_sd\_configs
- marathon\_sd\_configs
- nerve\_sd\_configs
- serverset\_sd\_configs
- triton\_sd\_configs

#### (3) PushGateway

由于Prometheus数据采集基于Pull模型进行设计，因此在网络环境的配置上必须要让Prometheus Server能够直接与Exporter进行通信。当这种网络需求无法直接满足时，就可以利用PushGateway来进行中转。可以通过PushGateway将内部网络的监控数据主动Push到Gateway当中。而Prometheus



Server则可以采用同样Pull的方式从PushGateway中获取到监控数据。

#### (4) Exporters

Exporter将监控数据采集的端点通过HTTP服务的形式暴露给Prometheus Server, Prometheus Server通过访问该Exporter提供的Endpoint端点, 即可获取到需要采集的监控数据。一般来说可以将Exporter分为2类:

①直接采集: 这一类Exporter直接内置了对Prometheus监控的支持, 比如cAdvisor, Kubernetes, Etcd等, 都直接内置了用于向Prometheus暴露监控数据的端点。

②间接采集: 间接采集, 原有监控目标并不直接支持Prometheus, 因此我们需要通过Prometheus提供的Client Library编写该监控目标的监控采集程序。例如: Mysql Exporter, JMX Exporter, Blackbox Exporter等。

#### (5) AlertManager

在Prometheus Server中支持基于PromQL创建告警规则, 如果满足PromQL定义的规则, 则会产生一条告警, 而告警的后续处理流程则由AlertManager进行管理。在AlertManager中我们可以与邮件, Slack等等内置的通知方式进行集成, 也可以通过Webhook自定义告警处理方式。AlertManager即Prometheus体系中的告警处理中心。

主要处理流程:

①接收到Alert, 根据labels判断属于哪些Route (可存在多个Route, 一个Route有多个Group, 一个Group有多个Alert) 。

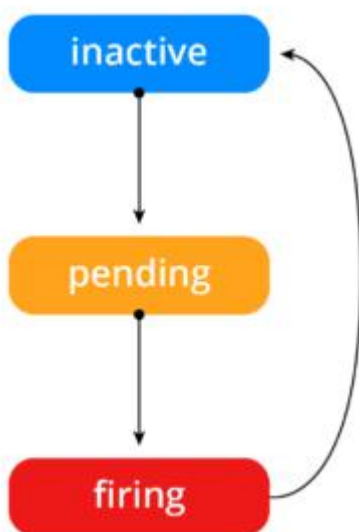
②将Alert分配到Group中, 没有则新建Group。

③新的Group等待group\_wait指定的时间 (等待时可能收到同一Group的Alert) , 根据resolve\_timeout判断Alert是否解决, 然后发送通知。

④已有的Group等待group\_interval指定的时间, 判断Alert是否解决, 当上次发送通知到现在的间隔大于repeat\_interval或者Group有更新时会发送通知。

alert工作流程:

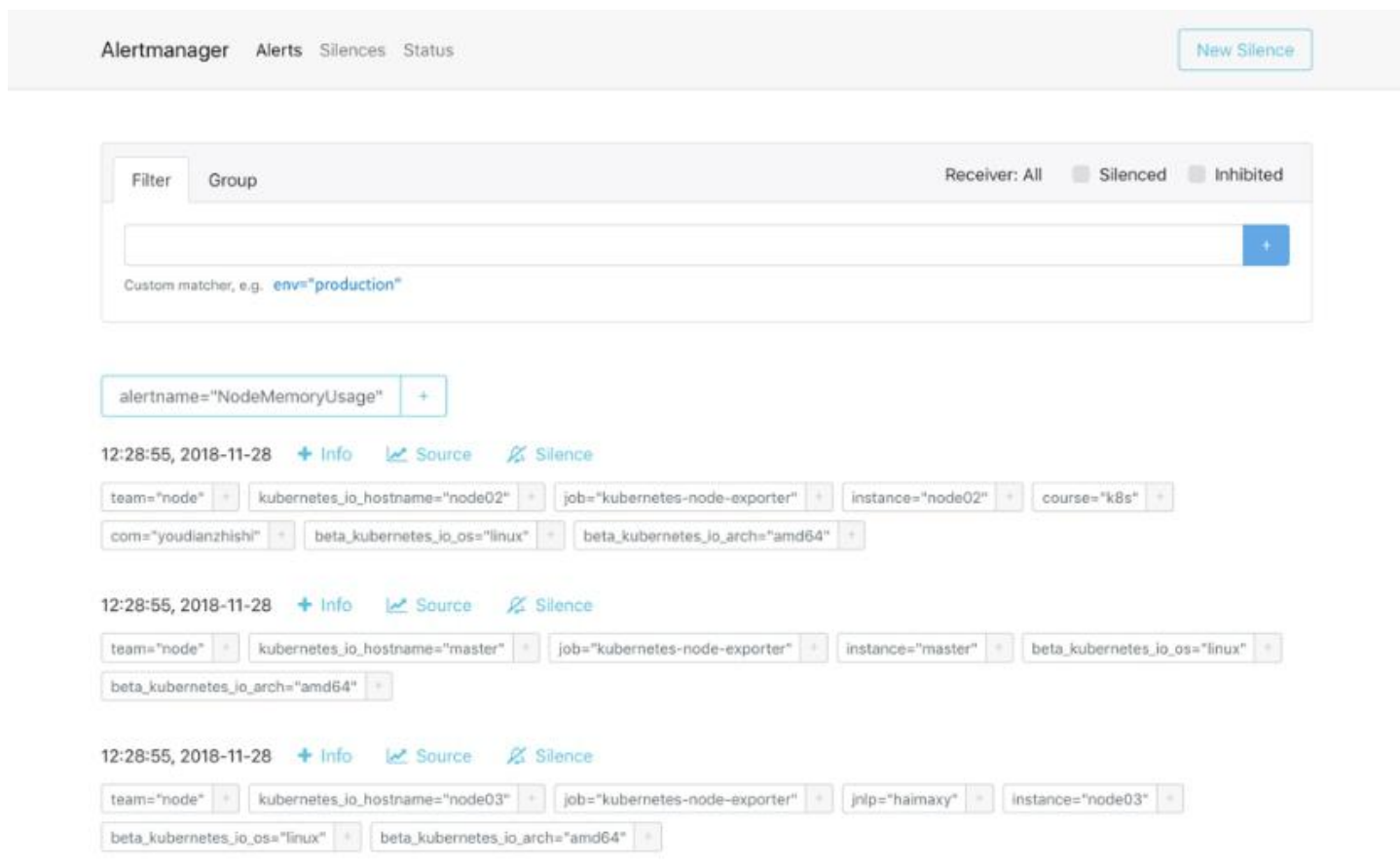
一旦这些警报存储在Alertmanager, 它们可能处于以下任何状态:



inactive: 表示当前报警信息既不是firing状态也不是pending状态。

pending: 表示在设置的阈值时间范围内被激活了。

firing: 表示超过设置的阈值时间被激活了。



在这个页面中可以进行一些操作，比如过滤、分组等等，里面还有两个新的概念：Inhibition(抑制)和 Silences(静默)。

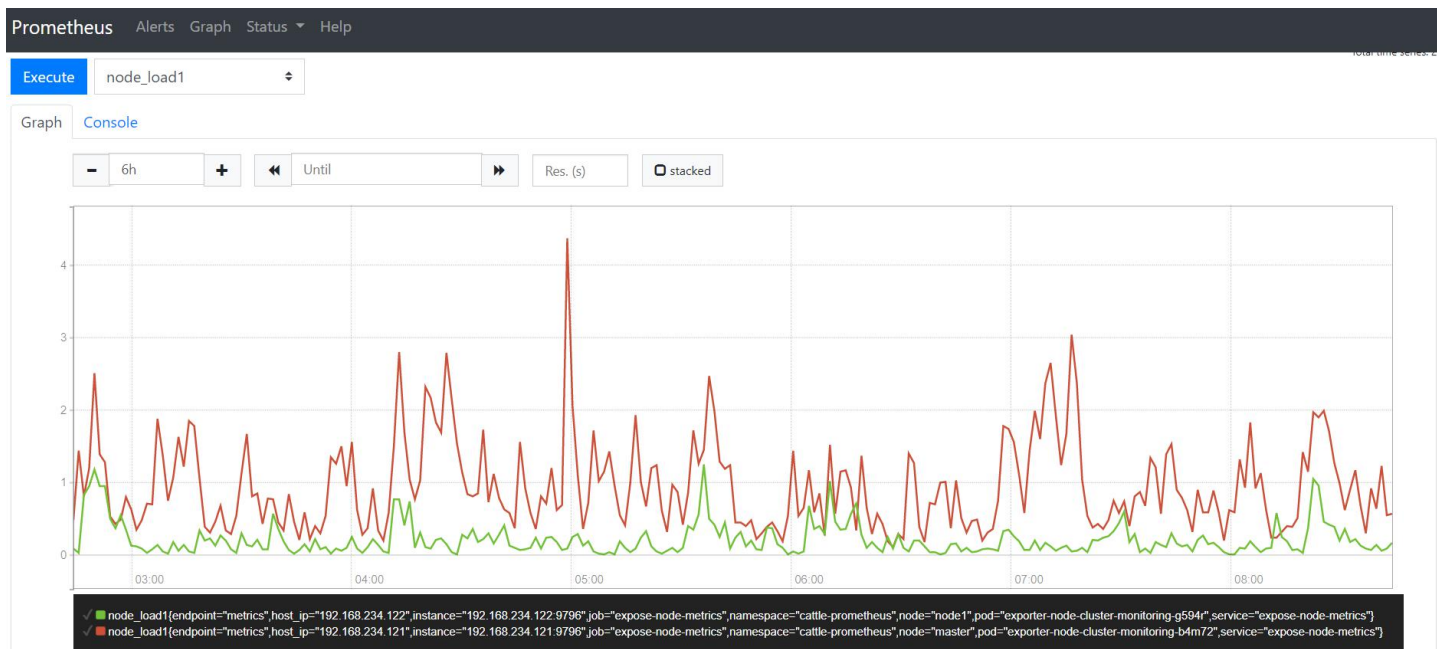
**Inhibition:** 如果某些其他警报已经触发了，则对于某些警报，Inhibition 是一个抑制通知的概念。例如：一个警报已经触发，它正在通知整个集群是不可达的时，Alertmanager 则可以配置成关心这个集群的其他警报无效。这可以防止与实际无关的数百或数千个触发警报的通知，Inhibition 需要通过上面的配置文件进行配置。

**Silences:** 静默是一个非常简单的方法，可以在给定时间内简单地忽略所有警报。Silences 基于 matchers 配置，类似路由树。来到的警告将会被检查，判断它们是否和活跃的 Silences 相等或者正则表达式匹配。如果匹配成功，则不会将这些警报发送给接收者。

## 4.5 数据可视化

### (1) Web Console

Prometheus 自带了 Web Console，安装成功后可以访问 <http://localhost:9090/graph> 页面，用它可以进行任何 PromQL 查询和调试工作，非常方便，例如：



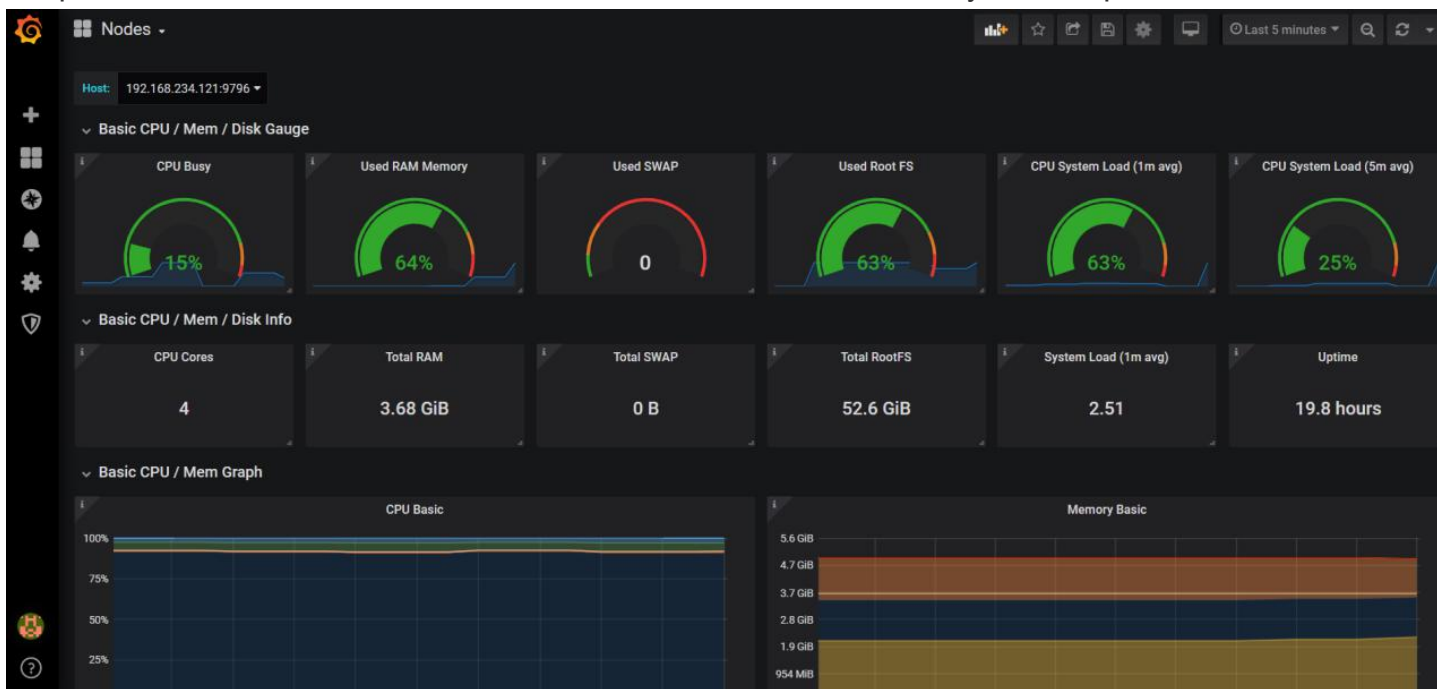
其中：

- ①alters可以查看当前报警的状态
- ②status->rules可以查看配置的报警规则
- ③status->targets可以查看配置的job及状态

通过上图不难发现，Prometheus 自带的 Web 界面比较简单，因为它的目的是为了及时查询数据，方便 PromQL 调试。

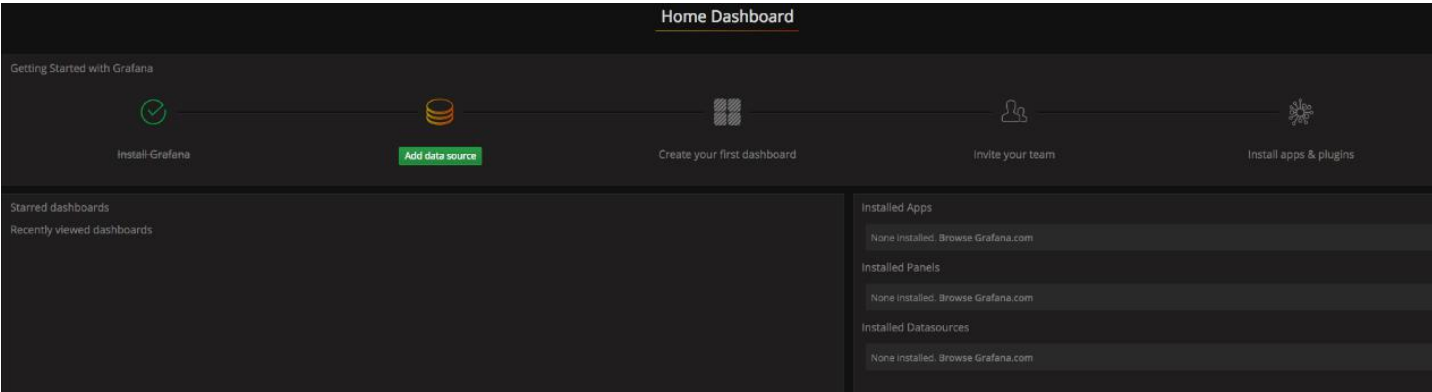
## (2) Grafana

grafana主要用于大规模指标数据的可视化展现，是网络架构和应用分析中最流行的时序数据展示工具，目前已经支持绝大部分常用的时序数据库。Grafana支持许多不同的数据源，每个数据源都有一个特定的查询编辑器，该编辑器定制的特性和功能是公开的特定数据来源。官方支持以下数据源：Graphite、Elasticsearch、InfluxDB、Prometheus、Cloudwatch、MySQL和OpenTSDB等。

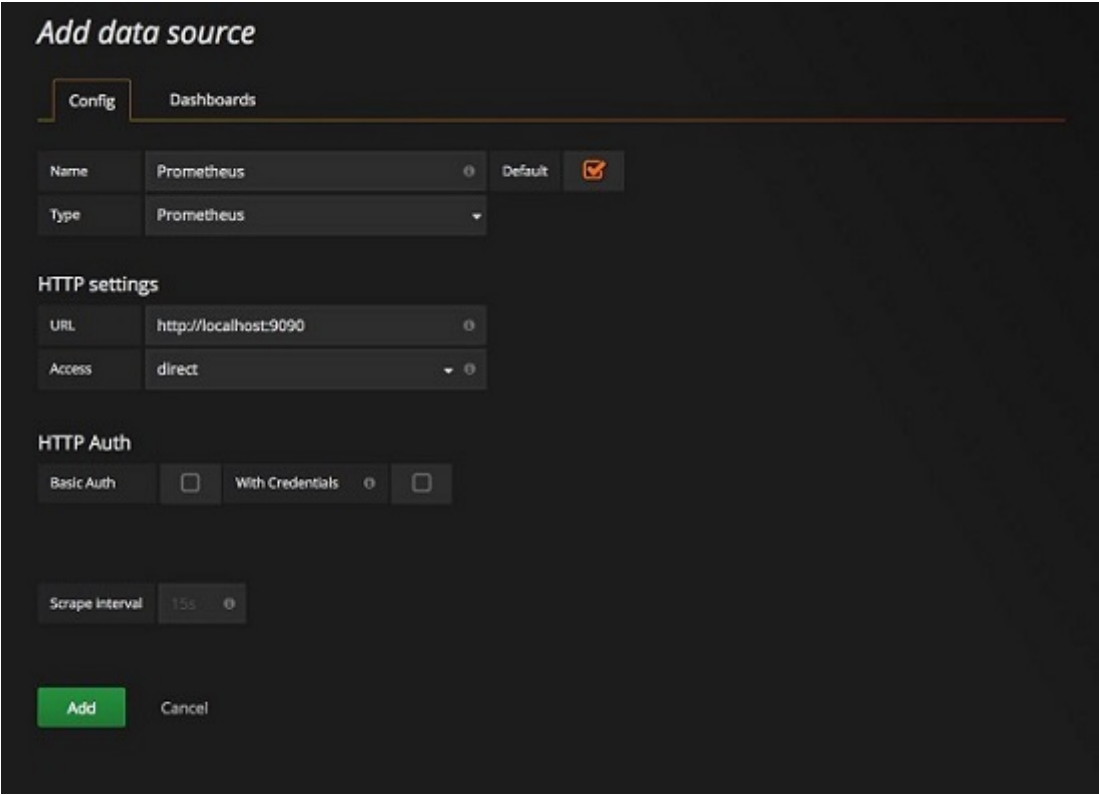


进入到Grafana的界面中，默认情况下使用账户admin/admin进行登录。在Grafana首页中显示默

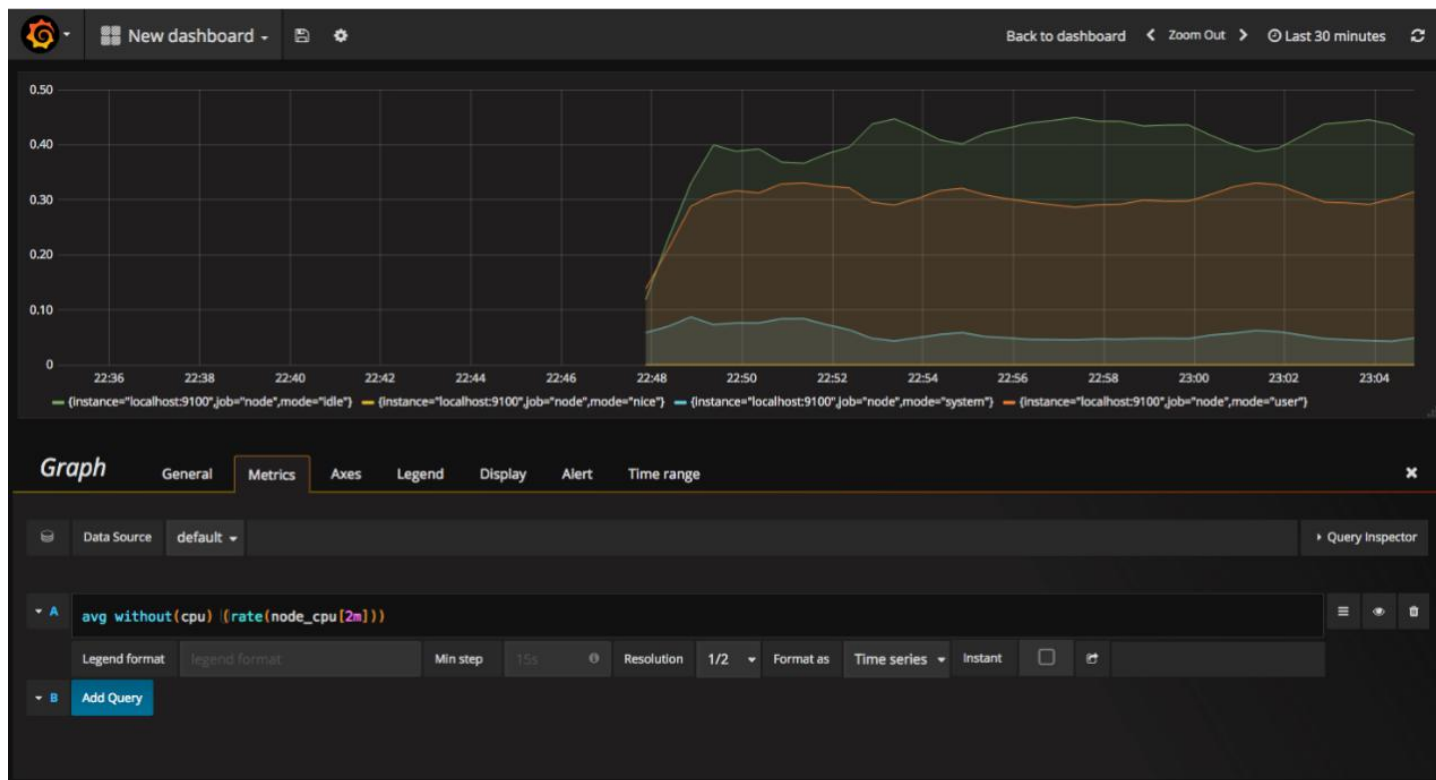
认的使用向导，包括：安装、添加数据源、创建Dashboard、邀请成员、以及安装应用和插件等主要流程。



这里将添加Prometheus作为默认的数据源，如下图所示，指定数据源类型为Prometheus并且设置Prometheus的访问地址即可，在配置正确的情况下点击“Add”按钮，会提示连接成功的信息：



在完成数据源的添加之后就可以在Grafana中创建我们可视化Dashboard了。Grafana提供了对PromQL的完整支持，如下所示，通过Grafana添加Dashboard并且为该Dashboard添加一个类型为“Graph”的面板。并在该面板的“Metrics”选项下通过PromQL查询需要可视化的数据：



点击界面中的保存选项，就创建了我们的可视化Dashboard了。当然作为开源软件，Grafana社区鼓励用户通过<https://grafana.com/dashboards>网站分享Dashboard，可以找到大量可直接使用的Dashboard：

## Dashboards

Official & community built dashboards

Filter by:

Data Source: All

Panel Type: All

Category: All






Collector: All

Search within this list

Share your dashboards

Sign up for a free [Grafana.com](#) account and share your creations with the community.

[Sign Up](#)

	<b>1 Node Exporter for Prometheus Dashboard English Version UPDATE 1102</b> by StarsL.cn Support Node Exporter v0.16 and above.Optimize the main metrics display.Includes: CPU, memory,... PROMETHEUS NODEEXPORTER	Downloads: 6108 Reviews: 13 ★★★★★
	<b>1 Node Exporter for Prometheus Dashboard English Version UPDATE 1102</b> by Leonardo da Costa Support Node Exporter v0.16 and above.Optimize the main metrics display.Includes: CPU, memory,... PROMETHEUS	Downloads: 274 Reviews: 1 ★★★★★
	<b>1 Node Exporter for Prometheus Dashboard English Version UPDATE 1102</b> by yasty25 Support Node Exporter v0.16 and above.Optimize the main metrics display.Includes: CPU, memory,... PROMETHEUS ICINGA: CHECK_PING	Downloads: 29 Reviews: 0
	<b>1 Node Exporter for Prometheus Dashboard English Version UPDATE 1102</b> by liujingyuyves Support Node Exporter v0.16 and above.Optimize the main metrics display.Includes: CPU, memory,... PROMETHEUS	Downloads: 12 Reviews: 0
	<b>1 Node Exporter for Prometheus Dashboard 中文版 UPDATE 1102</b> by StarsL.cn 支持 Node Exporter v0.16 及以上的版本。精简优化重要指标展示。包含：CPU 内存 磁盘 IO 网络... PROMETHEUS NODEEXPORTER: CPU, DISKSTATS, FILESYSTEM, MEMINFO, NETDEV, NETSTAT, TCPSTAT, VMSTAT	Downloads: 20291 Reviews: 32 ★★★★★

Grafana中所有的Dashboard通过JSON进行共享，下载并且导入这些JSON文件，就可以直接使用这些已经定义好的Dashboard。



## 4.6 高可用设计

### 4.6.1 存储

Prometheus内置了一个基于本地存储的时间序列数据库。在Prometheus设计上，使用本地存储可以降低Prometheus部署和管理的复杂度同时减少高可用（HA）带来的复杂性。在默认情况下，用户只需要部署多套Prometheus，采集相同的Targets即可实现基本的HA。同时由于Prometheus高效的数据处理能力，单个Prometheus Server基本上能够应对大部分用户监控规模的需求。

#### (1) 本地存储

通过Prometheus自带的tsdb（时序数据库），将数据保存到本地磁盘，为了性能考虑，建议使用SSD。Prometheus本地存储经过多年改进，自Prometheus 2.0 后提供的V3版本tsdb性能已经非常高。

本地存储也带来了一些不好的地方，首先就是数据持久化的问题，特别是在像Kubernetes这样的动态集群环境下，如果Prometheus的实例被重新调度，那所有历史监控数据都会丢失。其次本地存储也意味着Prometheus不适合保存大量历史数据(一般Prometheus推荐只保留几周或者几个月的数据)。最后本地存储也导致Prometheus无法进行弹性扩展。为了适应这方面的需求，Prometheus提供了remote\_write和remote\_read的特性，支持将数据存储到远端和从远端读取数据。通过将监控与数据分离，Prometheus能够更好地进行弹性扩展。

#### (2) 远端存储

Prometheus提供了remote\_write和remote\_read的特性，支持将数据存储到远端和从远端读取数据。通过将监控样本采集和数据存储分离，解决Prometheus的持久化问题，适用于大量历史监控数据的存储和查询。目前，远端存储主要包括OpenTSDB、InfluxDB、Elasticsearch、M3db等。

Remote Write：用户可以在Prometheus配置文件中指定Remote Write(远程写)的URL地址，一旦设置了该配置项，Prometheus将采集到的样本数据通过HTTP的形式发送给适配器(Adapter)。而用户则可以在适配器中对接外部任意的服务。外部服务可以是真正的存储系统，公有云的存储服务，也可以是消息队列等任意形式。



Remote Read：Prometheus的Remote Read(远程读)也通过了一个适配器实现。在远程读的流程当中，当用户发起查询请求后，Prometheus将向remote\_read中配置的URL发起查询请求(matchers,ranges)，Adaptor根据请求条件从第三方存储服务中获取响应的数据。同时将数据转换为Prometheus的原始样本数据返回给Prometheus Server。当获取到样本数据后，Prometheus在本地使用PromQL对样本数据进行二次处理。（注意：启用远程读设置后，只在数据查询时有效，对于规则文件的处理，以及Metadata API的处理都只基于Prometheus本地存储完成。）



通过远端存储可以分离监控样本采集和数据存储，解决Prometheus的持久化问题。

## 4.6.2 Prometheus高可用

### (1) 服务高可用

Prometheus以pull方式设计，通过主动发起的方式采集监控Metrics。Prometheus内置了一个基于本地存储机制的时间序列数据库TSDB，使用本地存储的方式降低了部署和管理Prometheus的复杂度。为了保证Prometheus服务的可用性，用户只需要部署多套Prometheus Server实例，先通过Nginx/HA Proxy 接收请求，然后通过Prometheus采集相同的Exporter目标，就可以实现基本的高可用。

### (2) 数据一致性

Pull模式下，同一个服务至少需要两个Prometheus节点同时拉取监控数据。这种情况下，就需要解决多个Prometheus Server实例之间的数据一致性问题。

解决方案：采用Prometheus的协助方案Thanos。同机部署Thanos Sidecar与Prometheus，Thanos Query调用查询的载体Sidecar，Sidecar与prometheus进行交互，获取Metrics并去重，来保证数据一致性。Thanos是CNCF孵化项目，目前沒有出稳定发行版，未来可能是Prometheus高可用官方解决方案，还需继续关注。

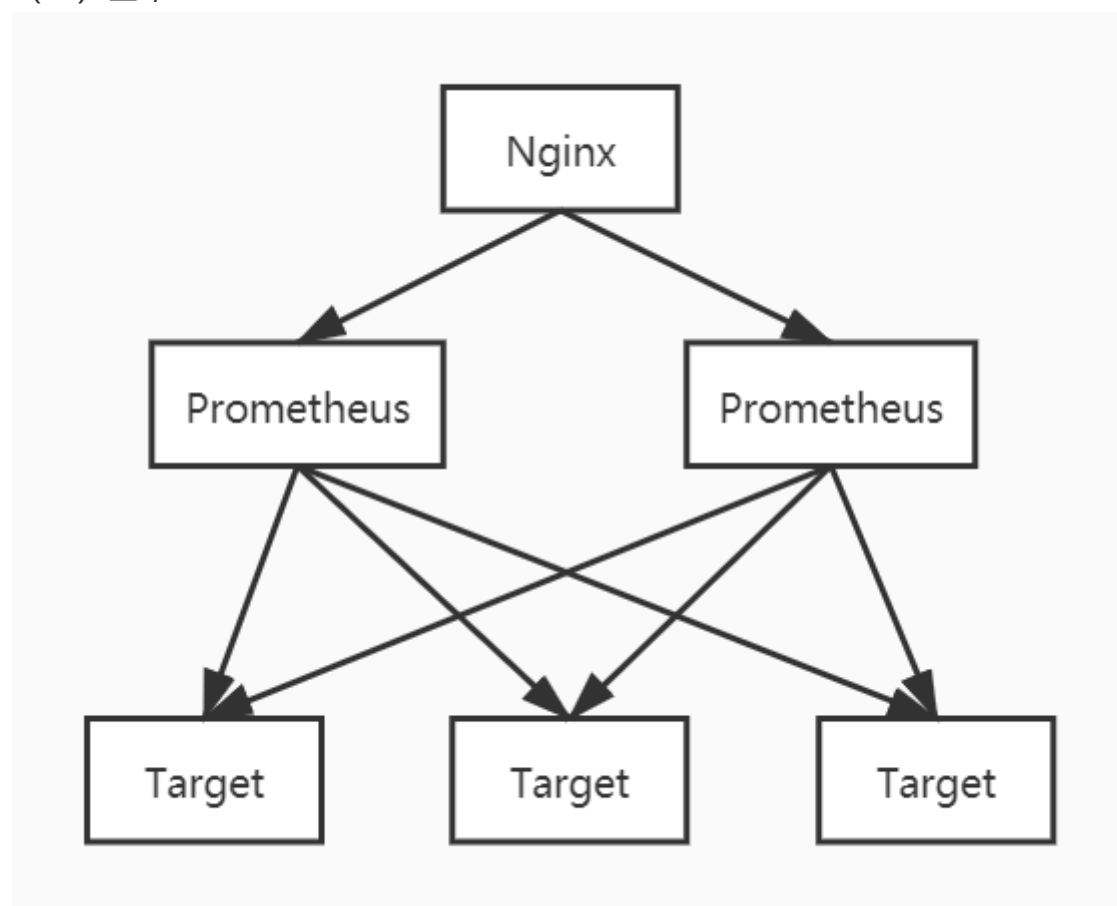
### (3) 水平可扩展

当单台Promthues Server无法处理大量的采集任务时，通过联邦集群的特性对Prometheus采集任务按照功能分区，对Promthues进行扩展，以适应规模的变化。

### (4) 数据持久化

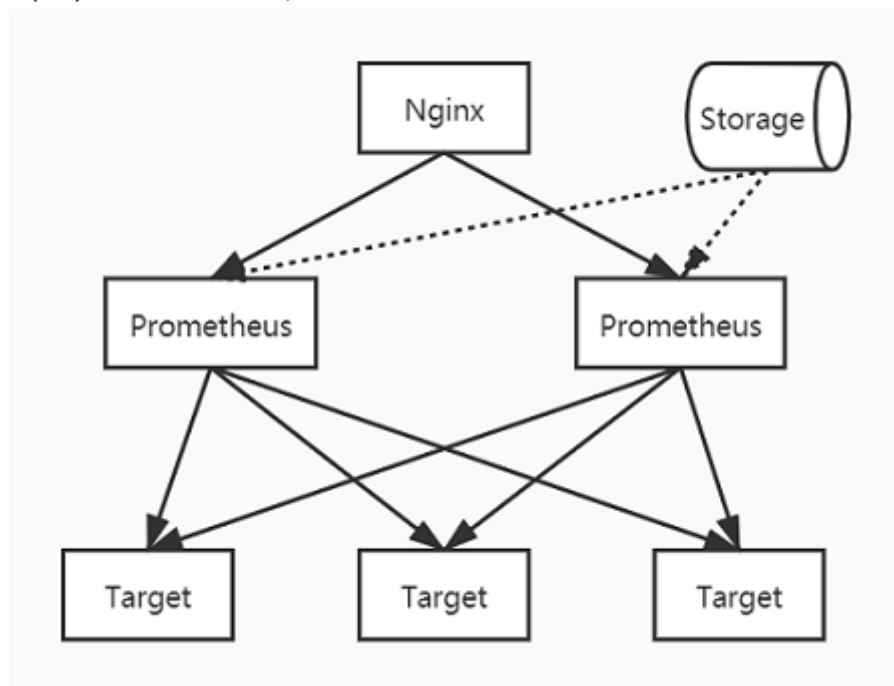
Prometheus的本地存储设计能够满足大部分监控规模的需求，但是本地存储无法存储大量历史数据，存在数据持久化问题。Prometheus允许用户通过接口将数据保存到第三方数据库中，这种方式在Promthues中称为远程存储。

#### (一) 基本HA



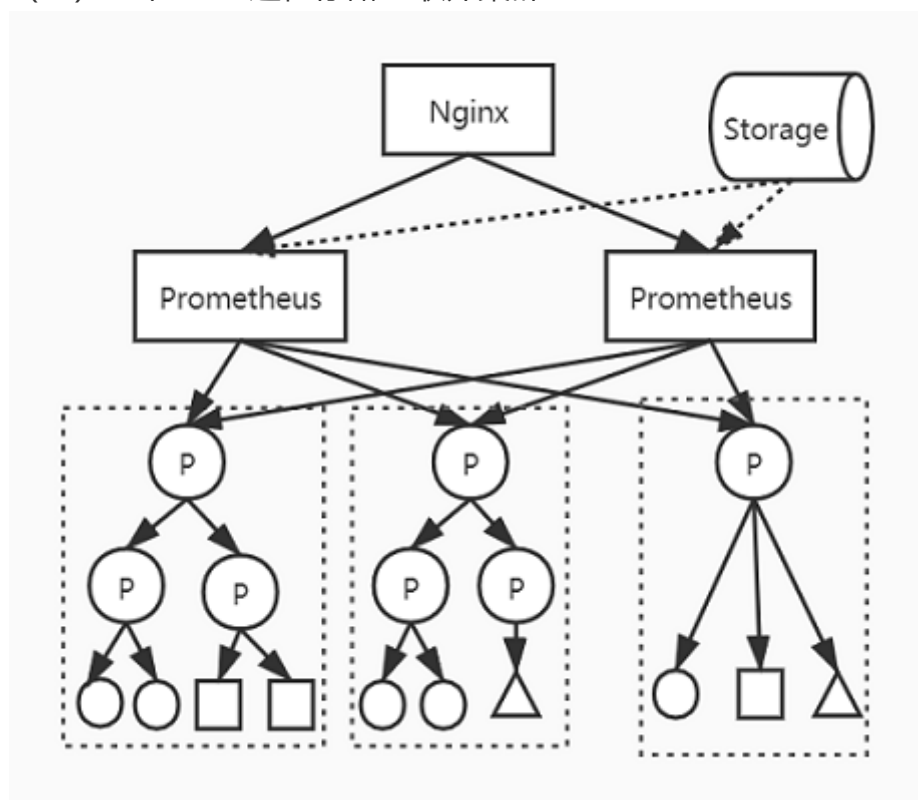
基本HA可以保证服务的高可用，无法长期存储历史数据。监控系统关注一段时间某个监控指标是否达到告警状态，多个Prometheus Server之间监控指标数据细微的差别，对Promethues是否产生告警通知影响很小，所以基本HA架构可以满足一般的监控场景。

## (二) 基本HA + 远程存储



在基本HA保证Promthues服务可用性的基础上，通过添加远程存储支持，确保了数据的持久化。基本HA和远程存储的方案适合要求监控数据持久化，保证Promthues Server可迁移的场景。

## (三) 基本HA + 远程存储 + 联邦集群



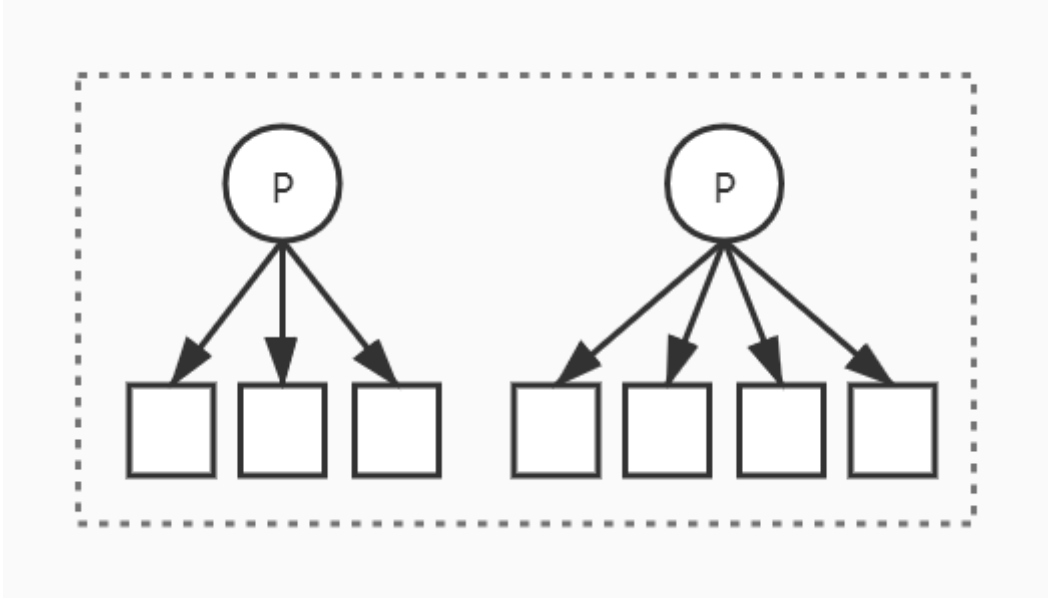
基本HA+远程存储+联邦集群的方案保证Promthues的服务高可用、数据持久化、水平可扩展。这种架构方案一般适合两个使用场景：

单数据中心和大量采集任务的场景，Promthues处理大量采集任务时存在性能瓶颈。采用Prometheus联邦集群对采集任务进行功能分区，将不同类型的采集任务划分到不同的Promthues中。

多数据中心的场景，最上层的Promthues Server无法直接从不同数据中心的Exporter拉取数据。采用联邦集群进行分层处理，每个数据中心部署一组Prometheus Server收集该数据中心的Metric，再由上层的Prometheus拉取监控指标。

#### (四) 按照实例进行功能分区

当单个采集任务的Target数目非常巨大，通过联邦集群简单进行功能分区，Prometheus Server也无法有效处理采集任务时，就要考虑在实例级别对采集任务进行功能划分了。通过将同一任务的不同实例划分到不同的Prometheus Server上。



设置Prometheus的relabel\_config，确保当前Prometheus Server只收集当前采集任务的一部分实例的Metric。

```
global:
  external_labels:
    slave: 1 # This is the 2nd slave. This prevents clashes between slaves.
scrape_configs:
- job_name: some_job
  # Add usual service discovery here, such as static_configs
  relabel_configs:
  - source_labels: [__address__]
    modulus: 4 # 4 slaves
    target_label: __tmp_hash
    action: hashmod
  - source_labels: [__tmp_hash]
    regex: ^1$ # This is the 2nd slave
    action: keep
```

并且通过当前数据中心的一个中心Prometheus Server将实例级监控Metric聚合到任务级别。

```
- scrape_config:
- job_name: slaves
  honor_labels: true
  metrics_path: /federate
  params:
    match[]:
      - '{__name__=~"^slave:.*"}' # Request all slave-level time series
static_configs:
- targets:
  - slave0:9090
  - slave1:9090
  - slave3:9090
  - slave4:9090
```

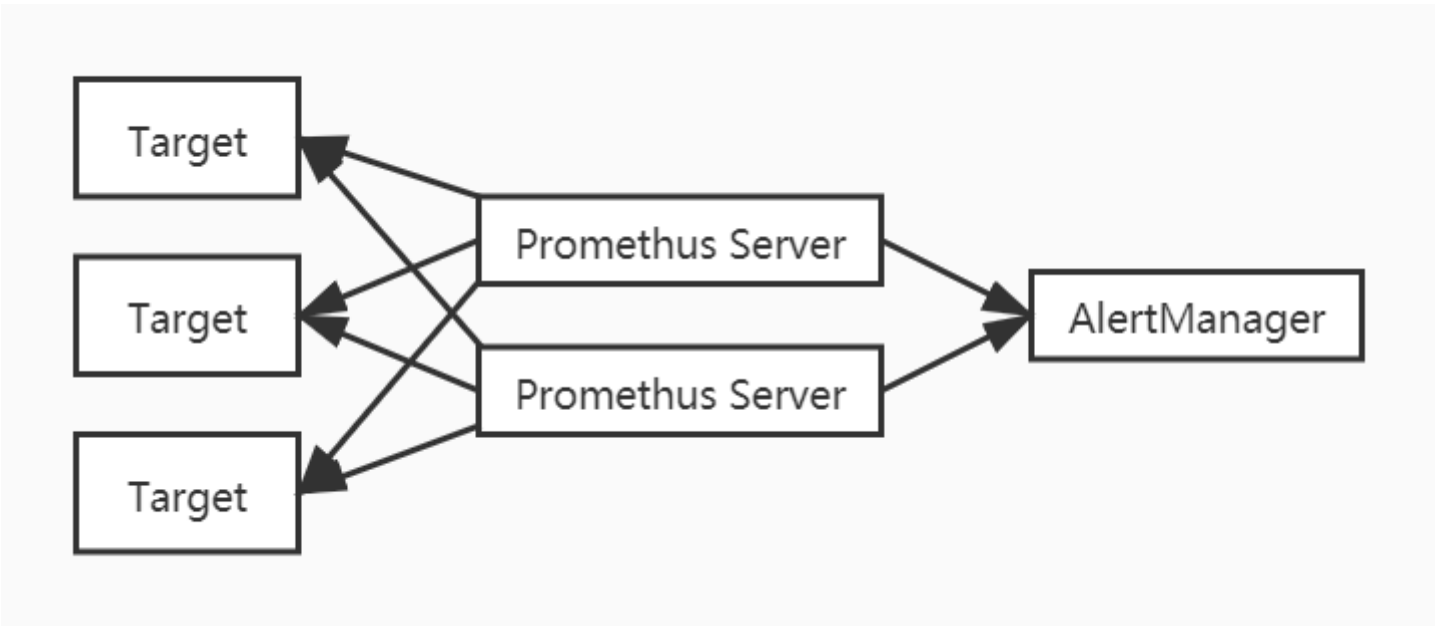
高可用方案的选择

下表展示了三种Promthues高可用方案各自解决的问题，用户可以根据自己的实际场景选择高可用部署方案。

方案	服务可用	数据持久化	水平扩展
基本HA	v	x	x
基本HA+远程存储	v	v	x
基本HA+远程存储+联邦集群	v	v	v

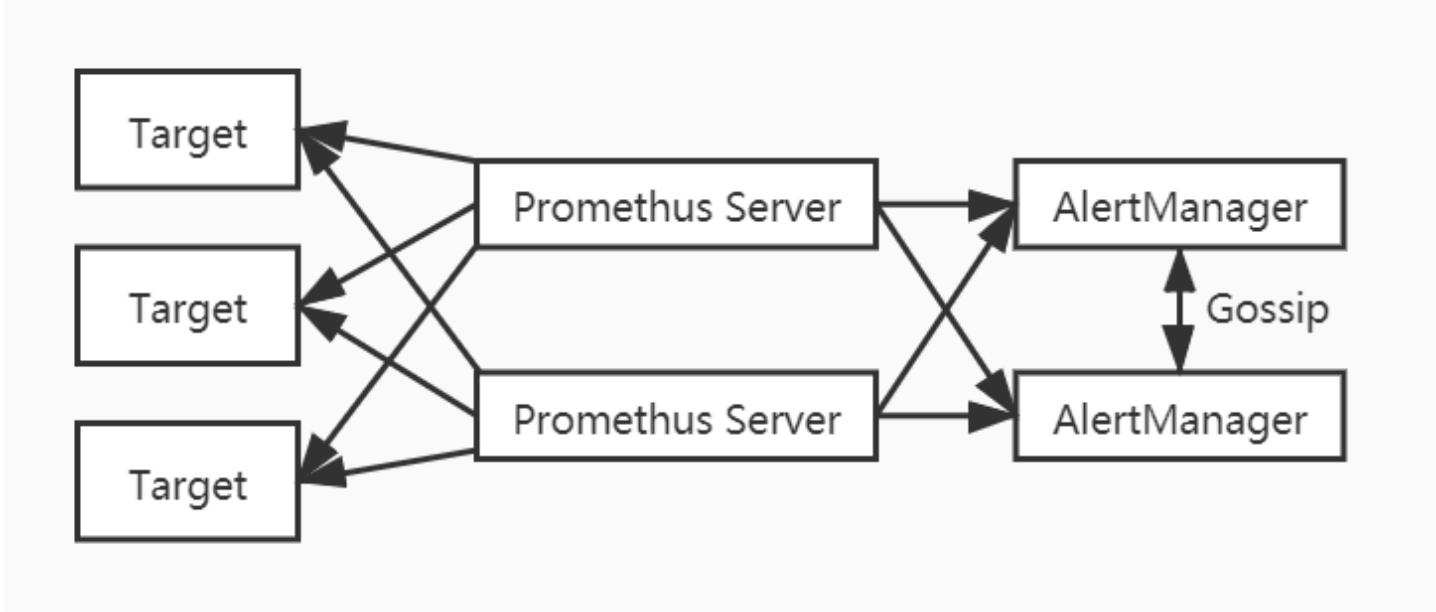
AlertManager的高可用

通常用户会部署两个或两个以上的Promtheus Server，来保证Promtheus服务可用性。这些Promtheus Server具有相同的Job配置，告警配置等。AlertManager可以同时接收多个相同配置的Promtheus Server产生的告警，基于告警去重机制做告警处理。但是单个AlertManager不能保证单点失效时告警处理的可用性，所以我们打算部署多套AlertManager。





由于多个AlertManager之间不了解彼此的存在，就会导致同一个告警通知被不同的AlertManager重复发送多次。为了解决这一问题，AlertManager引入Gossip共识机制。例如两个AlertManager接收到相同告警时，基于Gossip共识机制，只允许一个AlertManager将这个告警通知发送给Receiver。



## 4.7 性能优化与容量预估

### 4.7.1 Prometheus性能优化

#### 使用固态硬盘部署Prometheus

相比机械硬盘，固态硬盘在读写性能方面具有一定的优势。考虑将Prometheus Server部署在固态硬盘上，目的是提升吞吐量TPS，提高Prometheus性能。

#### 使用Recoding Rules优化性能

使用PromQL可以对Prometheus中的监控样本数据进行查询，聚合以及逻辑运算。每次查询Metric，直接使用复杂且计算量较大的PromQL时，可能会导致Prometheus的响应超时。这种情况下，Prometheus通过Recoding Rule定义这种计算规则，并保存记录Record。每次查询同一个Metric，直接读取已记录的Metric，实现了对复杂查询的性能优化。

#### 定义Recoding rules

根据规则中的定义，Prometheus会在后台完成expr中定义的PromQL表达式计算，并且将计算结果保存到新的时间序列record中。同时还可以通过labels为这些样本添加额外的标签。

```
# 指出规则类型 record后面接规则名称
record: <string>

# 写入PromQL表达式查询语句
expr: <string>

# 在存储数据之前加上标签
labels:
  [ <labelname>: <labelvalue> ]
```

## 示例

比如我们使用 `container_cpu_usage_seconds_total` 查询Kubernetes节点之间CPU利用率，记录Load time。CPU利用率查询表达式：

```
sum(rate(container_cpu_usage_seconds_total[5m])) /  
avg_over_time(sum(kube_node_status_allocatable_cpu_cores)[5m:5m])
```

使用 `container_memory_usage_bytes` 查询Kubernetes节点之间内存利用率，记录Load time。内存利用率查询表达式：

```
avg_over_time(sum(container_memory_usage_bytes)[15m:15m]) /  
avg_over_time(sum(kube_node_status_allocatable_memory_bytes)[5m:5m])
```

## 定义规则优化CPU和内存利用率的查询

```
groups:  
- name: k8s.rules  
  rules:  
  - expr: |  
      sum(rate(container_cpu_usage_seconds_total{image!="", container!=""}[5m])) by (namespace)  
      record: namespace:container_cpu_usage_seconds_total:sum_rate  
  - expr: |  
      sum(container_memory_usage_bytes{image!="", container!=""}) by (namespace)  
      record: namespace:container_memory_usage_bytes:sum
```

现在，我们将查询表达式更改为如下表达式，分别得出CPU和内存利用率，记录Load time。CPU利用率查询表达式：

```
sum(namespace:container_cpu_usage_seconds_total:sum_rate) /  
avg_over_time(sum(kube_node_status_allocatable_cpu_cores)[5m:5m])
```

内存利用率查询表达式：

```
sum(namespace:container_memory_usage_bytes:sum) /  
avg_over_time(sum(kube_node_status_allocatable_memory_bytes)[5m:5m])
```

通过对比优化查询前后的Load time，可以计算Recoding rules规则优化查询的效率。

## 4.7.2 容量预估

在部署Prometheus Server之前，对Prometheus Server的存储用量进行预估十分必要。否则，运维人员对业务数据的规模和所需存储资源的量级无法取得直观的认识。分配的存储资源过多，将会导致资源闲置和成本浪费，分配的存储容量不足，则系统无法应对业务的增长，严重时影响到监控服务的稳定性。

## 预估内存用量

表1.基于集群中节点/Pods数目的Prometheus内存用量要求

节点数目	Pods数目	RAM空间(按scale大小)
50	1800	6GB
100	3600	10GB
150	5400	12GB
200	7200	14GB

## 预估硬盘容量

表2.基于集群中节点/Pods数目的Prometheus数据库存储要求

节点数目	Pods数目	每天的存储量增长	每15天的存储量增长	网络(每个tsdb chunk)
50	1800	6.3 GB	94 GB	16 MB
100	3600	13 GB	195 GB	26 MB
150	5400	19 GB	283 GB	36 MB
200	7200	25 GB	375 GB	46 MB

在上述计算中，大约增加了预期大小的20%作为开销，以确保存储需求不超过计算值。