计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):283−287 [doi: 10.15888/j.cnki.csa.005854]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于Ceilometer Alarm API的操作设计和实现①

李超雷, 向忠清

(武汉烽火信息集成技术有限公司 IT事业部, 武汉 430074)

摘　要: 随着云计算技术的不断发展, 越来越多的企业采用OpenStack来构建私有云或公有云平台. 云平台正逐步替代传统服务器, 用来承载着企业和用户的IT业务. 为了保证云平台的服务质量, 本文基于OpenStack的报警功能接口——Ceilometer Alarm API设计和实现了对于云平台虚拟机监控报警功能的交互操作页面. 通过使用该功能, 用户可以监控虚拟机运行时的性能状态, 保证云平台的可靠运行.

关键词: 云计算; Openstack; Ceilometer; 报警

引用格式: 李超雷,向忠清.基于Ceilometer Alarm API的操作设计和实现.计算机系统应用,2017,26(7):283–287. http://www.c-s-a.org.cn/1003-3254/5854.html

Design and Implementation of the Operation Based on Ceilometer Alarm API

LI Chao-Lei, XIANG Zhong-Qing

(IT Division,WuHan Fiberhome Integrated Information Technology Co.Ltd., Wuhan 430074, China)

Abstract: With the continuous development of cloud computing technology, more and more enterprises begin to use OpenStack to build a private cloud or public cloud platform. Cloud platform is gradually replacing the traditional server, to undertake the enterprises and user IT business. In order to guarantee the service quality of the cloud platform, this article designs the interactive operation page of monitoring and alarming function for the virtual machine of cloud platform, based on the OpenStack alarm function interface -- Ceilometer Alarm API. Using this function , users can monitor the running state of the virtual machine, ensuring the reliable operation of the cloud platform.

Key words: cloud computing; Openstack; Ceilometer; alarm

1　OpenStack介绍

云计算技术在近几年得到了飞速的发展, 通常来说, 云计算主要包括基础设施即服务(IaaS)、平台作即服务(PaaS)和软件即服务(SaaS)[1]. 本文讨论的主要内容属于IaaS内容.

OpenStack[2-4]是一款开源云计算管理平台, 由NASA和Rackspace合作研发并发起, 目前已经发展成为仅次于Linux的全球第二大社区. OpenStack提供了基础设施即服务(IaaS)的解决方案, 其首要任务是简化云的部署过程并为其带来良好的可扩展性. OpenStack的核心项目包括计算(Nova)、网络(Neutron)、身份认证(Keystone)、块存储(Cinder)、对象存储(Swift)、测量(Ceilometer)和操作面板(Horizon). 其中Ceilometer提供了对整个云平台的资源进行计量和统计的功能[5]. 本文实现的虚拟机告警功能主要基于Ceilometer[6], 为了证明Ceilometer数据的有效性, 下一节内容主要对Ceilometer模块进行了分析.

2　Ceilometer详解

Ceilometer是OpenStack中的一个子项目, 它能把OpenStack内部发生的几乎所有事件都收集起来, 然后为计费和监控以及其他服务提供数据支撑.

① 收稿时间: 2016-11-10; 收到修改稿时间: 2016-12-12

2.1 核心架构

Ceilometer使用两种收集数据的方式, 一种是OpenStack各个服务内发出的notification消息, 一种是通过调用各个服务的API主动获取数据.

Ceilometer之所以采用这两种收集方式, 首先是因为, 在OpenStack内部, 发生的一些事件都会发出对应的notification消息, 比如创建和删除虚拟机等, 这些都是计量的重要信息, 因此, notification消息作为Ceilometer的第一数据来源. 除此之外, 有些计量信息通过notification消息是获取不到的, 比如说虚拟机的CPU使用率、磁盘I/O速率等. 这些信息不会主动的通过notification消息发送出来, 因此Ceilometer增加了第二种方式, 周期性的调用相关API去获取这些信息. 在Ceilometer源码中, 如图1所示, 在/etc/ceilometer/pipeline.yaml文件中配置了轮询时间为600秒, 也就是说每隔10分钟Ceilometer会主动获取当前系统中的各项信息并存入到数据库中, 此信息作为Ceilometer的第二种数据来源, 可以根据需要适当的增加或减少轮询时间, 需要注意的是, 较短的轮询时间会增加OpenStack后台的压力. Ceilometer轮询时间如下:

```
sources:
  - name: meter_source
    interval: 600
```

Ceilometer的各个服务中, 与采集相关的服务是ceilometer-collecter、ceilometer-agent-central、ceilometer-agent-compute、ceilometer-agent-notification. 关系如图1所示.

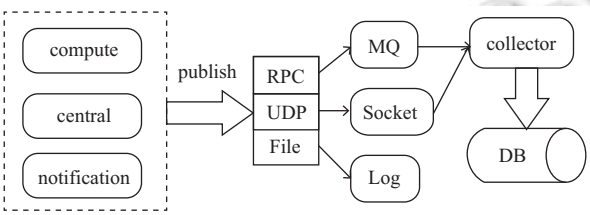

图1 Ceilometer组件关系图

如图1所示, agent-central、agent-compute、agent-notification负责采集信息, 其中agent-compute用来收集计算节点上的信息, 在每一个计算节点上都要运行一个agent-compute, 用来获取虚拟机的CPU、Disk IO、Network IO、Instance这些信息. agent-central运行在控制节点上, 它主要收集其他服务如Image、Volume、Objects、Network等信息. agent-notification则是负责收集各个组件推送到消息队列中的信息, 与agent-central和agent-compute不同的是, agent-notification只需要监听事件的发生, agent-central和agent-compute都需要定时poll轮询收集信息. 采集到的信息由collecter汇总处理, 并将数据写入到数据库中, collecter是在核心架构中唯一一个能够对数据库进行写操作的组件. 目前Ceilometer支持的存储包括mysql、DB2、HBase、mongoDB, 在本文所用的OpenStack Kilo版本中, Ceilometer使用mongoDB作为后端存储, 目前来看, 监控数据的持久化的压力还是很大的.

2.2 Ceilometer Alarm

Ceilometer Alarm是OpenStack Havana版本添加的新功能, 根据2.1节的介绍, Ceilometer已经实现了比较完善的监控体系, Alarm便是利用Ceilometer收集到的信息对云平台进行监控报警.

Alarm能够监控一个或多个指标的值, 若高于或低于阈值, 则执行相应的动作. 在Ceilometer中, 实现了2种Alarm, threshold和combination. Threshold针对某一个监控指标建立Alarm, 根据监控指标的阈值去判断Alarm的状态. Combination则是将多个Alarm组合起来的综合考虑监控指标的Alarm, 多个Alarm之间是or/and的关系.

OpenStack提供了RESTful API对Alarm进行操作, 例如创建、更新、删除、查询等, 表1详细的列出了创建一个Alarm需要的参数.

表1中AlarmThresholdRule和AlarmCombinationRule是两种Alarm类型的参数, 其中AlarmThresholdRule包括:

meter_name: 监控指标

query: 指定用于监控何种资源, 可以绑定至具体的虚拟机上

period, evaluation_periods: 确定监控的时间范

threshold: 阈值

comparison_operator: 和阈值比较的方式

statistic: 确定了使用何种数据与阈值threshold进行比较

exclude_outliners: 是否除去波动较大数据

AlarmCombinationRule包括:

operator: 定义alarm之间的逻辑关系

alarm_id: 为alarm的列表

表1 创建Alarm的详细参数

参数	类型	解释
name	str	name是project唯一的
description	str	alarm描述
enabled	bool	用于停止/启用该alarm
ok_actions	list	alarm为OK状态时的动作, 默认为[]
alarm_actions	list	alarm为alarm状态时的动作, 默认为[]
insufficient_data_actions	list	alarm为insufficient data状态时的动作, 默认为[]
repeat_actions	bool	alarm触发时, 是否重复执行对应动作
type	str	alarm类型, 有threshold和combination, 必填
threshold_rule	AlarmThresholdRule	当alarm为threshold时, 制定的threshold规则
combination_rule	AlarmCombinationRule	当alarm为combination时, 制定的combination规则
time_constraints	list	约束该alarm在哪段时间段执行, 默认是[]
state	str	alarm的状态
user_id	str	User id
project_id	str	Project id
timestamp	datetime	alarm的定义最后一次被更新的时间
state_timestamp	datetime	alarm的状态最后一次更改的时间

本文虚拟机报警功能实现的原理是利用Alarm中的actions功能. actions主要提供两种形式的接口, 一种是log, 该形式可以将报警信息写入到指定的日志文件中; 另一种是REST接口, 当报警发生时, OpenStack会使用http协议访问actions里设置的URL, 同时产生的报警信息会以JSON数据的格式包含在HTTP头文件中以POST形式发送过去. 本文采用的报警接收方式便是使用REST接口形式.

3 虚拟机报警的实现

根据前一节的描述, Ceilometer提供的Alarm功能可以提供监控功能, 不足的是, OpenStack仅仅是提供了Alarm的API, 没有提供友好的人机交互的页面. API通常是共程序开发人员使用, 用户使用起来并不方便. 本文接下来将基于OpenStack中的Ceilometer项目, 设计并实现虚拟机报警功能.

3.1 虚拟机报警功能流程

考虑到在实际的使用过程中, 用户并不需要考虑所创建的Alarm类型是Threshold还是Combination的, 用户只需要考虑自己需要监控何种数据, 并创建相应的报警规则便可. 报警规则所监控的数据可以是单一的某一项数据, 也可以是监控多种数据.

Web页面的流程上, 当用户规定好报警规则后, 可以根据需要选择是否发送邮件通知, 设置完通知邮件之后将报警规则绑定到一个或多个虚拟机上就可以了. 具体的创建流程如图2所示.

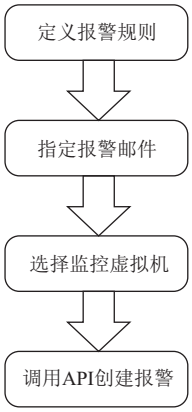

图2 告警规则创建流程

设置完报警规则之后, OpenStack会自动去监控所创建的Alarm的状态, 当报警触发之后, 会根据用户的设置发送相应的报警通知. Web后台也会收到该报警通知, 在接收到通知之后, Web后端会将数据写入数据库, 方便用户今后查看. 接收报警的流程如图3所示.

3.2 功能分析

根据3.1节的流程, 虚拟机报警功能主要有以下几个子功能的实现: 创建报警规则、查询报警规则、更新报警规则、删除报警规则、发送报警通知、查询已产生报警信息.

为了能够快速实现上述功能, 本文的Web后台采用Django[7]框架进行处理, Django是使用Python编写的一款开源的Web框架. 使用Django框架可以简便、快速的开发Web应用程序, 比较符合本文报警功能开发的需求.

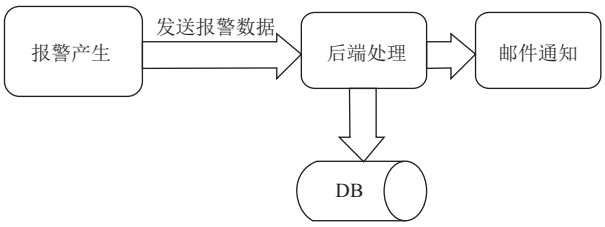

图3 报警接收流程

3.3 虚拟机报警功能实现

根据3.2节对虚拟机报警功能的需求分析, 对应每个功能, 本文将详述其具体实现, 根据Django项目的开发流程, 每个功能都有相对应的url和view, 对于需要对数据库操作的功能, 需要生成对应的model.

3.3.1 虚拟机报警规则查询

```
#url设置
url(r‘listAlarm/$’, monitor_view.getAlarmList),
#对应view
def getAlarmList(request):
#进行具体功能实现
```

该功能主要使用Ceilometer提供的API进行Alarm查询, 将查询到的信息进行过滤, 最终返回给Web页面. 用户的每次请求都会携带自身对应的Token, 因此不必担心接口会返回其他用户创建的报警规则. 但是, 由于Threshold和Combination这两种报警规则的存在, 为了屏蔽掉这两种规则的差别, 在getAlarmList的实现中需要过滤掉Combination中的子规则.

3.3.2 虚拟机报警规则创建

```
#url设置
url(r‘^createAlarm/$’, monitor_view.createAlarmsForOneRequest)
#对应view
def createAlarmsForOneRequest(request):
  #Do Create Alarm
```

该功能接收用户传入的数据进行Alarm的创建, 用户对于Alarm的类型是无感知的, 用户只需要考虑需要监控虚拟机的何种性能指标便可, 为了在Web上屏蔽掉Threshold和Combination的区别, 在后台的实现上可以通过判断报警规则中的监控数据是否单一来创建相应的Alarm, 核心代码如下:

```
#若监控数据唯一, 创建Threshold Alarm, 否则创建Combination Alarm
if len(alarm[‘rules’]) > 1:
  for rule in alarm[‘rules’]:
  #根据每条监控的数据创建Threshold Alarm

rule_data=ceilometer.CreateSingleAlarmData(rule)
  caller_assign_role=APICaller(rule_data)
  code, rule_response, e = caller_assign_role.call()
#将alarm_id放入list中, 为Combination Alarm提供数据准备
  rule_id.append(rule_response[‘alarm_id’])
  #创建Combination Alarm
alarm_data=ceilometer.CreateCombineAlarmData(rule_id)
else:
#创建Threshold Alarm

alarm_data=ceilometer.CreateThresholdAlarmData(alarm[‘rules’][0])
```

新建报警时是针对某台虚拟机, 因此需要使用API中的query字段, 具体数据形式如下:

```
‘query’:[{‘field’: ‘resource_id’,
  ‘op’: ‘eq’,
  ‘type’: ‘string’,
  ‘value’: instance_id}],
```

为了使用Alarm的action功能, 在创建Alarm时, 需要在对应action中添加OpenStack后台访问Django的URL地址, OpenStack会使用REST的形式将数据传送过来, action的具体形式如下:

```
action_url=email_host+‘/alarmInformation?user_id=%s&instance_name=%s&email=%s’%(user_id, instance_name, email)
#setting文件中设置email_host
‘email_host’: ‘http://10.89.155.102’
```

其中email_host在配置文件中设置, 为该项目的IP地址, email是用户的邮件地址, instance_name是虚拟机的名字, user_id为用户ID, 用户ID对于每个用户是唯一的, 用于识别报警所属用户.

3.3.3 虚拟机报警规则更新和删除

```
#url设置
url(r‘updataThreshold$’, monitor_view.updataThreshold)
url(r‘alarmDelete/$’, monitor_view.deleteAlarm)
```

```
#对应view
def updataThreshold(request):
    #Do Update Alarm
def deleteAlarm(request):
    #Do Delete Alarm
```

该功能在实现上需要注意的是, 对于Threshold类型的Alarm, 直接使用OpenStack提供的API进行更新和删除便可. 但是对于Combination类型的Alarm, 用户其alarm_id中的Alarm是无感知的, 在更新功能上实际是更新Combination的alarm_id中所对应的Threshold类型的Alarm. 在删除Combination类型的Alarm时, 需要删除alarm_id中的所有Alarm.

3.3.4 发送报警通知

```
#url设置
url(r'alarmInformation$', monitor_view.AlarmInformation)
#对应view
def AlarmInformation(request):
#Do sent_mail and save
```

该功能有两个主要用途, 第一, 提供OpenStack可访问接口, 接收报警信息; 第二, 发送报警通知并将报警信息存入数据库. 由于该功能与数据库相关, 需要使用Django中的model进行数据库的映射. Django的数据库映射功能使用十分简便, 在model.py文件中添加相关Model便可.

Django同样提供了发送邮件功能, 使用起来也同样十分方便. 首先, 在setting文件中设置好邮件服务器, 之后, 在需要使用到邮件发送功能的代码中直接import django.core.mail中的send_mail模块后, 便可以使用send_mail方法进行邮件的发送. setting文件的配置如下:

```
EMAIL_HOST = 'smtp.xxx.com'
#Email账号
EMAIL_HOST_USER = 'xxx@xxx.com'
#Email密码
EMAIL_HOST_PASSWORD = 'password'
EMAIL_USE_TLS = True
```

具体实现上, 当接收到OpenStack的报警通知, 将报警信息存入数据库并给用户发送邮件通知即可.

3.3.5 查询已产生报警信息

```
#url设置
url(r'listWarnings/$', monitor_view.listWarnings)
#对应view
def listWarnings(request):
#Do list warning
```

该功能主要用于用户查询已产生的报警信息, 使用Django的model可以很方便的查询到数据库所存的报警信息, 核心代码如下:

```
warningActions=Alarm.objects.filter(region=region_id, alarm_user_id=user_id).order_by('-alarm_create_time')
```

4 结语

本文基于OpenStack的Ceilometer实现了虚拟机的报警功能, 并且已经应用于实际环境当中. 目前, 该功能能够正常使用, 达到了虚拟机实时报警功能. 但是, 由于Ceilometer对数据的收集默认是600秒主动轮询一次, 报警的精度最好设置在600秒以上. 如果需要更加精细的报警, 则需缩短Ceilometer的轮询间隔, 这势必会增加服务器的压力, 有可能会影响到其它功能的正常运行. 如何能够提高Ceilometer的监控精度, 并不给服务器和数据库造成太大压力, 这将是今后研究的主要方向.

参考文献

1 Bohn RB, Messina J, Liu F, *et al*. NIST cloud computing reference architecture. 2011 IEEE World Congress on Services (SERVICES). Washington DC, USA. 2011. 594–596.

2 Floating IP. OpenStack: High Availability Guide. 2015.

3 Lamourine M. OpenStack. Login: the Magazine of USENIX & SAGE, 2014, 39(3): 17–20.

4 高贵升. 基于OpenStack的计算云的研究与实现[硕士学位论文]. 成都: 成都理工大学, 2012.

5 梁宇, 杨海波, 李鸿彬, 等. 基于OpenStack资源监控系统. 计算机系统应用, 2014, 23(4): 44–47, 16.

6 Dongmyoung B, Bumchul L. Analysis of telemetering service in OpenStack. 2015 International Conference on Information and Communication Technology Convergence. Jeju, South Korea. 2015. 272–274.

7 Forcier J, Bissex P, Chun W. Django Web开发指南. 徐旭铭, 译. 北京: 机械工业出版社, 2009.