# CEPH管理器守护进程（Mgr）

# 文档说明

官方文档（英文）：<http://docs.ceph.com/docs/master/mgr/>

由于Ceph中文社区没有对应这部分的内容，本文档主要摘录翻译了官方英文文档的Ceph Manager Daemon这一节。



# 序言

Ceph的管理守护进程（ceph-mgr）与监控进程一起，向外部监测和管理系统提供额外的监测和管理接口。

自12.x版本的 (luminous)Ceph发行版以来，ceph-mgr守护程序是正常操作所必需的。在11.x（kraken）中ceph -mgr守护程序是Ceph发行版中的可选组件。

默认情况下，管理守护进程不需要额外的配置，除了确保它正在运行。如果没有运行mgr守护进程，您将看到一个健康警告，ceph状态输出中的其他一些信息将会丢失或过期，直到mgr启动。

使用正常的部署工具（如ceph-ansible或ceph-deploy），在每个mon节点上设置ceph-mgr守护进程。将mgr守护进程放置在与mons相同的节点上并不是强制性的，但它几乎总是合理的。

# 1. 安装和配置

## 1.1 手动设置

通常，你可以使用ceph-ansible这样的工具来建立一个ceph-mgr守护进程。这些说明描述了如何手动设置ceph-mgr守护进程。

首先，为守护进程创建一个身份验证密钥：

ceph auth get-or-create mgr.$name mon 'allow profile mgr' osd 'allow \*' mds 'allow \*'

将该键放入路径mgr data中，对于集群“ceph”和mgr $ name“foo”来说应该是/var/lib/ceph/mgr/ceph-foo。

启动ceph-mgr守护进程：

ceph-mgr -i $name

通过ceph status的输出查看现在mgr已经启动，输出中应该包括一条有mgr状态的行：

mgr active: $name

## 1.2 客户端认证

管理器是一个新的守护进程，需要新的CephX功能。如果您从旧版本的Ceph升级集群，或者使用默认安装/部署工具，则管理客户端应该自动获得此功能。如果您使用其他地方的工具，则在调用某些ceph集群命令时可能会收到EACCES错误。为了解决这个问题，可以通过[*修改用户*](http://docs.ceph.com/docs/master/rados/operations/user-management/#modify-user-capabilities)*权限*来添加一个“mgr allow \*”节 到客户端的cephx功能中。

## 1.3 高可用性

一般来说，你应该在运行ceph-mon守护进程的每台主机上设置一个ceph-mgr来实现相同的可用性。

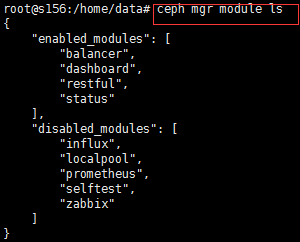
默认情况下，首先启动的ceph-mgr实例将被监视器激活，而其他的则作为备用。ceph-mgr守护进程中不需要法定人数（quorum）。

如果活动的（active）守护程序未能向监视器发送超过mon mgr beacon grace（默认30秒）的信标，则它将由备用守护程序取代。

如果你想抢占故障转移，你可以用ceph mgr fail <mgr name>显式标记一个ceph-mgr守护进程作为失败使用。

## 1.4 使用模块

使用ceph mgr module ls命令查看哪些模块可用，并且当前启用了哪些。使用命令ceph mgr module enable <module>和ceph mgr module disable <module>启用或禁用模块。

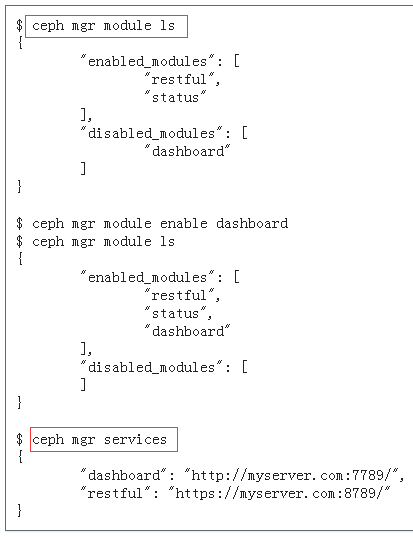


如果启用了一个模块，则活动的ceph-mgr守护进程将加载并执行它。在提供服务的模块下（例如HTTP服务器），模块可能会在加载时发布其地址。要查看这些模块的地址，请使用ceph mgr services命令 。

一些模块也可以实现特殊的待机模式，它在待机的ceph-mgr守护进程和活动的守护进程上运行。如果客户端尝试连接到备用模块，则会使提供服务的模块将其客户端重定向到活动的进程。

有关每个模块提供的功能的更多信息，请参阅个别管理器模块的文档页面。

以下是启用dashboard模块的示例：



## 1.5 调用模块命令

在一个模块实现命令行关联的情况下，这些命令将作为普通的Ceph命令来访问：

ceph <command | help>

如果您希望查看管理器处理的命令列表（其中普通的ceph help将显示所有的mon和mgr命令），则可以直接将命令发送到管理器守护程序：

ceph tell mgr help

请注意，没有必要寻址特定的mgr实例，只需mgr选择当前的活动守护进程即可。

## 1.6 配置

选项（mgr\_module\_path，OPT\_STR，CEPH\_PKGLIBDIR“/ mgr”）//从哪里加载python模块

**mgr module path**

|  |  |
| --- | --- |
| 描述： | 加载模块的路径。 |
| 类型： | String |
| 默认值： | "<library dir>/mgr" |

**mgr data**

|  |  |
| --- | --- |
| 描述： | 加载守护进程数据的路径（如keyring）。 |
| 类型： | String |
| 默认值： | "/var/lib/ceph/mgr/$cluster-$id" |

**mgr tick period**

|  |  |
| --- | --- |
| 描述： | mgr与监视器之间通信的时长（秒数），以及其他定期检查。 |
| 类型： | Integer |
| 默认值： | 5 |

**mon mgr beacon grace**

|  |  |
| --- | --- |
| 描述： | 距离上次mgr的最后一次通信过了多久，会视作通信失败 |
| 类型： | Integer |
| 默认值： | 30 |

# 2. ceph-mgr插件作者指南

## 2.1 创建一个插件

在pybind / mgr /中，创建一个python模块。在你的模块中，创建一个MgrModule继承的类。

重写的最重要的方法如下：

* 服务器类型模块的serve成员函数。应该永远禁用这个功能。
* 一个notify成员函数，当新集群的数据可用时假如你的模块需要。
* 一个handle\_command成员函数，假如你的模块暴露了CLI命令。

## 2.2 安装插件

一旦您的模块在配置mgr module path所设置的位置 ，您可以通过ceph mgr module enable命令启用它：

ceph mgr module enable mymodule

请注意，MgrModule接口不稳定，因此在运行任何较新或较旧版本的Ceph时，维护在Ceph树之外的任何模块都有可能中断。

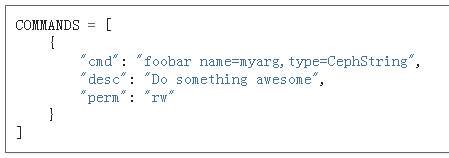
## 2.3 LOGGING

MgrModule实例有一个log属性，它是一个记录器实例，它将日志消息发送到Ceph日志记录层，并记录在mgr守护进程的日志文件中。

像使用其他Python记录器一样使用它。Python日志级别debug，info，warn，err分别映射到Ceph级别的20,4,1和0。

## 2.4 公开命令

将插件的COMMANDS类属性设置为这样的字典列表：



每个条目的cmd部分都以与内部的Ceph mon和管理套接字命令相同的方式进行解析（请参阅Ceph源中的mon / MonCommands.h为例）

## 2.5 配置设置

模块可以访问一个简单的键/值存储（键和值是字节字符串）来存储配置。不要用它来存储大量的数据。

配置值是使用mon的config-key命令存储的。

使用提示：

* 读取速度很快：ceph-mgr保留一份本地内存副本
* 不要用“ceph config-key”手动设置，mgr在运行时不会更新（只在模块内设置）。
* 写入块直到值被保存，但从另一个线程读取将立即看到新的值。

若你想将你模块内的任何配置暴露给用户，那么需要相应的命令中的hooks来暴露一个setter。

## 2.6 访问集群数据

模块可以访问mgr维护的Ceph集群状态的内存副本。Accessor函数作为MgrModule的成员暴露出来。

访问集群或守护进程状态的调用通常是从Python到本地C ++例程。这会有一些开销，但比调用REST API或调用SQL数据库要少得多。

访问集群结构或守护进程元数据没有一致性规则。例如，OSD可能存在于OSDMap中但没有元数据，反之亦然。在一个健康的集群中，这些将是非常罕见的短暂状态，但是应该写入插件以应付这种可能性。

请注意，这些访问器不能在模块的\_\_init\_\_方法中调用。否则会导致一个循环锁定异常。

**MgrModule.get(data\_name)**

|  |  |
| --- | --- |
| 描述： | 由插件调用以从ceph-mgr中获取名为cluster-wide的对象。 |
| 参数： | **data\_name** (str)  - 有效内容包括：   osd\_crush\_map\_text, osd\_map, osd\_map\_tree,  osd\_map\_crush, config, mon\_map, fs\_map, osd\_metadata,  pg\_summary, df, osd\_stats, health, mon\_status. |
| 注意： | 所有这些结构都有自己的JSON表示：试验一下或查看C ++ dump()方法来了解它们。 |

**MgrModule.get\_server(hostname)**

|  |  |
| --- | --- |
| 描述： | 由插件调用以从ceph-mgr中获取有关特定主机名的元数据。这是ceph-mgr从运行在特定服务器上的守护进程所报告的守护进程元数据中搜集的信息。 |
| 参数： | Parameters:**hostname** – a hostame |

**MgrModule.list\_servers()**

|  |  |
| --- | --- |
| 描述： | 像get\_server，但提供有关所有服务器的信息（例如：在守护进程元数据中提到的所有唯一的主机名） |
| 返回： | 关于所有服务器的信息列表 |
| 返回类型： | [list](http://docs.ceph.com/docs/master/rbd/api/librbdpy/#rbd.RBD.list) |

**MgrModule.get\_metadata（svc\_type，svc\_id ）**

|  |  |
| --- | --- |
| 描述： | 获取特定服务的守护进程元数据。 |
| 参数： | svc\_type（str） - 服务类型（例如'mds'，'osd'，'mon'）  svc\_id（str） - 服务ID。调用时将OSD整数ID转换为字符串 |
| 返回类型： | dict |

**MgrModule.get\_counter（svc\_type，svc\_name，path ）**

|  |  |
| --- | --- |
| 描述： | 由插件调用以获取特定服务上特定计数器的最新性能counter数据。 |
| 参数： | svc\_type（str） -  svc\_name（str） -  path（str） - 子系​​统和counter的名称以句点分隔连接，例如“mds.inodes”。 |
| 返回： | 返回（timestamp，value）的二元组列表。如果没有数据可用，这可能是空的。 |

|  |  |
| --- | --- |
| 描述： |  |
| 参数： |  |
| 注意： |  |

## 2.7 如果这些监视器挂了怎么办？

管理守护进程从监视器获取大部分状态（如集群映射）。如果监视器集群不可访问，则无论哪台active的管理器都将继续运行，并保持最新的状态。

但是，如果要创建一个向用户显示集群状态的模块，则可能不希望通过向用户显示过时的状态来误导他们。

要检查管理器守护进程当前是否与监视集群有连接，请使用以下函数：

**MgrModule.have\_mon\_connection（）**

|  |  |
| --- | --- |
| 描述： | 检查这个ceph-mgr守护进程是否与监视器打​​开连接。如果没有，那么很可能我们对群集的信息已经过时，或者监视器群集已关闭。 |

## 2.8 发送命令

提供了一个非阻塞设施来将监视器命令发送到集群。

**MgrModule.send\_command（\* args，\*\* kwargs ）**

|  |  |
| --- | --- |
| 描述： | 由插件调用来给监视器集群发送命令 |
| 参数： | **result**（*CommandResult*） - CommandResult 类的一个实例，在与MgrModule相同的模块中定义。它充当存储命令的完成和输出。如果你想阻止命令完成，可使用 CommandResult.wait()。 |
|  | **svc\_type**（*str*） -  **svc\_id**（*str*） - |
|  | **command**（*str*） - 一个JSON序列化的命令。它使用与ceph命令行相同的格式（一个命令参数字典，命令本身包含了额外的prefix键）。请参阅MonCommands.h以获取可用命令及其预期参数。 |
|  | **tag**（*str*） - 用于非阻塞操作：当命令完成时，触发了notify()对MgrModule实例的回调，notify\_type设置为“command”，notify\_id设置为命令的标记。 |

## 2.9 执行待机模式

对于某些模块，在备用管理守护程序以及活动守护进程上运行是非常有用的。例如，HTTP服务器可以有效地提供来自备用管理器的HTTP重定向响应，以便用户可以将浏览器指向任何管理员守护进程，而不必担心哪个进程处于活动状态。

备用管理器守护进程在每个模块中寻找StandbyModule的一个子类。如果没有找到该类，则在待机守护程序中根本不使用该模块。如果找到该类，则它的serve方法会被调用。StandbyModule 的实现必须从mgr\_module.MgrStandbyModule继承。

比起MgrModule，接口MgrStandbyModule大大受到限制——对于这个模块，没有一个Ceph簇是可用的。serve和shutdown方法和普通模块类一样使用。get\_active\_uri方法使备用模块能够发现其活跃部分（avtive peer）的地址以便重定向。请参阅源代码中MgrStandbyModuleCeph的定义以获取完整的方法列表。

有关如何使用此接口的示例，请查看dashboard模块的源代码。

## 2.10 LOGGING

使用你的模块的log属性作为你的记录器。这是一个通过ceph日志框架将日志输出配置为本地ceph-mgr日志文件的记录器。

Python日志的严重性按照如下标准映射到ceph的严重性：

* DEBUG是20
* INFO是4
* WARN是1
* ERR是0

## 2.11 干净地关闭

如果一个模块实现了serve()这个方法，它也应该实现shutdown()方法来彻底关闭：对模块行为不当的访问可能会阻止ceph-mgr的干净地关闭。

## 2.12 有什么缺失？

ceph-mgr python的接口不是一成不变的。如果你有一个目前接口不能满足的需求，请把它放在ceph-devel邮件列表中。虽然希望避免使接口膨胀，但在有充分理由的情况下，将现有数据公开给Python代码通常不是很困难。

# 3. 仪表板（DASHBOARD）插件

仪表板插件使用ceph-mgr托管的Web服务器可视化集群的统计信息。

## 3.1 启用

该*仪表板*模块用以下命令启用：

ceph mgr module enable dashboard

## 3.2 配置

像大多数Web应用程序一样，仪表板绑定到主机名和端口。默认情况下，ceph-mgr守护程序托管仪表板（例如：当前活动的管理程序）将绑定到端口7000和主机上任何可用的IPv4或IPv6地址。

由于每个ceph-mgr主机都是其本身的仪表板实例，所以也可能需要单独配置它们。主机名和端口可以通过配置密钥设备进行更改：

ceph config-key set mgr/dashboard/$name/server\_addr $IP

ceph config-key set mgr/dashboard/$name/server\_port $PORT

其中，$name是托管这个仪表板web app的ceph-mgr的ID。

这些设置也可以在集群范围内配置，而不是特定的管理程序。例如：

ceph config-key set mgr/dashboard/server\_addr $IP

ceph config-key set mgr/dashboard/server\_port $PORT

如果端口未配置，则Web应用程序将绑定到端口7000。如果未配置地址，则Web应用将绑定到::，这对应于所有可用的IPv4和IPv6地址。

## 3.3 反向代理

如果您通过反向代理配置访问仪表板，则可能希望在URL前缀下获取服务。要使仪表板使用包含前缀的超链接，可以通过以下命令设置url\_prefix：

ceph config-key set mgr/dashboard/url\_prefix $PREFIX

所以你可以在http://$IP:$PORT/$PREFIX/访问仪表板。

# 4. 本地池（LOCAL POOL）插件

该*localpool*插件可以自动创建定位于整个集群一个子集的RADOS池。例如，默认情况下，它将为集群中的每个不同机架创建一个池。

对于一些希望在本地以及全局范围内分发一些数据的部署，这可能非常有用。

## 4.1 启用

该*localpool*模块由以下命令启用：

ceph mgr module enable localpool

## 4.2 配置

该*localpool*模块可以支持以下选项：

* **subtree** (default: rack)：模块该为池创建哪个CRUSH子树类型。
* **failure\_domain**（默认：host）：我们应该把哪个故障域中的数据副本分隔开。
* **pg\_num**（默认值：128）：为每个池创建的PG的数量
* **num\_rep**（默认值：3）：每个池的副本数。（目前，池总是有副本的。）
* **min\_size**（默认值：none）：将min\_size设置为的值（如果未设置此选项，则不改变Ceph的默认值）
* **prefix**（默认：by- $ subtreetype-）：池名称的前缀。

这些选项通过配置密钥接口进行设置。例如，要将仅64个PG的副本数量更改为2：

ceph config-key set mgr/localpool/num\_rep 2

ceph config-key set mgr/localpool/pg\_num 64

# 5. RESTFUL插件

RESTful插件通过SSL安全连接提供REST API访问集群状态的权限。

## 5.1 启用

该*restful的*模块由以下命令启用：

ceph mgr module enable restful

在API端点可用之前，您还需要配置下面的SSL证书。默认情况下，模块将接受8003主机上所有IPv4和IPv6地址上的端口上的HTTPS请求。

## 5.2 安全

所有与*restful的*连接都使用SSL进行保护。您可以使用以下命令生成自签名证书：

ceph restful create-self-signed-cert

请注意，对于自签名证书，大多数客户端需要一个标志来允许连接和/或禁止警告消息。例如，如果ceph-mgr守护进程在同一主机上，则：

curl -k https://localhost:8003/

为了妥善保护部署，应该使用由组织的证书颁发机构签发的证书。例如，可以使用类似于以下的命令来生成密钥对：

openssl req -new -nodes -x509 \

-subj "/O=IT/CN=ceph-mgr-restful" \

-days 3650 -keyout restful.key -out restful.crt -extensions v3\_ca

然后，restful.crt应该由你组织中的CA（证书颁发机构）签名。一旦完成，您可以将用以下命令设置：

ceph config-key set mgr/restful/$name/crt -i restful.crt

ceph config-key set mgr/restful/$name/key -i restful.key

其中$name是ceph-mgr实例的名称（通常是主机名）。如果所有管理实例要共享相同的证书，则可以省略$name部分：

ceph config-key set mgr/restful/crt -i restful.crt

ceph config-key set mgr/restful/key -i restful.key

## 5.3 配置IP和端口

像任何其他RESTful API端点一样，*restful*绑定到某个IP和端口。默认情况下，当前活动的ceph-mgr守护进程将绑定到端口8003和主机上任何可用的IPv4或IPv6地址。

由于每个ceph-mgr主机都有自己的*restful*实例，因此也可能需要分别进行配置。IP和端口可以通过配置密钥设备进行更改：

ceph config-key set mgr/restful/$name/server\_addr $IP

ceph config-key set mgr/restful/$name/server\_port $PORT

$name是ceph-mgr守护进程的ID（通常是主机名）。

这些设置也可以在集群范围内配置，而不指定管理者。例如：

ceph config-key set mgr/restful/server\_addr $IP

ceph config-key set mgr/restful/server\_port $PORT

如果端口没有配置，*restful*将绑定到端口8003。如果没有配置地址，*restful*将绑定到::，这对应于所有可用的IPv4和IPv6地址。

## 5.4 负载平衡器

请注意，*restful*将*只*在那一刻活跃的管理器上启动。查询Ceph集群状态以查看哪个管理器处于活动状态（例如，ceph mgr dump）。无论哪个管理器守护进程当前处于活动状态，为了使API通过一致的URL可用，您可能需要在前端设置一个负载平衡器，以将流量引导至任何可用的管理器端点。

# 6. ZABBIX插件

Zabbix插件主动向Zabbix服务器发送信息，如：

* Ceph状态
* I / O操作
* I / O带宽
* OSD状态
* 存储利用率

## 6.1 要求

该插件要求在运行ceph -mgr的所有机器上都存在zabbix\_sender可执行文件。它可以使用包管理器（package manager）安装在大多数发行版上。

### 6.1.1 依赖

安装zabbix\_sender可以在Ubuntu或者CentOS下使用apt或者dnf来完成。

在Ubuntu Xenial上：

apt install zabbix-agent

在Fedora上：

dnf install zabbix-sender

## 6.2 启用

您可以用以下命令启用zabbix模块：

ceph mgr module enable zabbix

## 6.3 配置

两个配置键对模块的工作至关重要：

* zabbix\_host
* identifier（可选）

参数zabbix\_host控制zabbix\_sender将向其发送项目的Zabbix服务器的主机名 。如果您的安装需要，这可以是IP地址。

发送项目到Zabbix时，identifier参数控制作为源的标识符/主机名。这应该与Zabbix服务器中的主机名称相匹配。

当identifier参数未配置，给Zabbix发送数据将使用群集的ceph-<fsid>。

例如ceph-c4d32a99-9e80-490f-bd3a-1d22d8a7d354

可以配置的其他配置键及其默认值：

* zabbix\_port: 10051
* zabbix\_sender: /usr/bin/zabbix\_sender
* interval: 60

### 6.3.1 配置键

可以在任何机器上使用正确的cephx证书来设置配置键，这些机器通常是*client.admin*键目前值的监视器。

ceph zabbix config-set <key> <value>

例如：

ceph zabbix config-set zabbix\_host zabbix.localdomain

ceph zabbix config-set identifier ceph.eu-ams02.local

也可以显示模块的当前配置：

ceph zabbix config-show

## 6.4 手动发送数据

如果需要，可以要求模块立即发送数据，而不是等待一段间隔。

这可以用这个命令来完成：

ceph zabbix send

该模块立刻将其最新的数据发送到Zabbix服务器。

## 6.5 调试

如果您想调试Zabbix模块，请增加ceph-mgr的日志级别并检查日志。

[ mgr ]

debug mgr = 20

将日志设置为管理器调试时，插件将打印以*mgr [zabbix]*为前缀的各种日志行，以便于过滤。

# 7. PROMETHEUS插件

提供了一个从ceph-mgr的收集点传递Ceph性能计数的Prometheus出口。Ceph-mgr从性能计数器模式数据和实际计数器数据中接收来自所有MgrClient进程（例如监视器和OSD）的MMgrReport消息，并保留最后N个采样的循环缓冲区。这个插件创建一个HTTP端点（就像所有的Prometheus出口一样），并在轮询（或在Prometheus术语中为“scraped”）时检索每个计数器的最新样本。HTTP路径和查询参数被忽略；所有报告实体的所有现存计数器均以文本说明格式返回。（请参阅Prometheus [文档](https://prometheus.io/docs/instrumenting/exposition_formats/#text-format-details)。）

## 7.1 启用Prometheus输出

用以下命令启用Prometheus模块：

ceph mgr module enable prometheus

### 配置

默认情况下，模块将接受9283主机上所有IPv4和IPv6地址上的端口上的HTTP请求。端口和监听地址都可以用ceph config-key set设置，其中键为 mgr/prometheus/server\_addr和mgr/prometheus/server\_port。此端口已在Prometheus的[注册表中注册](https://github.com/prometheus/prometheus/wiki/Default-port-allocations)。

## 7.2 统计名称和标签

统计的名称与Ceph相同，对所有名称前缀把非法字符.，-和::转换为\_，和ceph\_。

所有*守护进程*统计信息都有一个ceph\_daemon标签，例如“osd.123”，标识它们来自的守护进程的类型和ID。有些统计信息可能来自不同类型的守护进程，所以在查询OSD的RocksDB统计信息时，可能需要在“osd”开头的ceph\_daemon上进行过滤，以避免混淆监视器rocksdb统计信息。

集群统计信息（例如：对Ceph集群是全局的）有他们基于什么报告的相应标签。例如，与池相关的度量就有一个pool\_id标签。

### 7.2.1 池和OSD元数据SERIES

在特定的元数据字段显示和查询，会输出特殊的series。

池有这样一个ceph\_pool\_metadata域：

ceph\_pool\_metadata{pool\_id="2",name="cephfs\_metadata\_a"} 0.0

OSD有一个ceph\_osd\_metadata这样的域：

ceph\_osd\_metadata{cluster\_addr="172.21.9.34:6802/19096",device\_class="ssd",id="0",public\_addr="172.21.9.34:6801/19096",weight="1.0"} 0.0

### 7.2.2 将驱动器统计与NODE\_EXPORTER关联

Ceph的prometheus输出被设计为与Prometheus node\_exporter的通用主机监控结合使用。

为了启用Ceph OSD统计信息与node\_exporter的驱动器统计信息的关联，特殊series输出如下：

ceph\_disk\_occupation{ceph\_daemon="osd.0",device="sdd",instance="myhost",job="ceph"}

要通过OSD ID获得磁盘统计信息，请如以下在prometheus查询中使用and on语法：

rate(node\_disk\_bytes\_written[30s]) and on (device,instance) ceph\_disk\_occupation{ceph\_daemon="osd.0"}

有关构建查询的更多信息，请参阅prometheus文档。

请注意，为了使此机制起作用，Ceph和node\_exporter必须同意instance标签的值。请参阅以下部分，了解如何用一种正确设置 instance的方式来启动 Prometheus。

## 7.3 配置PROMETHEUS服务器

有关如何添加scrap endpopints的完整详细信息，请参阅prometheus文档：本节中的注释提示如何配置Prometheus来最有用的方式捕获Ceph统计信息。

由于Ceph通过单个端点（endpoint）报告来自许多主机和服务的度量标准，以及一些与没有物理主机相关的度量标准（如池统计信息），因此此配置是必需的。

### 7.3.1 HONOR\_LABELS

要启用Ceph输出与任何主机相关的正确标记的数据，请在将ceph-mgr端点添加到你的prometheus配置时使用honor\_labels设置。

如果没有这个设置，那么Ceph输出的任何instance标签，比如那些在ceph\_disk\_occupation series中的，将被Prometheus覆盖。

### 7.3.2 CEPH的实例标签

默认情况下，Prometheus应用一个instance标签：包含series端点的主机名和端口。由于Ceph集群具有多个管理守护进程，因此当活动管理守护进程更改时，会导致instance标签虚假地更改。

在您的Prometheus目标配置中设置一个自定义的instance标签：您可能希望将其设置为您的第一个显示器的主机名，或者像“ceph\_cluster”这样的完全任意的内容。

### 7.3.3 NODE\_EXPORTER实例标签

设置您的instance标签以匹配在Ceph OSD 元数据中hostname字段中的值。这通常是节点的简短主机名。

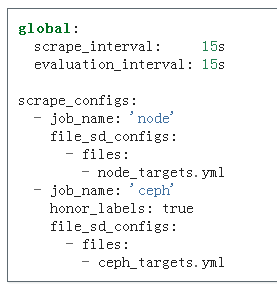
如果你想把Ceph数据与主机数据关联起来，这才是必要的；但你会发现在所有情况下执行这一步会很有用，以防你想在未来进行关联。

### 7.3.4 配置示例

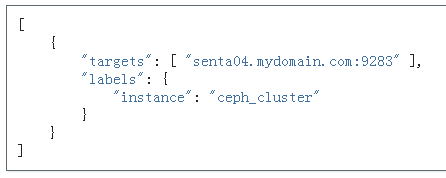
这个例子显示了在一个名为senta04服务器的服务器上运行ceph-mgr和node\_exporter的单个节点配置。

这只是一个例子：还有其他的方法来配置prometheus scrape目标和标签重写规则。

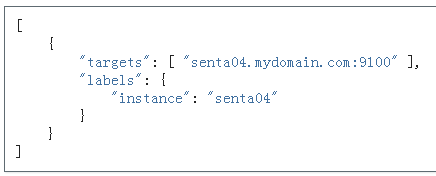
#### 7.3.4.1 PROMETHEUS.YML



#### 7.3.4.2 CEPH\_TARGETS.YML



#### 7.3.4.3 NODE\_TARGETS.YML



## 7.4 注意

计数器和仪表被输出了；目前直方图和长时间运行的平均值还没有。Ceph的二维直方图有可能被缩减为两个单独的一维直方图，长时间运行的平均值可以被导出为Prometheus的概要类型。

与许多Prometheus出口商一样，Timestamp是由服务器的scrape时间建立的（Prometheus希望它同步轮询实际的计数器过程）。它有可能与统计报告一起提供一个时间戳，但Prometheus团队强烈建议不要这样做。这意味着时间戳将被推迟一个不可预测的值。目前尚不清楚这是否会造成问题，但值得了解。

# 8. INFLUX插件

influx插件不断地收集并发送时间序列数据到influxdb数据库。

influx插件是在13.x Mimic版本中引入的。

## 8.1 启用

要启用该模块，请使用以下命令：

ceph mgr module enable influx

如果您希望随后禁用该模块，则可以使用等效的 禁用命令：

ceph mgr module disable influx

## 8.2 配置

为了使Influx模块向InfluxDB服务器发送统计信息，需要配置服务器地址和一些认证凭证。

使用以下命令设置配置值：

ceph config-key set mgr/influx/<key> <value>

最重要的设置是hostname，username和password。例如，典型的配置可能如下所示：

ceph config-key set mgr/influx/hostname influx.mydomain.com

ceph config-key set mgr/influx/username admin123

ceph config-key set mgr/influx/password p4ssw0rd

其他可选配置设置是：

**interval**

|  |  |
| --- | --- |
| 描述： | 向InfluxDB报告的时间间隔。默认5秒 |

**database**

|  |  |
| --- | --- |
| 描述： | InfluxDB数据库名称。默认“ceph”。您将需要创建此数据库并将写入权限授予配置的用户名，或者用户名必须具有管理员权限才能创建。 |

**port**

|  |  |
| --- | --- |
| 描述： | InfluxDB服务器端口。默认8086 |

**ssl**

|  |  |
| --- | --- |
| 描述： | 使用https连接InfluxDB服务器。使用“true”或“false”。默认为false |

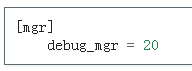
**verify\_ssl**

|  |  |
| --- | --- |
| 描述： | 验证InfluxDB服务器的https证书。使用“true”或“false”。默认为true |

## 8.3 调试

默认情况下，一些调试语句和错误语句已被设置为在日志文件中打印。如果需要，用户可以添加更多。要使用模块中的调试选项：

* 将其添加到ceph.conf文件中：



* 使用ceph tell mgr.<mymonitor> influx self-test这个命令。
* 检查日志文件。用户可能会发现使用mgr [influx]过滤日志文件更容易。

## 8.4 有趣的计数

下表描述了该模块输出的值的子集。

### 8.4.1 POOLS

|  |  |
| --- | --- |
| **Counter** | **Description** |
| bytes\_used | 池中使用的字节（不包括副本） |
| max\_avail | 池中的最大可用字节数 |
| objects | 池中的对象数量 |
| wr\_bytes | 写入池中的字节数 |
| dirty | 池中脏的字节数 |
| rd\_bytes | 在池中读取的字节数 |
| raw\_bytes\_used | 池中使用的字节（包括副本） |

### 8.4.2 OSDS

|  |  |
| --- | --- |
| **Counter** | **Description** |
| op\_w | 客户端写入操作 |
| op\_in\_bytes | 客户端操作的总写入大小 |
| op\_r | 客户端读取操作 |
| op\_out\_bytes | 客户端操作的总读取大小 |

|  |  |
| --- | --- |
| **Counter** | **Description** |
| op\_wip | 当前正在处理的复制操作（主要） |
| op\_latency | 客户端操作的延迟（包括队列时间） |
| op\_process\_latency | 客户端操作的延迟（不包括排队时间） |
| op\_prepare\_latency | 客户端操作的延迟（不包括排队时间和等待完成时间） |
| op\_r\_latency | 读取操作的延迟（包括排队时间） |
| op\_r\_process\_latency | 读取操作的延迟（不包括排队时间） |
| op\_w\_in\_bytes | 客户端数据写入 |
| op\_w\_latency | 写操作的延迟（包括排队时间） |
| op\_w\_process\_latency | 写操作的延迟（不包括排队时间） |
| op\_w\_prepare\_latency | 写入操作的延迟（不包括排队时间和等待完成） |
| op\_rw | 客户端读取 - 修改 - 写入操作 |
| op\_rw\_in\_bytes | 客户端读取 - 修改 - 写入操作写入 |
| op\_rw\_out\_bytes | 客户端读取 - 修改 - 写入操作读出 |
| op\_rw\_latency | 读取 - 修改 - 写入操作的延迟（包括排队时间） |
| op\_rw\_process\_latency | 读取 - 修改 - 写入操作的延迟（不包括排队时间） |
| op\_rw\_prepare\_latency | 读取 - 修改 - 写入操作的延迟（不包括排队时间和等待完成） |
| op\_before\_queue\_op\_lat | 调用队列op\_before\_dequeue\_op\_lat之前IO的延迟（真正排队到ShardedOpWq之前） |
| op\_before\_dequeue\_op\_lat | 调用dequeue\_op之前IO的延迟（已经出队并获得PG锁） |

延迟计数器的测量单位为微秒，除非说明书中另有说明。