ALIBABA CLOUD

阿里云

函数计算

监控报警

文档版本：20220708

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.监控指标

您可以在函数计算控制台查询函数计算资源概览指标以及资源所在地域、服务和函数维度的监控指标详情。具体的监控指标通过指定MetricName参数实现。本文介绍各种类型监控指标对应MetricName参数的取值和含义。

资源概览指标

您可以登录函数计算控制台，在**概览**页面的**资源使用统计**区域，查看资源概览指标的情况。

资源概览指标是您对所有地域或某指定地域内，函数计算整体资源使用情况和网络流量的监控度量，包括以下指标项：

指标类型	指标名称	单位	描述
概览	调用次数（Invocations）	次	调用函数的总请求次数。按一天或一个月粒度统计求和。
	资源使用量（Usage）	GB-秒	在调用函数时，函数消耗的资源，即函数内存×函数执行时间。按一天或一个月粒度统计求和。
	公网出流量（InternetOut）	GB	在调用函数时，函数执行在统计时间内的总公网出流量。按一天或一个月粒度统计求和。
资源使用	按量弹性实例资源使用量（ElasticOndemandUsage）	GB-秒	在调用函数时，函数消耗的按量弹性实例资源，即函数内存×函数执行时间。按一天或一个月粒度统计求和。
	按量性能实例资源使用量（EnhancedOndemandUsage）	GB-秒	在调用函数时，函数消耗的按量性能实例资源，即函数内存×函数执行时间。按一天或一个月粒度统计求和。
	预留弹性实例资源使用量（ElasticProvisionUsage）	GB-秒	在调用函数时，函数消耗的预留弹性实例资源，即函数内存×实例预留时间。按一天或一个月粒度统计求和。
	预留性能实例资源使用量（EnhancedProvisionUsage）	GB-秒	在调用函数时，函数消耗的预留性能实例资源，即函数内存×实例预留时间。按一天或一个月粒度统计求和。
公网流量	函数内数据传输流量（DataTransferInternetOut）	GB	在调用函数时，函数内部访问公网的数据传输流量。按一天或一个月粒度统计求和。
	函数请求响应流量（InvokeInternetOut）	GB	通过公网调用函数，函数执行完成并返回响应时所产生的流量。按一天或一个月粒度统计求和。
	CDN回源流量（InvokeCDNOut）	GB	以函数计算作为CDN源站，CDN回源时所产生的流量。按一天或一个月粒度统计求和。

地域维度指标

您可以登录函数计算控制台，在左侧导航栏选择**高级功能 > 监控大盘**，查看地域维度的指标情况。

地域维度指标是您对某一地域内函数计算资源整体使用情况的监控度量，包括以下指标项：

指标类型	指标名称	单位	描述
函数执行	调用次数（RegionTotalInvocations）	次	在某一地域内调用函数的总请求次数。按1分钟或1小时粒度统计求和。
错误次数	服务端错误（RegionServerErrors）	次	在某一地域内调用函数时，由于函数计算系统错误导致函数未被执行的总调用次数。按1分钟或1小时粒度统计求和。 **说明** 不包括HTTP触发器函数执行成功返回的 `5xx` 状态码。
	客户端错误（RegionClientErrors）	次	在某一地域内调用函数时，由于函数计算客户端错误导致函数未被执行，且返回 `4xx` 状态码的总调用次数。按1分钟或1小时粒度统计求和。部分状态码示例如下： • `400`：参数错误。 • `403`：没有访问权限。 • `404`：函数不存在。 • `499`：客户端连接断开。
	函数错误（RegionFunctionErrors）	次	在某一地域内调用函数时，由于函数本身原因导致函数执行失败的总请求次数。按1分钟或1小时粒度统计求和。
流控错误	并发实例超上限（RegionThrottles）	次	在某一地域内调用函数时，由于函数并发实例超上限导致函数执行失败，且返回 `429` 状态码的总调用次数。按1分钟或1小时粒度统计求和。
	实例总数超上限（RegionResourceThrottles）	次	在某一地域内调用函数时，由于实例总数超上限导致函数执行失败，且返回 `503` 状态码的总调用次数。按1分钟或1小时粒度统计求和。
按量实例数	按量实例上限（RegionConcurrencyLimit）	个	当前账号在某一地域内按量实例上限数，默认为300。
	按量实例数（RegionConcurrentCount）	个	在某一地域内调用函数时，实际并发占用的按量实例数。按1分钟或1小时粒度统计求和。
预留实例数	预留实例数（RegionProvisionedCurrentInstance）	个	当前账号在某一地域内所有函数的预留实例总数。

服务维度指标

您可以登录函数计算控制台，在左侧导航栏选择**高级功能 > 监控大盘**，然后在**服务名称**列表单击目标服务名称，查看服务维度的指标情况。

服务维度指标是您对某个指定服务内资源的使用情况进行监控度量，不仅可以从服务角度度量，还可以从服务版本角度和服务别名角度进行度量。服务角度、服务版本角度和服务别名角度都属于服务维度。服务维度的指标度量信息包括以下指标项：

> **说明** 服务版本角度或服务别名角度的指标名称前缀为 `ServiceQualifier`，例如函数总调用为 `ServiceQualifierTotalInvocations`。

指标类型	指标名称	单位	描述
函数执行	函数总调用（ServiceTotalInvocations）	次	某个指定服务内函数的总调用次数。按1分钟或1小时粒度统计求和。
错误次数	服务端错误（ServiceServerErrors）	次	在调用某个指定服务内的函数时，由于函数计算的系统错误导致函数未被执行的总调用次数。按1分钟或1小时粒度统计求和。 **说明** 不包括HTTP触发器函数执行成功返回的 `5xx` 状态码。
	客户端错误（ServiceClientErrors）	次	在调用某个指定服务内的函数时，由于函数计算客户端错误导致函数未被执行，且返回 `4xx` 状态码的总调用次数。按1分钟或1小时粒度统计求和。部分状态码如下： • `400`：参数错误。 • `403`：没有访问权限。 • `404`：函数不存在。 • `499`：客户端断开连接。
	函数错误（ServiceFunctionErrors）	次	在调用某个指定服务内的函数时，由于函数自身原因导致函数执行失败的总调用次数。按1分钟或1小时粒度统计求和。
流控错误	并发实例超上限（ServiceThrottles）	次	在调用某个指定服务内的函数时，由于函数并发实例超上限，且返回 `429` 状态码的总请求次数。按1分钟或1小时粒度统计求和。
	实例总数超上限（ServiceResourceThrottles）	次	在调用某个指定服务内的函数时，由于实例总数超上限，且返回 `503` 状态码的总请求次数。按1分钟或1小时粒度统计求和。
地域按量实例数	地域按量实例上限（RegionConcurrencyLimit）	个	当前账号在某一地域内按量实例上限数，默认为300。
	地域已使用按量实例数（RegionConcurrentCount）	个	在某一地域内调用函数时，实际并发占用的按量实例数。按1分钟或1小时粒度统计求和。
预留实例数	预留实例数（ServiceProvisionedCurrentInstance）	个	当前服务下所有函数的预留实例总数。

指标类型	指标名称	单位	描述
异步调用处理情况	异步请求入队（ServiceEnqueueCount）	个	异步调用中，到达函数计算的请求数。当入队请求数远大于请求处理完成数，有请求积压，请调整函数弹性管理（含预留模式）上限或联系我们进行处理。
	异步请求处理完成（ServiceDequeueCount）	个	异步调用中，函数计算处理完成的请求数。当入队请求数远大于请求处理完成数，有请求积压，请调整函数弹性管理（含预留模式）上限或联系我们进行处理。
异步消息处理延时	异步请求平均处理延时（ServiceAsyncMessageLatencyAvg）	毫秒	指定的时间范围内，所有异步调用请求从入队到开始处理的平均时延。当这个值过大时，表示有请求积压，请调整函数弹性管理（含预留模式）上限或联系我们进行处理。

函数维度指标

您可以登录函数计算控制台，在左侧导航栏选择**高级功能 > 监控大盘**，然后在**服务名称**列表单击目标服务名称。然后在**服务级别监控**页面的**函数名称**列表单击目标函数，查看函数维度的指标情况。

函数维度指标是您对某个指定函数资源的使用情况进行监控度量，不仅可以从函数角度度量，还可以从服务版本下函数角度和服务别名下函数角度进行度量。函数角度、服务版本下函数角度和服务版本下别名角度都属于函数维度。函数维度的指标度量信息包括以下指标项：

> **说明**
>
> - 服务版本下函数角度或服务别名下函数角度的指标名称的前缀为 `FunctionQualifier` ，例如调用次数为 `FunctionQualifierTotalInvocations` 。
> - 函数维度内CPU使用情况、内存使用情况和网络流量均需要开启实例级别指标后才可以进行监控度量。关于实例级别指标的详细信息，请参见实例级别指标。

指标类型	指标名称	单位	描述
调用次数	函数总调用（FunctionTotalInvocations）	次	基于预留和按量模式统计的函数总调用次数。按1分钟或1小时粒度统计求和。
	基于预留模式的调用（FunctionProvisionInvocations）	次	基于预留模式统计的函数总调用次数。按1分钟或1小时粒度统计求和。
	服务端错误（FunctionServerErrors）	次	在调用某个指定函数时，由于函数计算系统原因导致函数未被执行的总调用次数。按1分钟或1小时粒度统计求和。 **说明** 不包括HTTP触发器函数执行成功返回的 `5xx` 状态码。

指标类型 错误次数	指标名称	单位	描述
	客户端错误（FunctionClientErrors）	次	在调用某个指定函数时，由于函数计算客户端原因导致函数未被执行，且返回 `4xx` 状态码的总调用次数，按1分钟或1小时粒度统计求和。 部分状态码示例如下所示： • `400`：参数错误。 • `403`：没有访问权限。 • `404`：函数不存在。 • `499`：客户端断开连接。
	函数错误（FunctionFunctionErrors）	次	在调用某个指定函数时，由于函数自身原因导致函数调用失败的次数。按1分钟或1小时粒度统计求和。
流控错误	并发实例超上限（FunctionConcurrencyThrottles）	次	在调用函数时，由于函数并发实例超上限导致函数调用失败，且返回 `429` 状态码的总调用次数。按1分钟或1小时粒度统计求和。
	实例总数超上限（FunctionResourceThrottles）	次	在调用函数时，由于函数实例总数超上限导致函数执行失败，且返回 `503` 状态码的总调用次数。按1分钟或1小时粒度统计求和。
端到端延时	平均延时（FunctionLatencyAvg）	毫秒	在调用时，函数执行请求从抵达函数计算系统开始到离开函数计算系统所消耗的时间，且包含平台消耗的时间。按1分钟或1小时粒度统计求平均时间。
	最大延时（FunctionLatencyMax）	毫秒	在调用函数时，函数执行请求从抵达函数计算系统开始到离开函数计算系统所消耗的时间，且包含平台消耗的时间。按1分钟或1小时粒度统计求和。
单实例多请求数	并发请求数（FunctionConcurrentRequests）	个	在调用函数时，函数执行实例中并发执行的请求个数。按1分钟或1小时粒度统计求和。 **说明** 如果不开启单实例多请求，则默认单实例并发执行单请求。如需展示该指标，请开启实例级别指标。关于单实例多并发和实例级别指标的详细信息，请参见单实例多并发简介和实例级别指标。
	函数已使用按量实例数（FunctionOndemandActiveInstance）	个	在调用函数时，函数执行实际占用的按量实例数。
函数预留实例数	函数预留实例数（FunctionProvisionedCurrentInstance）	个	在调用函数时，函数执行实际占用的预留实例数。

指标类型	指标名称	单位	描述
CPU使用情况	CPU配额（FunctionCPUQuotaPercent）	%	在调用函数时，函数的CPU配额。按1分钟或1小时粒度统计求和。函数内存和CPU的对应关系是： • 按量模式：3 GB内存对应2 vCPU。 • 预留模式：2 GB内存对应1 vCPU。
	CPU使用情况（FunctionCPUPercent）	%	在调用函数时，函数的CPU使用率，表示实际使用的CPU的核数，例如100%代表1核。函数所有实例按1分钟或1小时粒度统计求和。
内存使用情况	内存配额（FunctionMemoryLimitMB）	MB	在调用函数时，函数可使用的内存上限。如果函数实际消耗内存超过此上限，则会出现内存溢出OOM错误。函数所有实例按1分钟或1小时粒度取最大值。
	已使用内存（FunctionMaxMemoryUsage）	MB	在调用函数时，函数执行所消耗的内存，表示函数实际消耗的内存。函数所有实例按1分钟或1小时粒度取最大值。
网络流量	入网流量（FunctionRXBytesPerSec）	kbps	在调用函数时，函数执行在单位时间内的入网流量。函数所有实例按1分钟或1小时粒度统计求和。
	出网流量（FunctionTXBytesPerSec）	kbps	在调用函数时，函数执行在单位时间内的出网流量。函数所有实例按1分钟或1小时粒度统计求和。
异步调用处理情况	异步请求入队（FunctionEnqueueCount）	个	在调用函数时，函数异步调用时，入队请求个数。按1分钟或1小时粒度统计求和。
	异步请求处理完成（FunctionDequeueCount）	个	在调用函数时，函数异步调用时，处理完成的总请求个数。按1分钟或1小时粒度统计求和。 **说明** 当异步请求处理完成数远小于异步请求入队数时，将导致消息积压，请调整函数弹性管理（含预留模式）或联系我们进行处理。
异步消息处理延时	平均时间（FunctionAsyncMessageLatencyAvg）	毫秒	函数异步调用时，异步调用消息从入队到开始处理的时延。按1分钟或1小时粒度统计求平均值。
	最大时间（FunctionAsyncMessageLatencyMax）	毫秒	函数异步调用时，异步调用消息从入队到开始处理的时延。按1分钟或1小时粒度统计求和。
	超时丢弃（FunctionAsyncEventExpiredDropped）	次	函数异步调用配置Destination时，丢弃的函数执行超时的总请求次数。按1分钟或1小时粒度统计求和。

指标类型	指标名称	单位	描述
异步调用触发事件	目标触发失败（FunctionDestinationErrors）	次	函数异步调用配置Destination时，函数执行中触发目标失败的请求数。按1分钟或1小时粒度统计求和。
	目标触发成功（FunctionDestinationSucceed）	次	函数异步调用配置Destination时，函数执行中触发目标成功的请求数。按1分钟或1小时粒度统计求和。
资源使用量(MB*ms)	使用量（FunctionCost）	MB*毫秒	在调用某个指定版本或别名服务内的所有函数时，函数消耗的资源，即函数内存×函数执行时间。按1分钟或1小时粒度统计求和。
异步请求积压数	积压数	个	函数异步调用时，入队请求中等待处理或处理中的总请求个数。按1分钟或1小时粒度统计求和。 **说明** 当异步请求积压数大于0时，请调整函数弹性管理（含预留模式）或联系我们进行处理。

更多信息

关于如何调用云监控的API查看监控详情的操作，请参见监控数据。

2.监控数据

本文介绍如何通过云监控的API调取函数计算的监控数据。如果您需要调取函数计算的监控数据，您可以通过API接口的相关请求参数调取，例如Project、StartTime、EndTime、Dimensions、Period、Metric。
关于API接口说明，详情请参见API概览。

Project

函数计算监控服务指标项的数据都使用相同的Project名称：acs_fc。

使用Java SDK设置代码示例如下：

```
QueryMetricRequest request = new QueryMetricRequest();
request.setProject("acs_fc");
```

StartTime和EndTime

云监控的时间参数取值范围使用左开右闭的形式，即（StartTime, EndTime]，处于边界StartTime的数据不会被获取，而处于边界EndTime的数据会被查询到。

> **说明** 云监控数据保留时间为31天，设置的StartTime和EndTime的时间间距不能大于31天，31天前的数据是查询不到的。

其他时间参数信息，详情请参见API概览。

使用Java SDK设置代码示例如下：

```
request.setStartTime("2017-04-26 08:00:00");
request.setEndTime("2017-04-26 09:00:00");
```

Dimensions

函数计算监控服务根据函数计算资源结构和使用场景，将监控指标分为地域维度、服务维度和函数维度。不同的维度使用的Dimensions参数不同。

- 地域维度数据的Dimensions设置如下：

  ```
  {"region": "${your_region}"}
  ```

- 服务维度数据的Dimensions设置如下：

  ```
  {"region": "${your_region}", "serviceName": "${your_serviceName}"}
  ```

- 函数维度数据的Dimensions设置如下：

  ```
  {"region": "${your_region}", "serviceName": "${your_serviceName}", "functionName": "${your_functionName}"}
  ```

> **说明** Dimenisons是一个JSON字符串，函数计算监控指标的Dimensions只有一对Key-Value。使用Java SDK设置代码示例如下：
>
> ```
> request.setDimensions("{\"region\":\"your_region\"}");
> ```

Period

函数计算监控指标的聚合粒度均为60s。

使用Java SDK设置代码示例如下：

```
request.setPeriod("60");
```

Metric

使用Java SDK设置代码示例如下：

```
request.setMetric("your_metric");
```

函数计算监控指标参考手册中详细介绍的各项指标项，对应的Metric名称如下表：

指标维度	Metric	指标项对应名称
地域	RegionTotalInvocations	TotalInvocations
	RegionBillableInvocations	BillableInvocations
	RegionThrottles	Throttles
	RegionClientErrors	ClientErrors
	RegionServerErrors	ServerErrors
	RegionBillableInvocationsRate	BillableInvocations占比
	RegionThrottlesRate	Throttles占比
	RegionClientErrorsRate	ClientErrors占比
	RegionServerErrorsRate	ServerErrors占比
服务	ServiceTotalInvocations	TotalInvocations
	ServiceBillableInvocations	BillableInvocations
	ServiceThrottles	Throttles
	ServiceClientErrors	ClientErrors
	ServiceServerErrors	ServerErrors
	ServiceBillableInvocationsRate	BillableInvocations占比
	ServiceThrottlesRate	Throttles占比
	ServiceClientErrorsRate	ClientErrors占比
	ServiceServerErrorsRate	ServerErrors占比
	FunctionTotalInvocations	TotalInvocations

指标维度	Metric	指标项对应名称
函数	FunctionBillableInvocations	BillableInvocations
	FunctionThrottles	Throttles
	FunctionFunctionErrors	FunctionErrors
	FunctionClientErrors	ClientErrors
	FunctionServerErrors	ServerErrors
	FunctionBillableInvocationsRate	BillableInvocations占比
	FunctionThrottlesRate	Throttles占比
	FunctionFunctionErrorsRate	FunctionErrors占比
	FunctionClientErrorsRate	ClientErrors占比
	FunctionServerErrorsRate	ServerErrors占比
	FunctionAvgDuration	平均Duration
	FunctionMaxMemoryUsage	最大内存使用

使用示例

*pom.xml*示例如下：

```
...
    <dependencies>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-core</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-cms</artifactId>
            <version>5.0.1</version>
        </dependency>
    </dependencies>
...
```

代码示例如下：

```
import com.alibaba.fastjson.JSONObject;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.cms.model.v20170301.QueryMetricListRequest;
import com.aliyuncs.cms.model.v20170301.QueryMetricListResponse;
import com.aliyuncs.exceptions.ClientException;
```

```
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
public class MonitorService {
    public static void main(String[] args) {
        IClientProfile profile = DefaultProfile.getProfile("cn-hangzhou", "<your_access_key
_id>", "<your_access_key_secret>");
        IAcsClient client = new DefaultAcsClient(profile);
        QueryMetricListRequest request = new QueryMetricListRequest();
        request.setProject("acs_fc");
        request.setPeriod("60");
        request.setStartTime("2017-04-26 16:20:00");
        request.setEndTime("2017-04-26 16:30:00");
        request.setAcceptFormat(FormatType.JSON);
        try {
            // Region维度JSONObject dim = new JSONObject();
            request.setMetric("RegionTotalInvocations");  // 选择metric
            dim.put("region", "<your_region>");  // 如: cn-shanghai
            request.setDimensions(dim.toJSONString());
            QueryMetricListResponse response = client.getAcsResponse(request);
            System.out.println(response.getCode());
            System.out.println(response.getMessage());
            System.out.println(response.getRequestId());
            System.out.println(response.getDatapoints());
            // Service维度dim = new JSONObject();
            request.setMetric("ServiceTotalInvocations");  // 选择metric
            dim.put("region", "<your_region>");
            dim.put("serviceName", "<your_service_name>");
            request.setDimensions(dim.toJSONString());
            response = client.getAcsResponse(request);
            System.out.println(response.getCode());
            System.out.println(response.getMessage());
            System.out.println(response.getRequestId());
            System.out.println(response.getDatapoints());
            // Function维度dim = new JSONObject();
            request.setMetric("FunctionTotalInvocations");  // 选择metric
            dim.put("region", "<your_region>");
            dim.put("serviceName", "<your_service_name>");
            dim.put("functionName", "<your_function_name>");
            request.setDimensions(dim.toJSONString());
            response = client.getAcsResponse(request);
            System.out.println(response.getCode());
            System.out.println(response.getMessage());
            System.out.println(response.getRequestId());
            System.out.println(response.getDatapoints());
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            e.printStackTrace();
        }
    }
}
```

3.实例级别指标

函数计算提供实例级别指标，通过实例级别指标您可以查看CPU使用情况、内存使用情况、实例网络情况和实例内请求数等核心指标信息。本文介绍实例级别指标的背景信息、定义、指标信息和配置方式。

背景信息

函数计算是事件驱动的全托管计算服务，您无需维护计算集群。但是您在业务代码开发到正常运行的过程中可能会遇到以下场景：

- 在CPU密集型场景中如何查看CPU的具体使用量。
- 使用单实例多并发时，如何设置合适的单实例并发数。
- 当函数在执行过程中失败时，确认函数执行失败原因，例如代码异常或函数实例性能异常导致执行失败。

函数计算推出实例级别指标功能，可以帮助您解决以上遇到的问题以及了解函数计算各个实例的健康状态。

什么是实例级别指标

实例级别指标是函数实例维度的性能指标，对函数实例进行实时监控和性能数据采集，并进行可视化展示，为您提供函数实例端到端的监控排查路径。

实例级别指标可通过以下维度进行呈现：

- 函数维度或函数Qualifier维度：指以函数维度进行的聚合，例如函数A同时有两个实例在执行，那么函数维度的CPU指标就是这两个实例中的CPU使用最大值。
- 实例维度：具体的某个特定函数实例的指标。

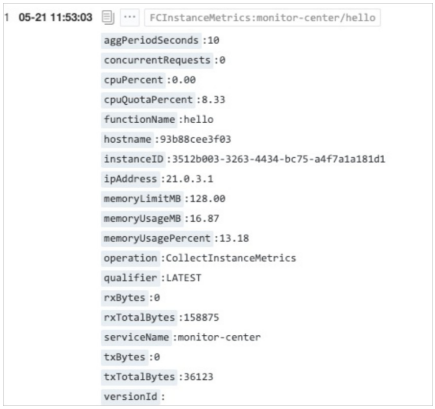
> **说明**
>
> - Qualifier指调用函数时传入的版本信息。取值即可以是版本号，也可以是别名。
> - 实例由函数计算系统动态创建与回收的，每个实例只会存在一小段时间，且您无法对实例进行操作。

指标信息

开启实例级别指标功能后，系统会收集函数执行的指标信息。您可以通过以下方式查看实例级别指标信息：

- 监控中心：函数计算的监控中心内置了实例级别可视化大盘。您可以登录函数计算控制台，在**监控中心**页面的可视化大盘中查看以下信息：
  - 函数维度实例的指标信息。
  - 所有实例的指标信息。
  - 指定实例的指标信息。
- 日志服务：函数计算会将实例指标信息导入到您的日志服务SLS内，您可以通过SLS分析能力及ARMS日志接入能力创建自定义的可视化大盘。详细信息，请参见查询和分析日志及开始使用日志监控。

每个实例的实例级别指标信息每隔一段时间会记录一次信息，并将该信息记录在日志内，具体形式如下：



实例级别指标会采集以下指标信息：

名称	描述	示例值
cpuPercent	CPU使用率。代表实际使用的CPU的核数，可能会超过100%。	120%
cpuQuotaPercent	实例预期的CPU的最大值，函数内存和CPU的对应关系是： • 按量模式：3 GB内存对应2 vCPU。 • 预留模式：2 GB内存对应1 vCPU。	50%
memoryUsageMB	实例消耗内存。单位：MB。	16.87
memoryLimitMB	实例内存的上限。单位：MB。	1024
rxBytes	记录日志的时间间隔内，函数实例接收的流量。单位：Byte。	158
txBytes	记录日志的时间间隔内，函数实例发送的流量。单位：Byte。	1598
rxTotalBytes	自函数实例启动开始，函数实例接收的流量。单位：Byte。	158875
txTotalBytes	自函数实例启动开始，函数实例发送的流量。单位：Byte。	36123
concurrentRequests	当前实例的请求数。	10
hostname	函数实例的Hostname。	36123

说明

- cpuQuotaPercent是理论值，cpuPercent值有可能超过cpuQuotaPercent值，此时当前函数实例抢占了同宿主机下其他函数实例的资源。
- 函数实例和系统模块通信，会产生少量流量，所以即使函数内没有任何网络访问也会有少量收发流量。
- 函数实例流量仅代表此实例的网络输入输出流量，不区分公网或私网流量，无法根据此监控图推算流量费用。

配置实例级别指标

1. 登录函数计算控制台。
2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。
4. 在**服务列表**页面，找到目标服务。在其**操作**列，单击**配置**。
5. 在编辑目标服务页面的日志配置区域的**实例级别指标**后，选中**启用**，然后单击**保存**。

> **说明** 如果您在创建服务时未启用日志功能，您需在**日志配置**区域启用日志功能和实例级别指标。

执行结果

成功开启实例级别指标后，在**监控中心**页面您可以查看实例级别指标信息，例如CPU使用情况、内存使用情况、实例网络情况和实例内请求数等核心指标。

4.链路追踪

4.1. 链路追踪简介

本文介绍函数计算集成链路追踪功能的背景、使用场景、核心功能、采样规则配置和开启方式。

背景介绍

阿里云链路追踪服务（Tracing Analysis ）基于OpenTracing标准，全面兼容开源社区，为分布式应用的开发者提供了完整的分布式调用链查询和诊断，分布式拓扑动态发现，应用性能实时汇总等功能。

函数计算与链路追踪集成，支持使用Jaeger上传链路信息，使您能够跟踪函数的执行，帮助您快速分析和诊断Serverless架构下的性能瓶颈，提高Serverless场景的开发诊断效率。

使用场景

当函数计算与链路追踪集成后，您可以记录请求在函数计算的耗时时间、查看函数的冷启动时间、记录函数内部时间的消耗等。链路追踪功能可以帮助您排查以下问题：

- 函数执行总超时，需要定位函数的性能瓶颈。
- 函数执行时间很短，但是端对端时延很长时，需要定位相关原因。
- 分布式系统中，请求横跨多个云服务，需要分析和诊断函数的性能瓶颈。

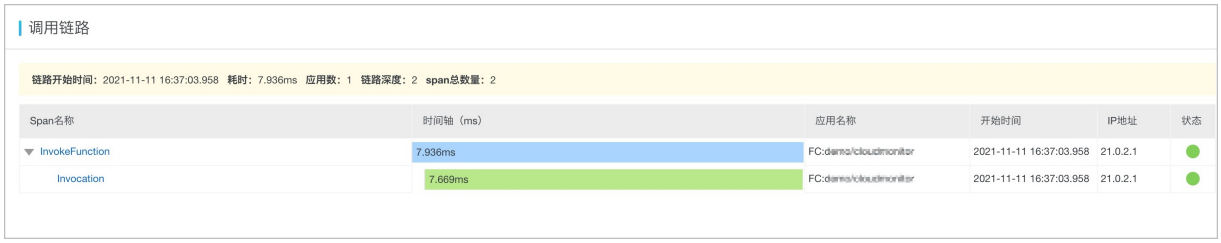
核心功能

函数计算的链路追踪功能可以串联整个调用链，包含以下核心功能：

- 自动记录函数计算内部关键步骤耗时时间。更多信息，请参见自动记录函数计算内部关键步骤耗时。
- 串联上游服务：如果请求Header中带有链路上下文信息，则函数计算会根据上下文创建子链路。更多信息，请参见串联上游服务。
- 串联下游服务：函数计算会将链路上下文传入到函数Context参数中，帮助您追踪函数内部调用链路。更多信息，请参见串联下游服务。
- 查看应用拓扑。
- 查看错误接口执行情况，定位错误原因。

自动记录函数计算内部关键步骤耗时

服务开启链路追踪功能后，您默认可以看到以下调用链。



Span名称说明如下：

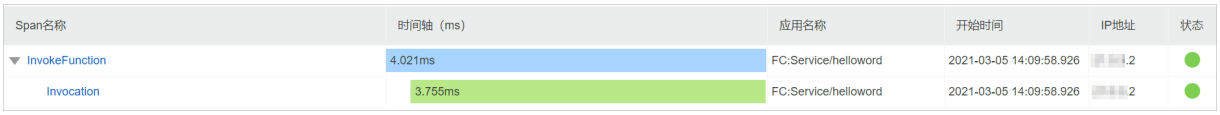
- InvokeFunction：当前请求在函数计算的总停留时间。
- ColdStart：函数系统层冷启动的时间，冷启动不是每次调用都出现，只在重新申请执行环境时会进行冷启动。
  - PrepareCode：函数下载代码或下载自定义镜像的时间，如果PrepareCode时间过长，您可以适当精简代码包来缩短准备代码的时间。

  - **RuntimeInitialization**：执行环境启动的时间，包含启动实例的时间、实例健康检查时间。在自定义运行时和自定义镜像中，如果RuntimeInitialization执行时间过长，需要检查一下对应的HTTP Server和镜像的启动行为。

- **Initializer**：初始化函数的执行时间，初始化函数当且仅当容器冷启动的时候才会被执行。
- **Invocation**：函数的执行时间，您可以在函数中获取到Invocation的上下文，详细记录函数调用中的耗时。

应用名称：函数计算生成的应用命名方式为 `FC:ServiceName/FunctionName`

当请求没有遇到冷启动时，链路中没有冷启动时间和Initializer的时间。调用链如下所示。



串联上游服务

服务开启链路追踪后，如果请求Header中带有链路上下文信息SpanContext，则函数计算会根据上下文创建子链路。

函数计算识别的链路上下文请求头如下：

- `x-fc-tracing-opentracing-span-context`：用于传递链路上下文信息SpanContext信息。
- `x-fc-tracing-opentracing-span-context-baggage-`：用于传递跨上下文的Baggage信息。如果有多个Baggage则需要上传多个Header，例如 `x-fc-tracing-opentracing-span-context-baggage-key1: val1`，`x-fc-tracing-opentracing-span-context-baggage-key2: val2`。

在调用函数时添加链路上下文信息Header即可注入上下文信息。

以Node.js为例：

```
'use strict';
const FCClient = require('@alicloud/fc2');
var client = new FCClient('<account id>', {
    accessKeyID: '<access key id>',
    accessKeySecret: '<access key secret>',
    region: 'cn-shanghai',
});
var serviceName = '<service name>';
var funcName = '<function name>';
async function test() {
    try {
        // 注入Span Context Header信息。
        var headers = {
            'x-fc-tracing-opentracing-span-context': '124ed43254b54966:124ed43254b5****:0:1
',
            'x-fc-tracing-opentracing-span-context-baggage-key': 'val'
        };
        var resp = await client.invokeFunction(serviceName, functionName, 'event', headers
= headers);
    } catch (err) {
        console.error(err);
    }
}
```

串联下游服务

函数计算会将链路上下文传入到函数中，帮助追踪函数内部的调用链路。

- 对于内置Runtime，可以通过 `context.tracing` 获取链路上下文信息。
- 对于Custom Runtime或Custom Container，可以通过请求Header获取函数计算链路上下文信息。
  - `x-fc-tracing-opentracing-span-context`：函数计算InvokeFunction的链路上下文，函数内基于此上下文创建追踪段。
  - `x-fc-tracing-opentracing-span-baggages`：经过Base64编码的跨上下文Baggage。
  - `x-fc-tracing-jaeger-endpoint`：Jaeger的Server端地址，您直接将函数中的追踪段上传至此地址。

`context.tracing` 结构示例如下：

```
{
  "openTracingSpanContext": "5f22f355044a957a:5708f3a95a4ed10:5f22f355044a****:1",
  "openTracingSpanBaggages": {
    "key1": "val1",
    "key2": "val2"
  },
  "jaegerEndpoint": "http://tracing-analysis-dc-zb-internal.aliyuncs.com/adapt_fcfc@fcfc@fcfc/api/traces"
}
```

函数中获取SpanContext示例如下：

- Node.js：

```
exports.handler = (event, context, callback) => {
  var params = {
        openTracingSpanContext: context.tracing.openTracingSpanContext,
        openTracingSpanBaggages:context.tracing.openTracingSpanBaggages,
        // jaegerEndpoint is confidential, do not print it out easily
        // jaegerEndpoint:context.tracing.jaegerEndpoint
    }
  console.log('tracing params',params)
  callback(null,'success');
}
```

- PHP：

```
function handler($event, $ctx) {
    $logger = $GLOBALS['fcLogger'];
    $openTracingSpanContext = $ctx['tracing']['openTracingSpanContext'];
    $openTracingSpanBaggages = $ctx['tracing']['openTracingSpanBaggages'];
    // jaegerEndpoint is confidential, do not print it out easily
    $jaegerEndpoint = $ctx['tracing']['jaegerEndpoint'];
    $logger->info($openTracingSpanContext);
    $logger->info($openTracingSpanBaggages['key1']);
  return 'success';
}
```

- Java：

```
package example;
import com.aliyun.fc.runtime.Context;
import com.aliyun.fc.runtime.StreamRequestHandler;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
public class HelloFC implements StreamRequestHandler {
    public void handleRequest(
            InputStream inputStream, OutputStream outputStream, Context context) throws I
OException {
        String spanContext = context.getTracing().getOpenTracingSpanContext();
        String jaegerEndpoint = context.getTracing().getJaegerEndpoint();
        String spanBaggage = context.getTracing().getOpenTracingSpanBaggages().get("key1"
);
        outputStream.write(new String("success").getBytes());
    }
}
```

- Custom Runtime或Custom Container获取Header即可，以Golang语言为例：

```
  spanContext := req.Header.Get('x-fc-tracing-opentracing-span-context')
  spanBaggages := req.Header.Get('x-fc-tracing-opentracing-span-baggages')
  jaegerEndpoint := req.Header.Get('x-fc-tracing-jaeger-endpoint')
```

自定义采样规则

如果您需要自定义采样规则，您可以登录链路追踪控制台设置远程采样规则。更多信息，请参见使用Jaeger进行远程采样策略配置。配置完成后，函数计算会使用您设置的远程采样规则采样。

函数计算在链路追踪中对应的服务名称为**fc-tracing**，默认使用的采样规则为**RatelimitingSampler**，以每秒1个请求的速率采样。

函数计算使用的默认采样规则如下：

```
{
  "default_strategy": {
    "type": "ratelimiting",
    "param": 1,
  }
}
```

开启方式

关于链路追踪功能的开启方式，请参见配置链路追踪。

4.2. 配置链路追踪

本文介绍在函数计算控制台配置链路追踪功能、使用自定义链路追踪接入点及查看函数调用链信息的操作步骤。

背景信息

链路追踪简介

创建服务时配置链路追踪

1. 登录函数计算控制台。

2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。
4. 在**服务列表**页面，单击**创建服务**。

5. 在**创建服务**面板，设置服务名称，在**链路追踪功能**后选中**启用**，然后单击**确定**。

> **说明** 选中**开启链路追踪**后，该服务下的所有函数均会开启链路追踪。

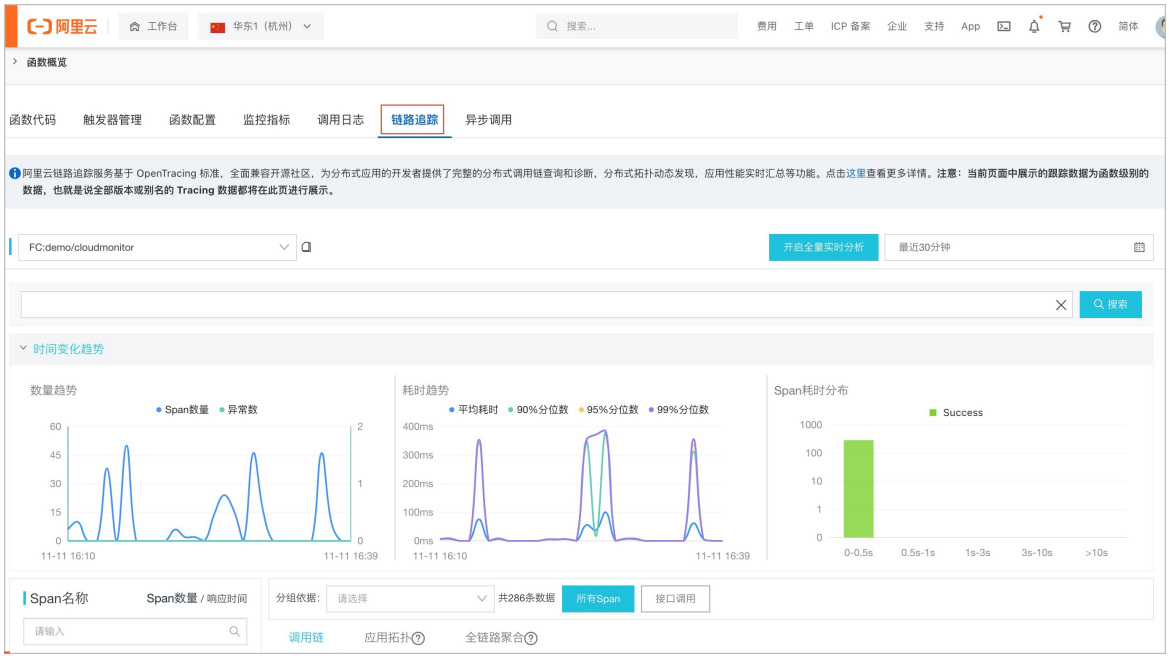
服务创建完成后配置链路追踪

1. 登录函数计算控制台。
2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。
4. 在**服务列表**页面，找到目标服务。在其**操作**列，单击**配置**。
5. 在编辑服务页面的**链路追踪配置**区域，选中**启用**，然后单击**保存**。

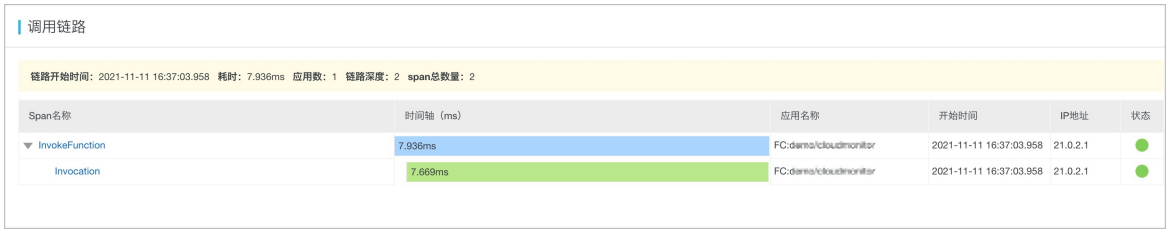
查看函数调用链信息

当您的服务开启链路追踪功能后，函数计算会自动记录该服务下所有函数的调用链信息。

1. 登录函数计算控制台。
2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。
4. 在**服务列表**页面，单击目标服务名称。
5. 在目标服务页面，单击目标函数名称。
6. 在函数详情页面，单击**链路追踪**，查看函数调用链信息。更多信息，请参见调用链分析。



7. 在**链路追踪**页签的**调用链**页签单击具体调用链，可查看详细信息。



参数说明如下：

> **说明** 当请求没有遇到冷启动时，链路中没有冷启动时间和Initializer的时间。

- **TracingService/TracingFunction**：当前请求在函数计算的总停留时间，默认命名方式为`servicName/functionName`。
- **ColdStart**：函数系统层冷启动的时间，冷启动不是每次调用都出现，只在重新申请执行环境时出现。
  - **PrepareCode**：函数下载代码或下载自定义镜像的时间，如果PrepareCode时间过长，您可以适当精简代码包来缩短准备代码的时间。

- RuntimeInitialization：执行环境启动的时间，包含启动实例的时间、实例健康检查时间。在自定义运行时和自定义镜像中，如果RuntimeInitialization执行时间过长，需要检查一下对应的HTTP Server和镜像的启动行为。
- Initializer：初始化函数的执行时间，初始化函数当且仅当容器冷启动的时候才会被执行。
- Invocation：函数的执行时间，您可以在函数中获取到Invocation的上下文，详细记录Invocation中的耗时。

4.3. 创建自定义Span

开启链路追踪后，函数计算会自动记录请求在系统侧的耗时，包含冷启动耗时、Initializer函数的耗时和函数的执行时间等。如果您还需查看函数内业务侧的耗时，例如在函数内访问RDS、NAS等服务的耗时，则可通过本文提供的创建自定义Span步骤实现。

前提条件

- 安装Serverless Devs和Docker
- 配置Serverless Devs

流程说明

您可以基于函数计算的链路创建自定义Span，将自定义的Span串联到函数计算的调用链中。函数计算的链路分析基于OpenTracing协议的Jaeger实现，提供以下两种方式创建自定义Span：

- 使用Jaeger SDK
- 使用OpenTelemetry

Jaeger和OpenTelemetry的详细信息，请分别参见Jaeger和OpenTelemetry。

两种方式的流程说明如下：

1. 在函数中引入Jaeger或OpenTelemetry依赖包。引入三方依赖的具体操作，请参见为函数安装第三方依赖。
2. 在函数中获取函数计算调用链的Span Context，根据函数计算的Span Context，创建自定义Span，在需要记录耗时的代码片段前后添加埋点。
3. 完成后，您可以在函数详情页面的**链路追踪**页签或链路追踪控制台查看调用链信息。

使用Jaeger SDK

本文以Node.js运行时为例，介绍如何通过Serverless Devs创建自定义Span，创建并部署函数的步骤。

> **说明** 您可以使用Serverless Devs安装依赖包，并打包部署到函数计算，您也可以通过其他方式打包并部署。Serverless Devs的更多信息，请参见什么是Serverless Devs。

1. 创建代码目录。

```
mkdir jaeger-demo
```

2. 初始化Node.js运行时模板。
   i. 执行以下命令，进入代码目录：

```
cd jaeger-demo
```

ii. 在代码目录中执行以下命令，初始化Node.js 12的项目：

```
s init devsapp/start-fc-event-nodejs12
```

输出示例：

```
Serverless Awesome: https://github.com/Serverless-Devs/package-awesome
Please input your project name (init dir) (start-fc-event-nodejs12)
```

iii. 设置项目名称，然后按回车键。

Serverless Devs已默认为您生成一个名为start-fc-event-nodejs12的项目，您可按需修改项目名称，本文以默认的项目名称为例。
输出示例：

```
Serverless Awesome: https://github.com/Serverless-Devs/package-awesome
Please input your project name (init dir) start-fc-event-nodejs12
file decompression completed
please select credential alias (Use arrow keys)
❯ default
```

iv. 按需选择别名，然后按回车键。

v. 按需选择是否部署该项目。

由于本示例介绍的项目还需安装依赖，所以在 **是否立即部署该项目？**`(Y/n)` 后设置 `n` 选择不部署该项目。

3. 执行以下命令，进行项目的代码目录：

```
cd start-fc-event-nodejs12/code
```

4. 在代码目录中，执行以下命令，初始化*package.json*文件。

```
npm init -y
```

输出示例：

```
Wrote to /test/jjyy/start-fc-event-nodejs12/code/package.json:
{
  "name": "code",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

5. 执行以下命令，安装Jaeger依赖。

```
npm i jaeger-client --save
```

输出示例：

```
added 15 packages in 1s
```

6. 编辑函数代码。

   代码内容如下所示：

```
var initTracer = require('jaeger-client').initTracer;
var spanContext = require('jaeger-client').SpanContext;
module.exports.handler = function(event, context, callback)
{
    console.log('invoking...',context.tracing);
    var config = {
        serviceName: 'e2e-test',
        reporter: {
            // Provide the traces endpoint; this forces the client to connect directly
to the Collector and send
            // spans over HTTP
            collectorEndpoint: context.tracing.jaegerEndpoint,
            flushIntervalMs: 10,
        },
        sampler: {
            type: "const",
            param: 1
        },
    };
    var options = {
        tags: {
            'version': 'fc-e2e-tags',
        },
    };
    var tracer = initTracer(config, options);
    var invokSpanContextStr = context.tracing.openTracingSpanContext;
    console.log('spanContext', invokSpanContextStr);
    var invokSpanContext = spanContext.fromString(invokSpanContextStr);
    var span = tracer.startSpan("CustomSpan", {
        childOf: invokSpanContext
    });
    span.finish();
    var params = {
        openTracingSpanContext: context.tracing.openTracingSpanContext,
        openTracingSpanBaggages: context.tracing.openTracingSpanBaggages,
        jaegerEndpoint: context.tracing.jaegerEndpoint
    }
   callback(null,'success')
}
```

7. 安装依赖。

   i. 执行以下命令，进入项目目录：

```
cd ..
```

ii. 执行以下命令，安装依赖：

```
s build --use-docker
```

输出示例：

```
[2021-11-12T18:33:43.856] [INFO ] [S-CLI] - Start ...
[2021-11-12T18:33:44.677] [INFO ] [FC-BUILD] - Build artifact start...
[2021-11-12T18:33:44.701] [INFO ] [FC-BUILD] - Use docker for building.
[2021-11-12T18:33:44.988] [INFO ] [FC-BUILD] - Build function using image: registry
.cn-beijing.aliyuncs.com/aliyunfc/runtime-nodejs12:build-1.9.21
[2021-11-12T18:33:45.011] [INFO ] [FC-BUILD] - skip pulling image registry.cn-beiji
ng.aliyuncs.com/aliyunfc/runtime-nodejs12:build-1.9.21...
[2021-11-12T18:33:47.915] [INFO ] [FC-BUILD] - Build artifact successfully.
Tips for next step
======================
* Invoke Event Function: s local invoke
* Invoke Http Function: s local start
* Deploy Resources: s deploy
End of method: build
```

8. 执行以下命令，部署项目：

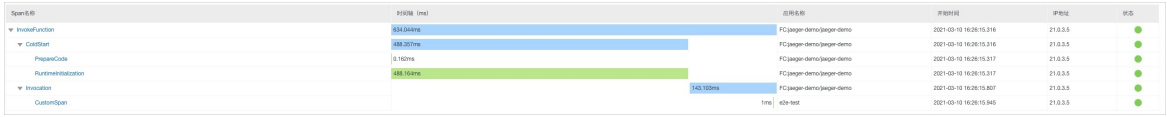
```
s deploy -y
```

输出示例：

```
[2021-11-12T18:35:26.015] [INFO ] [S-CLI] - Start ...
[2021-11-12T18:35:26.633] [INFO ] [FC-DEPLOY] - Using region: cn-hangzhou
[2021-11-12T18:35:26.634] [INFO ] [FC-DEPLOY] - Using access alias: default
[2021-11-12T18:35:26.634] [INFO ] [FC-DEPLOY] - Using accessKeyID: LTAI4G4cwJkK4Rza6xd9
****
......
 Make service fc-deploy-service success.
 Make function fc-deploy-service/event-nodejs12 success.
[2021-11-12T18:35:37.661] [INFO ] [FC-DEPLOY] - Checking Service fc-deploy-service exis
ts
[2021-11-12T18:35:37.718] [INFO ] [FC-DEPLOY] - Checking Function event-nodejs12 exists
Tips for next step
======================
* Display information of the deployed resource: s info
* Display metrics: s metrics
* Display logs: s logs
* Invoke remote function: s invoke
* Remove Service: s remove service
* Remove Function: s remove function
* Remove Trigger: s remove trigger
* Remove CustomDomain: s remove domain
fc-deploy-test:
  region:   cn-hangzhou
  service:
    name: fc-deploy-service
  function:
    name:       event-nodejs12
    runtime:    nodejs12
    handler:    index.handler
    memorySize: 128
    timeout:    60
```

9. 结果验证。

   在函数计算控制台查看调用链，可以看到您刚创建的自定义Span与函数计算的系统Span连接起来。



使用OpenTelemetry

本文以Python运行时为例，介绍如何通过Serverless Devs创建自定义Span，创建并部署函数的步骤。

> **说明** 您可以使用Serverless Devs安装依赖包，并打包部署到函数计算，您也可以通过其他方式打包并部署。Serverless Devs的更多信息，请参见什么是Serverless Devs。

1. 创建代码目录。

```
mkdir opentelemetry-demo
```

2. 进入代码目录。

```
cd opentelemey-demo
```

3. 初始化Python运行时模板。
   i. 执行以下命令，初始化Python 3的项目：

   ```
   s init devsapp/start-fc-event-python3
   ```

   输出示例：

   ```
   Serverless Awesome: https://github.com/Serverless-Devs/package-awesome
   Please input your project name (init dir) (start-fc-event-python3)
   ```

   ii. 设置项目名称，然后按回车键。

   Serverless Devs已默认为您生成一个名为start-fc-event-python3的项目，您可按需修改该名称，本文以默认的项目名称为例。
   输出示例：

   ```
   Serverless Awesome: https://github.com/Serverless-Devs/package-awesome
   Please input your project name (init dir) start-fc-event-python3
   file decompression completed
   please select credential alias (Use arrow keys)
   ❯ default
   ```

   iii. 按需选择别名，然后按回车键。

   iv. 按需选择是否部署该项目。

   由于本示例介绍的项目还需安装依赖，所以在 **是否立即部署该项目？**`(Y/n)` 后设置 `n` 选择不部署该项目。

4. 安装OpenTelemetry依赖。
   i. 执行以下命令，进入代码目录：

   ```
   cd start-fc-event-python3/code
   ```

   ii. 在代码目录中创建*requirements.txt*文件，然后编辑该文件，文件内容如下所示：

   ```
   opentelemetry-api==1.6.2
   ```

5. 编辑函数代码。

   执行以下命令，编辑函数代码：

   ```
   vim index.py
   ```

   函数代码内容如下所示：

```
# -*- coding: utf-8 -*-
import logging
import time
from opentelemetry import trace
from opentelemetry.exporter import jaeger
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchExportSpanProcessor
from opentelemetry.trace import set_span_in_context
trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)
def handler(event, context):
    logger = logging.getLogger()
    logger.info("start to init tracer")
    init_tracer(context.tracing.jaeger_endpoint)
    jaeger_span_context = context.tracing.span_context
    fc_invocation_span = get_fc_span(jaeger_span_context)
    with fc_invocation_span as parent:
        print('Hello world!')
    context = set_span_in_context(parent)
    child = tracer.start_span("child", context=context)
    # print(child)
    time.sleep(0.1)
    child.end()
    return 'hello world'
def init_tracer(endpoint):
    jaeger_exporter = jaeger.JaegerSpanExporter(
        service_name='opentelemetry-service',
        collector_endpoint=endpoint,
        transport_format='thrift'  # optional
    )
    # Create a BatchExportSpanProcessor and add the exporter to it
    span_processor = BatchExportSpanProcessor(jaeger_exporter)
    # add to the tracer
    trace.get_tracer_provider().add_span_processor(span_processor)
def get_fc_span(jaeger_span_context):
    jaeger_span_context_arr = jaeger_span_context.split(":")
    tid = int(jaeger_span_context_arr[0], 16)
    sid = int(jaeger_span_context_arr[1], 16)
    fid = int(jaeger_span_context_arr[3], 16)
    span_context = trace.SpanContext(
        trace_id=tid,
        span_id=sid,
        is_remote=True,
        trace_flags=trace.TraceFlags(fid),
        trace_state=[],
    )
    return trace.DefaultSpan(span_context)
```

注意 当您编辑好函数代码后，请执行以下命令，然后按回车键，退出编辑页面：

```
:WQ
```

6. 安装依赖。

   i. 执行以下命令，进入项目目录：

   ```
   cd ..
   ```

   ii. 执行以下命令，安装依赖：

   ```
   s build --use-docker
   ```

   输出示例：

   ```
   [2021-11-12T18:53:05.818] [INFO ] [S-CLI] - Start ...
   [2021-11-12T18:53:06.638] [INFO ] [FC-BUILD] - Build artifact start...
   [2021-11-12T18:53:06.659] [INFO ] [FC-BUILD] - Use docker for building.
   [2021-11-12T18:53:06.888] [INFO ] [FC-BUILD] - Build function using image: registry
   .cn-beijing.aliyuncs.com/aliyunfc/runtime-python3.6:build-1.9.21
   [2021-11-12T18:53:06.942] [INFO ] [FC-BUILD] - skip pulling image registry.cn-beiji
   ng.aliyuncs.com/aliyunfc/runtime-python3.6:build-1.9.21...
   [2021-11-12T18:53:10.570] [INFO ] [FC-BUILD] - Build artifact successfully.
   Tips for next step
   ======================
   * Invoke Event Function: s local invoke
   * Invoke Http Function: s local start
   * Deploy Resources: s deploy
   End of method: build
   ```

7. 部署项目。

   执行以下命令，部署项目：

   ```
   s deploy -y
   ```

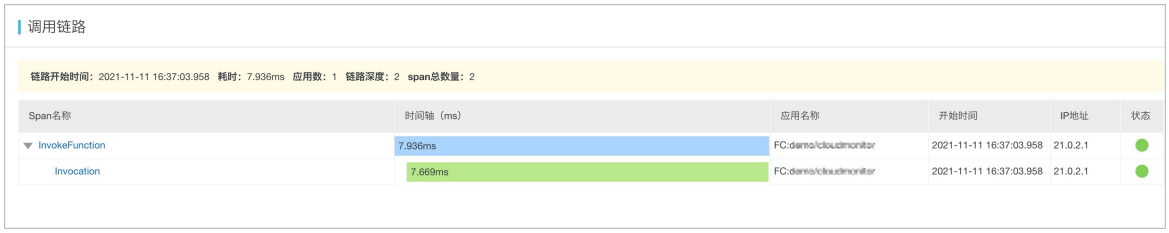
   输出示例：

   ```
   [2021-11-12T18:55:02.640] [INFO ] [S-CLI] - Start ...
   [2021-11-12T18:55:03.455] [INFO ] [FC-DEPLOY] - Using region: cn-hangzhou
   [2021-11-12T18:55:03.456] [INFO ] [FC-DEPLOY] - Using access alias: default
   [2021-11-12T18:55:03.456] [INFO ] [FC-DEPLOY] - Using accessKeyID: LTAI4G4cwJkK4Rza6xd9
   ****
   [2021-11-12T18:55:03.457] [INFO ] [FC-DEPLOY] - Using accessKeySecret: eCc0GxSpzfq1DVsp
   nqqd6nmYNN****
    Using fc deploy type: sdk, If you want to deploy with pulumi, you can [s cli fc-defaul
   t set deploy-type pulumi] to switch.
   [2021-11-12T18:55:04.142] [INFO ] [FC-DEPLOY] - Checking Service fc-deploy-service exis
   ts
   [2021-11-12T18:55:04.722] [INFO ] [FC-DEPLOY] - Checking Function event-py3 exists
   [2021-11-12T18:55:04.831] [INFO ] [FC-DEPLOY] - Fc detects that you have run build comm
   and for function: event-py3.
   [2021-11-12T18:55:04.831] [INFO ] [FC-DEPLOY] - Using codeUri: /test/jjyy/opentelemetry
   -demo/start-fc-event-python3/.s/build/artifacts/fc-deploy-service/event-py3
   [2021-11-12T18:55:04.835] [INFO ] [FC-DEPLOY] - Fc add/append some content to your orig
   in environment variables for finding dependencies generated by build command.
   {
     "LD_LIBRARY_PATH": "/code/.s/root/usr/local/lib:/code/.s/root/usr/lib:/code/.s/root/u
   sr/lib/x86_64-linux-gnu:/code/.s/root/usr/lib64:/code/.s/root/lib:/code/.s/root/lib/x86
   _64-linux-gnu:/code/.s/root/python/lib/python2.7/site-packages:/code/.s/root/python/lib
   ```

```
/python3.6/site-packages:/code:/code/lib:/usr/local/lib",
  "PATH": "/code/.s/root/usr/local/bin:/code/.s/root/usr/local/sbin:/code/.s/root/usr/b
in:/code/.s/root/usr/sbin:/code/.s/root/sbin:/code/.s/root/bin:/code:/code/node_modules
/.bin:/code/.s/python/bin:/code/.s/node_modules/.bin:/usr/local/bin:/usr/local/sbin:/us
r/bin:/usr/sbin:/sbin:/bin",
  "NODE_PATH": "/code/node_modules:/usr/local/lib/node_modules",
  "PYTHONUSERBASE": "/code/.s/python"
}
 Make service fc-deploy-service success.
 Make function fc-deploy-service/event-py3 success.
[2021-11-12T18:55:10.693] [INFO ] [FC-DEPLOY] - Checking Service fc-deploy-service exis
ts
[2021-11-12T18:55:10.737] [INFO ] [FC-DEPLOY] - Checking Function event-py3 exists
Tips for next step
======================
* Display information of the deployed resource: s info
* Display metrics: s metrics
* Display logs: s logs
* Invoke remote function: s invoke
* Remove Service: s remove service
* Remove Function: s remove function
* Remove Trigger: s remove trigger
* Remove CustomDomain: s remove domain
fc-deploy-test:
  region:   cn-hangzhou
  service:
    name: fc-deploy-service
  function:
    name:       event-py3
    runtime:    python3
    handler:    index.handler
    memorySize: 128
    timeout:    60
```

8. 结果验证。

   在函数计算控制台查看调用链，可以看到您刚创建的自定义Span与函数计算的系统Span连接起来。



5.调用分析

5.1. 调用分析简介

本文介绍什么是调用分析，包括调用分析的概念、使用场景及注意事项、函数指标及日志查询。

什么是调用分析

调用分析功能是函数请求级别的执行状态汇总。开启调用分析功能后，系统会收集函数每次执行的指标信息，包括性能指标、异常指标及追踪指标，并将这些指标信息投递到您在日志配置时选择的日志仓库中。

- 性能指标：内存使用情况、函数执行时间、初始化时间及冷启动信息等。
- 异常指标：是否执行失败、错误详情等。
- 追踪指标：链路追踪详情、是否采样等。

开启调用分析后可以洞察每一次的函数调用，帮助您掌握函数执行情况。

- 查看函数执行详情、性能指标及错误信息等。
- 与函数日志结合，创建自定义监控大盘。
  您可以在函数中记录一些业务相关日志，结合业务日志与分析日志来创建自定义监控大盘。更多信息，请参见基于请求级别指标创建Grafana大盘。
- 生效范围
  调用分析是服务的配置项，开启调用分析将对服务下的所有函数生效，每个函数的每次执行都会记录一条日志。
- 前提条件
  开启调用分析前，需要开启日志配置。调用分析的日志会被记录到您配置的日志库中，如果您没有配置日志，需要配置日志。
- 费用
  日志投递到日志服务中，会产生一定的日志存储费用。关于日志服务定价，请参见产品定价。

概念

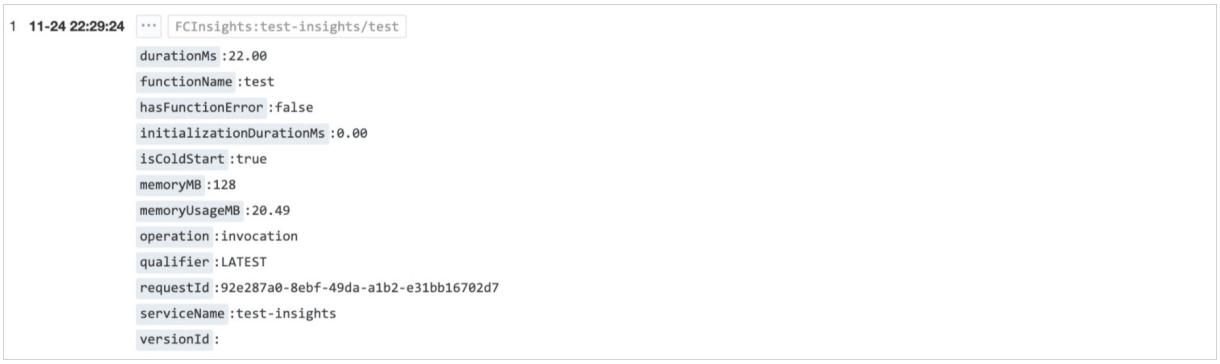
使用场景

注意事项

函数指标

开启调用分析功能后，系统会收集函数每次执行的指标信息。同时，这些指标信息将被投递到您在配置日志时选择的日志仓库中。如果您没有配置日志，则无法开启调用分析功能。

调用分析会将指标信息记录到您的Logstore中，具体形式如下：



调用分析会采集以下指标。

指标名称	描述	示例值	是否每次调用都会记录
serviceName	服务名称	my-service	是
functionName	函数名称	my-function	是
versionId	版本名称	12	是
qualifier	服务别名	prod	是，默认为LATEST。
requestId	请求ID	db72ce53-ccbe-4216-af55-642622e01494	是
operation	操作名称	invocation	是
memoryMB	函数的内存限制	512	是
memoryUsageMB	函数执行使用的内存	410	是
durationMs	函数执行时间	20.20	是
isColdStart	是否冷启动	false	是，默认值为false。
hasFunctionError	函数执行是否出错	false	是，默认值为false。
errorType	函数错误类型，包含以下三种： • FunctionOOMError：内存溢出。 • FunctionTimeoutError：执行时间超限。 • FunctionUnhandledError：未捕获的其他异常。	FunctionUnhandledError	否，仅在函数执行出现错误即 `hasFunctionError:true` 时记录。
initializationDurationMs	初始化函数执行时间	99.00	否，仅当发生冷启动且配置初始化函数时记录。

指标名称	描述	示例值	是否每次调用都会记录
traceContext	链路追踪上下文信息	371d3ff242fcee9:371d3ff242fcee9:0:1	否，仅当配置链路追踪时记录。
isSampled	请求是否被链路追踪采样	true	否，仅当配置链路追踪时记录。

日志查询

调用分析的日志主题遵循以下格式：`FCInsights:serviceName/functionName`。

您可以在日志服务中通过查询条件`__topic__: "FCInsights:test-insights/test"`筛选出所有调用分析产生的日志。更多信息，请参见查询和分析日志。

5.2. 配置调用分析

本文介绍如何在函数计算控制台配置调用分析功能及查看调用分析信息的操作步骤。

背景信息

调用分析简介。

前提条件

- 函数计算
  - 创建服务
- 日志服务SLS
  - 创建日志项目
  - 创建日志库

为服务配置调用分析

1. 登录函数计算控制台。
2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。
4. 在**服务列表**页面，找到目标服务。在其**操作**列，单击**配置**。
5. 在编辑服务页面的**日志配置**区域，配置以下参数，然后单击**保存**。

参数名称	参数说明	示例值
日志功能	是否支持将函数调用执行的日志存储至日志服务SLS。关于日志功能的详细信息，请参见配置日志。	启用
日志项目	选择已配置的日志项目。	aliyun-fc-hangzhou-2238f0df-a742-524f-9f90-976ba*******
日志库	选择已配置的日志库。	function-log
请求级别指标	是否查看该服务下所有函数的某次调用所消耗的时间及内存。	启用

配置完成后，您可以在某次函数调用结束后，在**调用日志**页签下的**调用请求列表**页签找到对应的调用请求行，然后单击请求ID查看本次调用消耗的时间、内存和调用方式等信息。

5.3. 基于请求级别指标创建Grafana大盘

函数计算提供的调用分析功能会收集函数每次执行的指标信息，并将这些指标投递到您在日志配置时选择的Logstore。您可以结合业务日志与分析日志来创建自定义监控大盘，即Grafana大盘。本文介绍如何在阿里云应用实时监控服务ARMS控制台，创建函数基于请求级别指标的Grafana大盘。

前提条件

- 配置日志
- 配置调用分析
- 免费开通ARMS

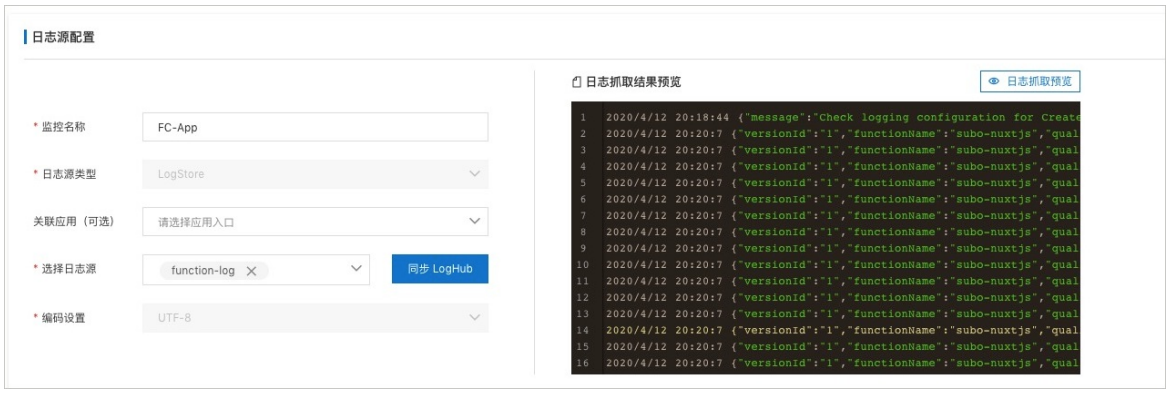
背景信息

您可以将函数执行的日志存储至阿里云日志服务SLS，ARMS日志监控与SLS打通，支持完全自定义的监控任务。对于使用函数计算的高度定制化的业务场景，您可通过创建ARMS日志监控任务来自由统计所需指标，生成需要的数据与报表，灵活地配置报警。最终，函数执行的指标信息将可视化地为您呈现。

调用分析所采集的指标详情，请参见调用分析简介。

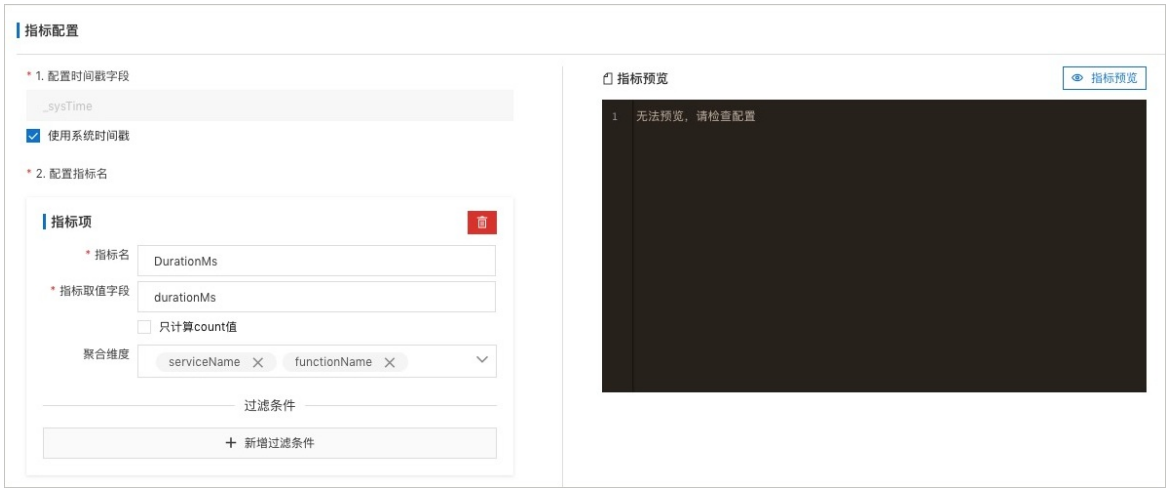
操作步骤

1. 登录ARMS控制台。
2. 在左侧导航栏选择**业务监控 > 日志接入**。
3. 在**日志接入**页面右上角单击**创建日志监控**。
4. 在**新建日志监控**页面的**日志源配置**区域设置如下参数。



参数	说明
监控名称	设置监控名称。
关联应用（可选）	选择需要关联的应用监控下的应用。设置关联应用后可以在**日志接入**页面按应用筛选过滤。
选择日志源	如果下拉框中没有可选的日志源，单击**同步LogHub**。 选择日志源后日志会自动抓取，您也可以在**日志抓取结果预览**区域单击右上角的**日志抓取预览**。

5. 在**指标配置**区域配置时间戳字段和指标名，配置完成后单击**保存监控配置**。
   在**配置指标名**区域单击**新增指标**，设置指标名、指标取值字段和聚合维度；单击**新增过滤条件**可以增加该指标的过滤条件。

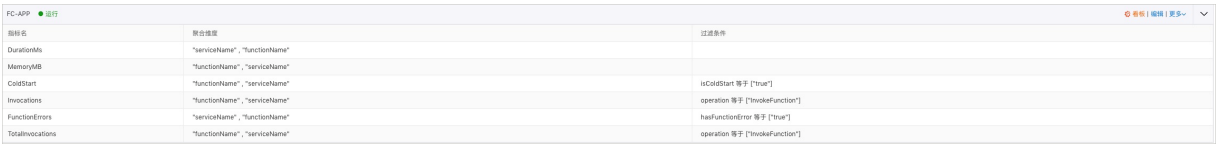


指标参数	说明
配置时间戳字段	选择日志中的时间戳字段。如果日志中没有时间戳字段，可以选中**使用系统时间戳**，即仅监控当前时间点的数据。
配置指标名	单击**新增指标**可以增加多个监控指标。
指标名	设置指标名称。
指标取值字段	是衡量目标的度量，选择日志中的数值类型字段名。ARMS的指标对应于实时计算后的Count、Max、Min、Sum、Avg等值。如果只需要统计日志行数，选中**只计算count值**。
聚合维度	是衡量目标的思维角度，选择日志里的非数值类型字段。例如想统计每个品牌的销售额，那么品牌名称就是聚合维度字段。最多支持选择8个聚合维度。
过滤条件	单击**新增过滤条件**可以增加多个过滤条件。 选择日志中的某一个字段作为过滤指标。例如：log_type等于engine。

您可以基于请求级别指标及函数业务指标进行自定义配置，获取您的自定义Grafana大盘。

查看Grafana大盘

在**日志接入**页面可以查看创建的所有日志监控。



- 单击各监控区域右上角的**看板**可以以大盘样式展示监控数据。

- 单击监控名称所在行可以展开或合并该监控的指标信息。
- 单击各监控区域右上角的**编辑**，在**日志监控设置**页面的**规则配置**页签可以修改监控信息，在**删除**页签可以删除该监控。

> **注意** 此操作将会清除该日志监控的所有数据，且删除之后无法恢复。

- 单击各监控区域右上角的**更多**可以启动或停止该监控。

更多信息

开始使用日志监控

6.ARMS高级监控

6.1. Java函数监控

函数计算无缝对接ARMS应用监控，您可以通过配置环境变量获得ARMS的APM应用监控功能，ARMS将对Java 8运行环境的应用进行无侵入零代码改动的高级监控，获得实例级别的可观测性，为您提供更丰富的指标，例如总请求量、响应时间及异常信息等。本文介绍如何将函数接入ARMS应用监控平台。

前提条件

- 成功创建函数
- 开通ARMS服务

背景信息

函数计算无缝对接ARMS应用监控平台后，您可以通过ARMS应用监控平台对目标函数进行监控追踪获取相关信息，例如实例级别的可观测性、链路追踪信息、Java虚拟机指标及代码级别的剖析（Profiling）信息等。

功能	描述
实例级别的可观测性	以函数实例作为维度，聚合丰富的主机监控指标，例如CPU、内存及请求等。
链路追踪	ARMS探针自动获取函数与上下游组件的拓扑关系及相关指标，您可以在ARMS监控平台查看相关信息，例如数据库、Redis及MQ等。
Java虚拟机指标	ARMS探针自动获取Java虚拟机应用，您可以在ARMS监控平台中查看Java虚拟机应用的相关监控数据，例如GC次数、堆信息及线程栈信息等。
代码级别的剖析	您可以查看函数执行过程中代码级别的剖析（Profiling）信息，例如每个方法的耗时、异常等。

操作步骤

1. 登录函数计算控制台。
2. 在左侧导航栏，单击**服务及函数**。
3. 在顶部菜单栏，选择地域。

   > **注意** 当您的函数成功接入ARMS监控平台后，如果您需要查看对应监控信息，要保证ARMS监控平台和函数所属同一地域。

4. 在**服务列表**页面，单击目标服务名称。
5. 在**函数管理**页面，找到目标函数。在其**操作**列，单击**配置**。
6. 在编辑函数页面的**环境变量**区域，按需选择配置环境变量的方式：
   - 使用表单编辑（默认方式）
     a. 单击**+添加变量**。
     b. 配置环境变量的键值对：
        - **键**：设置为FC_EXTENSIONS_ARMS_LICENSE_KEY。
        - **值**：设置为License Key信息。关于License Key信息的获取方式，请参见获取License Key信息。

- 使用JSON格式编辑
  a. 单击**使用JSON格式编辑**。
  b. 在输入框内，输入对应的JSON格式的键值对，格式如下：

```
{
    "FC_EXTENSIONS_ARMS_LICENSE_KEY":"iioe7jcnuk@a0bcdaec24f****"
}
```

  关于键值对的参数设置，如下所示：
  - **键**：设置为**FC_EXTENSIONS_ARMS_LICENSE_KEY**。
  - **值**：设置为License Key信息。关于License Key信息的获取方式，请参见获取License Key信息。

7. 单击**保存**。
   成功配置环境变量后，您的函数将被添加到ARMS应用监控进行高性能管理，同时ARMS监控将对您的服务进行计费。更多信息，请参见收费规则。

相关操作

如果您接入的函数的运行环境不是Java 8，您可以参考ARMS相关文档修改相关信息，更多信息，请参见应用监控接入概述。

执行结果

当您成功将函数接入ARMS应用监控平台后，您可以在ARMS控制台中查看接入的函数。
您可以登录ARMS控制台，选择**应用监控 > 应用列表**，然后单击目标应用名称，查看详细监控信息。更多信息，请参见应用总览。

> **说明** 目标应用的名称格式为FC:{serviceName}/{functionName}，例如FC:Service/Function。

7.监控报警FAQ

7.1. 我如何批量下载程序运行过程中产生的日志？

如果您的日志保存在日志服务的Logstore中，您可以通过日志服务的API查询和下载相关内容。

7.2. 函数调用正常，为什么我在监控页面看不到调用次数等指标？

请检查您使用的账户是否是子账户，是否具备云监控的只读权限。

8.函数计算支持被审计的事件说明

函数计算已与操作审计服务集成，您可以在操作审计中查询用户操作函数计算产生的管控事件。
操作审计记录了用户通过OpenAPI或控制台等方式操作云资源时产生的管控事件，函数计算支持在操作审计中查询的事件如下表所示。

事件名称	事件含义
CreateAlias	创建别名
CreateCustomDomain	创建自定义域名
CreateFunction	创建函数
CreateService	创建服务
CreateTrigger	创建触发器
DeleteAlias	删除别名
DeleteCustomDomain	删除自定义域名
DeleteFunction	删除函数
DeleteFunctionAsyncInvokeConfig	删除一个服务下某个函数的异步调用配置
DeleteLayerVersion	删除层版本
DeleteService	删除服务
DeleteTrigger	删除触发器
GetAlias	获取别名信息
GetCustomDomain	获取自定义域名
GetFunction	获取函数信息
GetFunctionAsyncInvokeConfig	查询一个服务下某个函数的异步调用配置
GetFunctionCode	获取函数代码包
GetLayerVersion	获取层版本信息
GetProvisionConfig	获取预留配置
GetResourceTags	获取资源所有的标签信息
GetService	获取服务信息
GetTrigger	获取触发器信息
ListAliases	获取别名列表

事件名称	事件含义
ListFunctionAsyncInvokeConfigs	查询一个服务下某个函数的所有异步配置
ListFunctions	获取函数列表
ListLayers	获取层列表
ListLayerVersions	获取层的版本列表
ListProvisionConfigs	获取预留配置列表
ListServices	获取服务列表
ListTriggers	获取触发器列表
OpenService	开通服务
PutFunctionAsyncInvokeConfig	创建或更新函数的异步调用配置
PutProvisionConfig	设置预留配置
TagResource	为资源创建标签
UnTagResource	删除资源的标签
UpdateAlias	更新别名
UpdateCustomDomain	更新自定义域名配置
UpdateFunction	更新函数信息
UpdateService	更新服务信息
UpdateTrigger	更新触发器