云监控

开发指南

MORE THAN JUST CLOUD | Alibaba Cloud

开发指南

接口简介

欢迎使用阿里云监控（Alibaba Cloud Monitor）。用户可以使用本文档介绍的API对云监控服务进行相关操作。

目前提供监控数据查询的接口。请确保在使用这些接口前，已充分了解CloudMonitor产品说明和使用协议。

注意事项

OpenAPI提供最近31天的监控数据。

调用方式

服务地址

CloudMonitor API的服务接入地址为 metrics.aliyuncs.com

通信协议

支持通过HTTP进行请求通信

请求方法

支持HTTP GET方法发送请求，这种方式下请求参数需要包含在请求的URL中。

请求参数

每个请求都需要包含公共的鉴权、签名相关请求参数和相关操作所特有的请求参数。

字符编码

请求及返回结果都使用UTF-8字符集进行编码。

公共参数

名称	类型	是否必须	描述
Format	String	否	返回值的类型，云监控仅支持JSON
Version	String	是	API版本号，为日期形式：YYYY-MM-DD，本版本对应为2015-10-20
AccessKeyId	String	是	阿里云颁发给用户的访问服务所用的密钥ID
Signature	String	是	签名结果串，关于签名的计算方法，请参见<签名机制>。
SignatureMethod	String	是	签名方式，目前支持HMAC-SHA1
Timestamp	String	是	请求的时间戳。日期格式按照ISO8601标准表示，并需要使用UTC时间。格式为：YYYY-MM-DDThh:mm:ssZ 例如，2014-01-10T12:00:00Z（为北京时间2014年1月10日20点0分0秒）
SignatureVersion	String	是	签名算法版本，目前版本是1.0
SignatureNonce	String	是	唯一随机数，用于防止网络重放攻击。用户在不同请求间要使用不同的随机数值
RegionId	String	是	保留关键字，目前云监控全部使用cn-hangzhou

签名机制

CloudMonitor服务会对每个访问的请求进行身份验证，使用HTTP需要在请求中包含签名（Signature）信息。CloudMonitor通过使用Access Key ID和Access Key Secret进行对称加密的方法来验证请求的发送者身份。

Access Key ID和Access Key Secret由阿里云官方颁发给访问者（可以通过阿里云官方网站申请和管理），其中Access Key ID用于标识访问者的身份；Access Key Secret是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密，只有阿里云和用户知道。

Java 签名算法可直接引用签名算法参考。

签名处理步骤

使用请求参数构造规范化的请求字符串（Canonicalized Query String）。

使用上一步构造的规范化字符串按照下面的规则构造用于计算签名的字符串。

```
StringToSign=
HTTPMethod + “&” +
percentEncode(“/”) + ”&” +
percentEncode(CanonicalizedQueryString)
```

其中HTTPMethod是提交请求用的HTTP方法，比GET.percentEncode(“/”)是按照1.b中描述的URL编码规则对字符“/”进行编码得到的值，即“%2F”。

percentEncode(CanonicalizedQueryString)是对第1步中构造的规范化请求字符串按1.b中描述的URL编码规则编码后得到的字符串

按照RFC2104的定义，使用上面的用于签名的字符串计算签名HMAC值。注意：计算签名时使用的Key就是用户持有的Access Key Secret并加上一个“&”字符(ASCII:38)，使用的哈希算法是SHA1。

4. 按照Base64编码规则把上面的HMAC值编码成字符串，即得到签名值（Signature）。
5. 将得到的签名值作为Signature参数添加到请求参数中，即完成对请求签名的过程。

构造请求字符串的方法

按照参数名称的字典顺序对请求中所有的请求参数（包括文档中描述的“公共请求参数”和给定了的请求接口的自定义参数，但不能包括“公共请求参数”中提到Signature参数本身）进行排序。

注：当使用GET方法提交请求时，这些参数就是请求URI中的参数部分（即URI中“?”之后由“&”连接的部分）。

对每个请求参数的名称和值进行编码。名称和值要使用UTF-8字符集进行URL编码。

a. 对于字符 A-Z、a-z、0-9以及字符“-”、“_”、“.”、“~”不编码。

b. 对于其他字符编码成“%XY”的格式，其中XY是字符对应ASCII码的16进制表示。比如英文的双引号（”）对应的编码就是%22。

c. 对于扩展的UTF-8字符，编码成“%XY%ZA…”的格式。

d. 需要说明的是英文空格（ ）要被编码是%20，而不是加号（+）。

注：一般支持URL编码的库（比如Java中的java.net.URLEncoder）都是按照“application/x-www-form-urlencoded”的MIME类型的规则进行编码的。实现时可以直接使用这类方式进行编码，把编码后的字符串中加号（+）替换成%20、星号（*）替换成%2A、%7E替换回波浪号（~），即可得到上述规则描述的编码字符串。

对编码后的参数名称和值使用英文等号（=）进行连接。

4. 再把英文等号连接得到的字符串按参数名称的字典顺序依次使用&符号连接，即得到规范化请求字符串。

以QueryMetric为例，签名前的请求URL为：

```
http://metrics.aliyuncs.com/?Action=QueryMetric&period=60&StartTime=2016-02-02T10:33:56Z
&Dimensions={instanceId:'i-23gp0zfjl'}&Timestamp=2016-02-04T03:17:29Z&Project=acs_ecs
&SignatureVersion=1.0&Format=JSON&SignatureNonce=53fddcfe-422a-4177-b983-e33981c9084c
&Version=2015-10-20&AccessKeyId=TestId&Metric=CPUUtilization&SignatureMethod=HMAC-
SHA1&RegionId=cn
```

那么StringToSign就是:

```
GET&%2F&AccessKeyId%3DTestId%26Action%3DQueryMetric%26Dimensions%3D%257BinstanceId%253A%2527i-
23gp0zfjl%2527%257D%26Format%3DJSON%26Metric%3DCPUUtilization%26Project%3Dacs_ecs%26RegionId%3D
cn%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3D530b9e7a-71e5-4744-8548-
77c5df29b8cb%26SignatureVersion%3D1.0%26StartTime%3D2016-02-
02T10%253A33%253A56Z%26Timestamp%3D2016-02-04T03%253A17%253A29Z%26Version%3D2015-10-
20%26period%3D60
```

假如使用的Access Key Id是“TestId”，Access Key Secret是“TestSecret”，用于计算HMAC的Key就是“TestSecret&”，则计算得到的签名值是：

```
IxsQ79fVwUu33iwZeH11Z2PfwqQ=
```

签名后的请求URL为（注意增加了Signature参数）：

```
http://metrics.aliyuncs.com/?Action=QueryMetric&period=60&StartTime=2016-02-
02T10%3A33%3A56Z&Dimensions=%7BinstanceId%3A%27i-23gp0zfjl%27%7D
```

```
&Timestamp=2016-02-
04T03%3A17%3A29Z&Project=acs_ecs&SignatureVersion=1.0&Format=JSON&SignatureNonce=530b9e7a-71e5-
4744-8548-77c5df29b8cb
&Version=2015-10-20&AccessKeyId=TestId&Metric=CPUUtilization&SignatureMethod=HMAC-
SHA1&RegionId=cn&Signature=IxsQ79fVwUu33iwZeH11Z2PfwqQ%3D
```

接口说明

概述

对CloudMonitor服务接口的调用是通过向CloudMonitor服务端发送HTTP请求（目前仅支持HTTP），并获取CloudMonitor服务对该请求响应结果的过程。

CloudMonitor服务端在接收到用户请求后，对请求做必要的身份验证和参数验证，在所有验证成功后根据请求的指定参数提交或完成相应操作，并把处理的结果以HTTP响应地形式返回给调用者。

请求组成

请求由以下几个部分组成：

- HTTP方法——目前CloudMonitor服务的所有接口都支持GET方法的调用，上报监控数据还支持POST方法。
- 请求URL——请求的服务地址、要执行的操作名称、操作参数和公共请求参数都包含在请求的URL中。
- 服务端地址：CloudMonitor服务的域名是http://metrics.aliyuncs.com/。
- 操作名称：每个接口都需要指定要执行的操作名称，即Action参数。
- 操作参数：根据要执行的操作不同，需要传入不同的操作参数，详见每个接口的说明。
- 公共请求参数：包含时间戳、签名信息等每个请求都要包含的参数。
- 为了使服务端能够正确地验证用户的身份并授权请求执行，请求在提交前要进行签名处理。签名的规则参见签名机制一节。
- 在服务端对请求处理完成后，会返回响应结果。响应结果分为成功结果和错误消息，格式描述参见返回结果一节。客户端可以解析响应的消息体，得到执行结果。

请求参数

名称	类型	是否必须	描述
Action	String	是	操作接口名,系统规定参数,取值: QueryMetricList
Project	String	是	名字空间,表明监控数据所属产品,如“acs_ecs”,” acs_rd

			s”等,可用命名空间
Metric	String	是	监控项名称,可用名称,参考Metric List
Period	String	否	时间间隔，统一用秒数来计算,例如 60, 300, 900。 如果不填写,则按照注册监控项时申明的上报 周期来查询原始数据;如果填写统计周期,则 查询对应的统计数据
StartTime	String	是	开始时间,可以传入距离 1970 年 1 月 1 日 0 点的毫秒数，也可以传入format数据，如2015-10-20 00:00:00
EndTime	String	否	可以传入距离 1970 年 1 月 1 日 0 点的毫秒数，也可以传入format数据，如2015-10-20 00:00:00
Dimensions	String	是	定位监控项数据位置的维度，例如磁盘IO这个监控项，通过实例和磁盘名称两个维度可以定位到唯一的监控点位置。知意：各监控项的dimensions详见“预设监控项参考”
Cursor	String	否	分页指针，代表下一次查询从上一次查询基础上继续查询。
Length	String	否	本次查询的条数，最大1000，若设置超越1000的数字，则自动重置为1000。

参数说明

Project、Metric 、Period、Dimensions如何赋值，请查看预设监控项参考。

开始和结束时间执行的是左开右闭的模式，startTime不能等于或者大约endTime。

Cursor是分页模式下的参数，只要存在就说明还有下一页，返回为null则说明没有下一页了。

Period一般包含60（一分钟）、300（五分钟）、900（十五分钟）。请根据文档以及查询场景的需要考虑period。比如查询一天范围使用period=60，则返回1000条数据（实际存在1440，因为最大返回值不超过1000，则只返回前1000条）；如果使用period=300，则返回288条数据。

参数第一个字母都是大写。

本接口支持RAM子账号调用，授权时操作描述符为：“cms:QueryMetricList”，资源描述符为：“*”。

返回参数

名称	类型	描述
Code	String	返回码，一般包括200（正常），400（参数错误），403（权限错误），500（服务器错误）等代码，非200都是错误码，请参阅错误码提示。
Msg	String	查询返回的状态描述信息。Code为”200”时Msg为空
Success	Boolean	当次查询是否成功执行，如果服务器端有异常此返回值为false，正常为true
Size	Intege	当次查询实际返回的条数
RequestId/TracerId	String	唯一请求标识，出现问题时，可以提供此字段给技术人员进行问题排查，用于定位问题。
Datapoints	JSON	监控数据
Cursor	String	根据用户查询的时间范围，如果超过length指定的条数，则返回cursor，作为下一次查询的参数。若cursor不存在，则说明已经全部查询完成

返回值错误码

错误码	描述	含义
400	Bad Request	参数错误
403	Forbidden	权限限制，ak跟查询的内容不符
500	Internal Server Error	服务器错误

调用示例

示例以java为例，签名方法详见：调用方式中的签名算法

签名代码：UrlUtil.java

```
import java.net.URLEncoder;
import java.util.Map;

import org.apache.commons.lang.StringUtils;
import org.apache.log4j.Logger;

public class UrlUtil {
private static Logger logger = Logger.getLogger(UrlUtil.class);

private final static String CHARSET_UTF8 = "utf8";

/**
*
* @param url
* @return
*/
public static String urlEncode(String url) {
if (!StringUtils.isEmpty(url)) {
try {
url = URLEncoder.encode(url, "UTF-8");
} catch (Exception e) {
logger.warn("Url encode error:" + e.getMessage());
}
}

return url;
}

public static String generateQueryString(Map<String, String> params, boolean isEncodeKV) {
StringBuilder canonicalizedQueryString = new StringBuilder();
for (Map.Entry<String, String> entry : params.entrySet()) {
if (isEncodeKV)
canonicalizedQueryString.append(percentEncode(entry.getKey())).append("=")
.append(percentEncode(entry.getValue())).append("&");
else
canonicalizedQueryString.append(entry.getKey()).append("=")
.append(entry.getValue()).append("&");
}
if (canonicalizedQueryString.length() > 1) {
canonicalizedQueryString.setLength(canonicalizedQueryString.length() - 1);
}
return canonicalizedQueryString.toString();
}

public static String percentEncode(String value) {
try {
// 使用URLEncoder.encode编码后，将"+","*","%7E"做替换即满足 API规定的编码规范
```

```
return value == null ? null : URLEncoder.encode(value, CHARSET_UTF8)
.replace("+", "%20").replace("*", "%2A").replace("%7E", "~");
} catch (Exception e) {
//不可能发生的异常
}
return "";
}
}
```

SignatureUtils.java

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.util.Map;
import java.util.TreeMap;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.lang.StringUtils;


public class SignatureUtils {

private final static String CHARSET_UTF8 = "utf8";
private final static String ALGORITHM = "UTF-8";
private final static String SEPARATOR = "&";

public static Map<String, String> splitQueryString(String url)
throws URISyntaxException, UnsupportedEncodingException {
URI uri = new URI(url);
String query = uri.getQuery();
final String[] pairs = query.split("&");
TreeMap<String, String> queryMap = new TreeMap<String, String>();
for (String pair : pairs) {
final int idx = pair.indexOf("=");
final String key = idx > 0 ? pair.substring(0, idx) : pair;
if (!queryMap.containsKey(key)) {
queryMap.put(key, URLDecoder.decode(pair.substring(idx + 1), CHARSET_UTF8));
}
}
return queryMap;
}

public static String generate(String method, Map<String, String> parameter,
String accessKeySecret) throws Exception {
String signString = generateSignString(method, parameter);
System.out.println("signString---"+signString);
byte[] signBytes = hmacSHA1Signature(accessKeySecret + "&", signString);
String signature = newStringByBase64(signBytes);
```

```
System.out.println("signature---"+signature);
if ("POST".equals(method))
return signature;
return URLEncoder.encode(signature, "UTF-8");

}

public static String generateSignString(String httpMethod, Map<String, String> parameter)
throws IOException {
TreeMap<String, String> sortParameter = new TreeMap<String, String>();
sortParameter.putAll(parameter);

String canonicalizedQueryString = UrlUtil.generateQueryString(sortParameter, true);
if (null == httpMethod) {
throw new RuntimeException("httpMethod can not be empty");
}

StringBuilder stringToSign = new StringBuilder();
stringToSign.append(httpMethod).append(SEPARATOR);
stringToSign.append(percentEncode("/")).append(SEPARATOR);
stringToSign.append(percentEncode(canonicalizedQueryString));

return stringToSign.toString();
}

public static String percentEncode(String value) {
try {
return value == null ? null : URLEncoder.encode(value, CHARSET_UTF8)
.replace("+", "%20").replace("*", "%2A").replace("%7E", "~");
} catch (Exception e) {
}
return "";
}

public static byte[] hmacSHA1Signature(String secret, String baseString)
throws Exception {
if (StringUtils.isEmpty(secret)) {
throw new IOException("secret can not be empty");
}
if (StringUtils.isEmpty(baseString)) {
return null;
}
Mac mac = Mac.getInstance("HmacSHA1");
SecretKeySpec keySpec = new SecretKeySpec(secret.getBytes(CHARSET_UTF8), ALGORITHM);
mac.init(keySpec);
return mac.doFinal(baseString.getBytes(CHARSET_UTF8));
}

public static String newStringByBase64(byte[] bytes)
throws UnsupportedEncodingException {
if (bytes == null || bytes.length == 0) {
return null;
}

return new String(Base64.encodeBase64(bytes, false), CHARSET_UTF8);
}
```

```
public static void main(String[] args) {
String str =
"GET&%2F&AccessKeyId%3DCdwKFNmXeHJuMOrT%26Action%3DDescribeInstances%26Format%3DJSON%26Regi
onId%3Dcn-hangzhou%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3D9fdf20f2-9a32-4872-bcd4-
c6036082ebef%26SignatureVersion%3D1.0%26Timestamp%3D2015-12-
21T09%253A05%253A44Z%26Version%3D2014-05-26";
byte[] signBytes;
try {
signBytes = SignatureUtils.hmacSHA1Signature("byczfpx4PKBzUNjjL4261cE3s6HQmH" + "&", str.toString());
String signature = SignatureUtils.newStringByBase64(signBytes);

} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}
}
```

ECS 实例监控信息查询示例

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URISyntaxException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
import java.util.SimpleTimeZone;
import java.util.UUID;
import net.sf.json.JSON;
import net.sf.json.JSONArray;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.methods.PostMethod;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
public class EcsTemplate {
private final static String SIGNATURE_VERSION = "1.0";
private final static String defaultSignatureType = "HMAC-SHA1";
private final static String API_Format = "JSON";
//cms version
private final static String API_VERSION = "2015-10-20";
//输入你的AK信息
private final static String accessKeyId = "YouraccessKeyId";
private final static String accessKeySecret = "YouraccessKeySecret";
private final static String domainnew = "metrics.aliyuncs.com";
protected String domain="metrics.aliyuncs.com";
//protected String domain = "alert.aliyuncs.com";
private static String formatISO8601Date(Date date) {
```

```
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'",
Locale.US);
df.setTimeZone(new SimpleTimeZone(0, "GMT"));
return df.format(date);
}
protected HttpHeaders buildHttpHeaders(String sessionId) {
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
headers.set("Authentication", sessionId);
return headers;
}
public static void main(String[] args) {
Map<String, String> parameters = new HashMap<String, String>();
//String action = "DescribeInstances";
String action = "QueryMetricList";
parameters.put("RegionId", "cn");
parameters.put("Action", action);
parameters.put("Project", "acs_ecs");
parameters.put("Metric", "CPUUtilization");
parameters.put("StartTime", "2016-01-01 00:00:00");
parameters.put("Period", "60");
parameters.put("Length", "1000");
//注意dimension格式！！
parameters.put("Dimensions", "{\"instanceId\":\"********\"}");
parameters.put("AccessKeyId", accessKeyId);
parameters.put("Format", API_Format);
parameters.put("SignatureMethod", defaultSignatureType);
parameters.put("SignatureNonce", UUID.randomUUID().toString());
parameters.put("SignatureVersion", SIGNATURE_VERSION);
parameters.put("Version",API_VERSION );
parameters.put("Timestamp", formatISO8601Date(new Date()));
String url = "http://"+domainnew;
if(!url.endsWith("/")){
url += "/";
}
url += "?";
url += UrlUtil.generateQueryString(parameters, true);
String signature = null;
try {
signature = SignatureUtils.generate("GET", parameters, accessKeySecret);
} catch (Exception e) {
e.printStackTrace();
}
url += "&Signature=" + signature;
HttpMethod http1 = new PostMethod();
HttpMethod httpMethod = new GetMethod(url);
System.out.println(url);
HttpClient httpClient = new HttpClient(new MultiThreadedHttpConnectionManager());
httpClient.getHttpConnectionManager().getParams().setConnectionTimeout(6000);//设置请求超时时间6秒
httpClient.getHttpConnectionManager().getParams().setSoTimeout(30000); //设置读取超时时间
try {
int statusCode = httpClient.executeMethod(httpMethod);
//if(statusCode!=200)return null;
byte[] bytes = httpMethod.getResponseBody();
String result = "[" + statusCode + "]"+new String(bytes, "UTF-8");
System.out.println(result);
```

```
} catch (HttpException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

签名算法参考

以下是Java版签名算法参考和C++版签名算法参考

Java签名算法参考

签名代码：UrlUtil.java

```
import java.net.URLEncoder;
import java.util.Map;

import org.apache.commons.lang.StringUtils;
import org.apache.log4j.Logger;

public class UrlUtil {
private static Logger logger = Logger.getLogger(UrlUtil.class);

private final static String CHARSET_UTF8 = "utf8";

/**
*
* @param url
* @return
*/
public static String urlEncode(String url) {
if (!StringUtils.isEmpty(url)) {
try {
url = URLEncoder.encode(url, "UTF-8");
} catch (Exception e) {
logger.warn("Url encode error:" + e.getMessage());
}
}

return url;
}

public static String generateQueryString(Map<String, String> params, boolean isEncodeKV) {
StringBuilder canonicalizedQueryString = new StringBuilder();
for (Map.Entry<String, String> entry : params.entrySet()) {
if (isEncodeKV)
```

```
canonicalizedQueryString.append(percentEncode(entry.getKey())).append("=")
.append(percentEncode(entry.getValue())).append("&");
else
canonicalizedQueryString.append(entry.getKey()).append("=")
.append(entry.getValue()).append("&");
}
if (canonicalizedQueryString.length() > 1) {
canonicalizedQueryString.setLength(canonicalizedQueryString.length() - 1);
}
return canonicalizedQueryString.toString();
}

public static String percentEncode(String value) {
try {
// 使用URLEncoder.encode编码后，将"+","*","%7E"做替换即满足 API规定的编码规范
return value == null ? null : URLEncoder.encode(value, CHARSET_UTF8)
.replace("+", "%20").replace("*", "%2A").replace("%7E", "~");
} catch (Exception e) {
//不可能发生的异常
}
return "";
}
}
```

SignatureUtils.java

```


import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.util.Map;
import java.util.TreeMap;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.lang.StringUtils;


public class SignatureUtils {

private final static String CHARSET_UTF8 = "utf8";
private final static String ALGORITHM = "UTF-8";
private final static String SEPARATOR = "&";

public static Map<String, String> splitQueryString(String url)
throws URISyntaxException, UnsupportedEncodingException {
URI uri = new URI(url);
String query = uri.getQuery();
final String[] pairs = query.split("&");
TreeMap<String, String> queryMap = new TreeMap<String, String>();
for (String pair : pairs) {
```

```
final int idx = pair.indexOf("=");
final String key = idx > 0 ? pair.substring(0, idx) : pair;
if (!queryMap.containsKey(key)) {
queryMap.put(key, URLDecoder.decode(pair.substring(idx + 1), CHARSET_UTF8));
}
}
return queryMap;
}

public static String generate(String method, Map<String, String> parameter,
String accessKeySecret) throws Exception {
String signString = generateSignString(method, parameter);
System.out.println("signString---"+signString);
byte[] signBytes = hmacSHA1Signature(accessKeySecret + "&", signString);
String signature = newStringByBase64(signBytes);
System.out.println("signature---"+signature);
if ("POST".equals(method))
return signature;
return URLEncoder.encode(signature, "UTF-8");

}

public static String generateSignString(String httpMethod, Map<String, String> parameter)
throws IOException {
TreeMap<String, String> sortParameter = new TreeMap<String, String>();
sortParameter.putAll(parameter);

String canonicalizedQueryString = UrlUtil.generateQueryString(sortParameter, true);
if (null == httpMethod) {
throw new RuntimeException("httpMethod can not be empty");
}

StringBuilder stringToSign = new StringBuilder();
stringToSign.append(httpMethod).append(SEPARATOR);
stringToSign.append(percentEncode("/")).append(SEPARATOR);
stringToSign.append(percentEncode(canonicalizedQueryString));

return stringToSign.toString();
}

public static String percentEncode(String value) {
try {
return value == null ? null : URLEncoder.encode(value, CHARSET_UTF8)
.replace("+", "%20").replace("*", "%2A").replace("%7E", "~");
} catch (Exception e) {
}
return "";
}

public static byte[] hmacSHA1Signature(String secret, String baseString)
throws Exception {
if (StringUtils.isEmpty(secret)) {
throw new IOException("secret can not be empty");
}
if (StringUtils.isEmpty(baseString)) {
return null;
```

```
}
Mac mac = Mac.getInstance("HmacSHA1");
SecretKeySpec keySpec = new SecretKeySpec(secret.getBytes(CHARSET_UTF8), ALGORITHM);
mac.init(keySpec);
return mac.doFinal(baseString.getBytes(CHARSET_UTF8));
}

public static String newStringByBase64(byte[] bytes)
throws UnsupportedEncodingException {
if (bytes == null || bytes.length == 0) {
return null;
}

return new String(Base64.encodeBase64(bytes, false), CHARSET_UTF8);
}

public static void main(String[] args) {
String str =
"GET&%2F&AccessKeyId%3DCdwKFNmXeHJuMOrT%26Action%3DDescribeInstances%26Format%3DJSON%26Regi
onId%3Dcn-hangzhou%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3D9fdf20f2-9a32-4872-bcd4-
c6036082ebef%26SignatureVersion%3D1.0%26Timestamp%3D2015-12-
21T09%253A05%253A44Z%26Version%3D2014-05-26";
byte[] signBytes;
try {
signBytes = SignatureUtils.hmacSHA1Signature("byczfpx4PKBzUNjjL4261cE3s6HQmH" + "&", str.toString());
String signature = SignatureUtils.newStringByBase64(signBytes);

} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}
}
```

C++ 签名算法参考

```
#include "ali_rpc_request.h"
#include "ali_string_utils.h"
#include "ali_encode_utils.h"
#include "ali_urlencode.h"
#include "ali_log.h"
#include <stdio.h>
#include <time.h>
static std::string get_utc_string() {
time_t now;
struct tm *timenow;
now = time(&now);
timenow = gmtime(&now);
std::string res= get_format_string("%d-%02d-%02dT%02d:%02d:%02dZ", timenow->tm_year + 1900,
timenow->tm_mon + 1,
timenow->tm_mday,
timenow->tm_hour,
```

```
timenow->tm_min,
timenow->tm_sec);
return res;
}

AliRpcRequest::AliRpcRequest(std::string version,
std::string appid,
std::string secret,
std::string url)
: AliHttpRequest(url),
version_(version),
appid_(appid),
secret_(secret),
url_(url) {

}

std::string AliRpcRequest::GetSignUrl() {
std::string encoded;
std::map<std::string, std::string> mapWithPublicArgs;

time_t now;
time(&now);
this->sign_nounce = get_format_string("%ld", now);
this->utc_time_ = get_utc_string();
if(this->query_.size()) {
std::vector<std::string> vec_queries;
strsplit(this->query_, vec_queries, "&");
for(int i = 0; i < vec_queries.size(); i++) {
std::string& item = vec_queries[i];
int pos = item.find("=");
mapWithPublicArgs[item.substr(0, pos)] = item.substr(pos + 1, item.size() - pos - 1);
}
}
mapWithPublicArgs["Format"] = "JSON";
mapWithPublicArgs["Version"] = this->version_;
mapWithPublicArgs["AccessKeyId"] = this->appid_;
mapWithPublicArgs["SignatureMethod"] = "HMAC-SHA1";
mapWithPublicArgs["TimeStamp"] = utc_time_;
mapWithPublicArgs["SignatureVersion"] = "1.0";
mapWithPublicArgs["SignatureNonce"] = sign_nounce;//

for(std::map<std::string, std::string>::iterator it = mapWithPublicArgs.begin();
it != mapWithPublicArgs.end(); it++) {
if(!encoded.empty()) {
encoded.append("&");
}
append_format_string(encoded, "%s=%s", urlencode(it->first).c_str(), urlencode(it->second).c_str());
}

encoded = this->method_ + "&" + urlencode("/") + "&" + urlencode(encoded);
ALI_LOG("sign str=%s\n", encoded.c_str());
return encoded;
}

int AliRpcRequest::CommitRequest() {
```

```
std::string sign_url = GetSignUrl();
std::string sign = encode_compute_hmac_sha1(this->secret_ + "&",
(char*)sign_url.c_str(),
sign_url.size());
AddRequestQuery("TimeStamp", utc_time_);
AddRequestQuery("Format", "JSON");
AddRequestQuery("Version", this->version_);
AddRequestQuery("AccessKeyId", this->appid_);
AddRequestQuery("SignatureMethod", "HMAC-SHA1");
AddRequestQuery("SignatureVersion", "1.0");
AddRequestQuery("SignatureNonce", this->sign_nounce);
AddRequestQuery("Signature", sign);
return AliHttpRequest::CommitRequest();
}
```

注意事项

1. “求和值”为统计周期内数据的累计和，以ECS的公网流量为例，原单位为KB/min,则统计周期为5min的求和计算，返回结果为5min的流量总和。
2. 新版接口支持ECS基础监控数据查询，旧版接口不支持。
3. OpenAPI支持查询最近31天的监控数据。
4. 参数中的Dimension为json串，例如：{"instanceId":"i-23gyb3kkd"}。

云服务器ECS监控项参考

基础监控项

- ECS基础监控数据，无需安装插件即可查询监控数据。
- Project为acs_ecs_dashboard，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值ecs实例的instanceId。

Metric	监控项描述	单位	Dimensions	Statistics
CPUUtilization	CPU百分比	Percent	instanceId	Average、Minimum、Maximum
InternetInRate	公网流入带宽	bits/s	instanceId	Average、Minimum、Maximum
IntranetInRate	私网流入带宽	bits/s	instanceId	Average、Minimum、Maximum
InternetOutRate	公网流出带宽	bits/s	instanceId	Average、Minimum、Maximum

IntranetOutRate	私网流出带宽	bits/s	instanceId	Average、Minimum、Maximum
InternetOutRate_Percent	公网流出带宽使用率	%	instanceId	Average
DiskReadBPS	系统磁盘总读BPS	Bps	instanceId	Average、Minimum、Maximum
DiskWriteBPS	系统磁盘总写BPS	Bps	instanceId	Average、Minimum、Maximum
DiskReadIOPS	系统磁盘读IOPS	Count/Second	instanceId	Average、Minimum、Maximum
DiskWriteIOPS	系统磁盘写IOPS	Count/Second	instanceId	Average、Minimum、Maximum
VPC_PublicIP_InternetInRate	专有网络公网流入带宽	bits/s	instanceId	Average、Minimum、Maximum
VPC_PublicIP_InternetOutRate	专有网络公网流出带宽	bits/s	instanceId	Average、Minimum、Maximum
VPC_PublicIP_InternetOutRate_Percent	专有网络公网流出带宽使用率	%	instanceId	Average

操作系统级别监控项

新版插件监控项

- period最小为15秒。

Metric	监控项描述	单位	Dimensions	统计方法
cpu_idle	Host.cpu.idle，当前空闲CPU百分比	%	instanceId	Average、Minimum、Maximum
cpu_system	Host.cpu.system，当前内核空间占用CPU百分比	%	instanceId	Average、Minimum、Maximum
cpu_user	Host.cpu.user，当前用户空间占用CPU百分比	%	instanceId	Average、Minimum、Maximum
cpu_wait	Host.cpu.iowait，当前等待IO操	%	instanceId	Average、Minimum、

	作的CPU百分比			Maximum
cpu_other	Host.cpu.other，其他占用CUP百分比，其他消耗，计算方式为（Nice + SoftIrq + Irq + Stolen）的消耗	%	instanceId	Average、Minimum、Maximum
cpu_total	Host.cpu.total，当前消耗的总CPU百分比	%	instanceId	Average、Minimum、Maximum
memory_totalspace	Host.mem.total，内存总量	bytes	instanceId	Average、Minimum、Maximum
memory_usedspace	Host.mem.used，已用内存量，用户程序使用的内存 + buffers + cached，buffers为缓冲区占用的内存空间，cached为系统缓存占用的内存空间	bytes	instanceId	Average、Minimum、Maximum
memory_actualusedspace	Host.mem.actualused，用户实际使用的内存，计算方法为（used - buffers - cached）	bytes	instanceId	Average、Minimum、Maximum
memory_freespace	Host.mem.free，剩余内存量	bytes	instanceId	Average、Minimum、Maximum
memory_freeutilization	Host.mem.freeutilization，剩余内存百分比	%	instanceId	Average、Minimum、Maximum
memory_usedutilization	Host.mem.usedutilization，内存使用率	%	instanceId	Average、Minimum、Maximum
load_1m	Host.load1，过去1分钟的系统平均负载，Windows操作系统没有此指标	无	instanceId	Average、Minimum、Maximum
load_5m	Host.load5，过去5分钟的系统平均负载，Windows操作系统没有此指标	无	instanceId	Average、Minimum、Maximum

load_15m	Host.load15，过去15分钟的系统平均负载，Windows操作系统没有此指标	无	instanceId	Average 、Minimum 、Maximum
diskusage_used	Host.diskusage.used，磁盘的已用存储空间	bytes	instanceId, device	Average 、Minimum 、Maximum
diskusage_utilization	Host.disk.utilization，磁盘使用率	%	instanceId, device	Average 、Minimum 、Maximum
diskusage_free	Host.diskusage.free，磁盘的剩余存储空间	bytes/s	instanceId, device	Average 、Minimum 、Maximum
diskusage_total	Host.diskussage.total，磁盘存储总量	bytes	instanceId, device	Average 、Minimum 、Maximum
disk_readbytes	Host.disk.readbytes，磁盘每秒读取的字节数	bytes/s	instanceId, device	Average 、Minimum 、Maximum
disk_writebytes	Host.disk.writebytes，磁盘每秒写入的字节数	bytes/s	instanceId, device	Average 、Minimum 、Maximum
disk_readiops	Host.disk.readiops，磁盘每秒的读请求数量	次/秒	instanceId, device	Average 、Minimum 、Maximum
disk_writeiops	Host.disk.writeiops，磁盘每秒的写请求数量	次/秒	instanceId, device	Average 、Minimum 、Maximum
fs_inodeutilization	Host.fs.inode，inode使用率	%	instanceId, device	Average 、Minimum 、Maximum
networkin_rate	Host.netin.rate，网卡每秒接收的比特数，即网卡的上行带宽	bits/s	instanceId、device	Average 、Minimum 、Maximum
networkout_rate	Host.netout.rate，网卡每秒发送的比特数，即网卡的下行带宽	bits/s	instanceId、device	Average 、Minimum 、Maximum
networkin_packages	Host.netin.packages，网卡每秒接收的数据包数	个/秒	instanceId、device	Average 、Minimum 、Maximum
networkout_packages	Host.netout.packages，网卡每秒发送的数据包数	个/秒	instanceId、device	Average 、Minimum 、Maximum
networkin_error	Host.netin.error	个/秒	instanceId、	Average 、

packages	package，设备驱动器检测到的接收错误包的数量		device	Minimum 、Maximum
networkout_errorpackages	Host.netout.errorpackages，设备驱动器检测到的发送错误包的数量	个/秒	instanceId、device	Average 、Minimum 、Maximum
net_tcpconnection	Host.tcpconnection，各种状态下的TCP连接数包括LISTEN、SYN_SENT、ESTABLISHED、SYN_RECV、FIN_WAIT1、CLOSE_WAIT、FIN_WAIT2、LAST_ACK、TIME_WAIT、CLOSING、CLOSED	个	instanceId，state	Average 、Minimum 、Maximum

旧版插件监控项

- period最小为60秒。

Metric	监控项描述	单位	Dimensions	统计方法
vm.DiskIOReadNew	磁盘IO读	Bps	instanceId,diskname	Average 、Minimum 、Maximum
vm.DiskIOWriteNew	磁盘IO写	Bps	instanceId,diskname	Average 、Minimum 、Maximum
vm.DiskUtilization	磁盘使用率	Percent	instanceId,mountpoint	Average 、Minimum 、Maximum
vm.LoadAverage	平均负载	无	instanceId,period,period有“1min”,”5min”,”15min”三种值	Average 、Minimum 、Maximum
vm.MemoryUtilization	内存使用率	Percent	instanceId	Average 、Minimum 、Maximum
vm.TcpCount	tcp连接数	count/min	instanceId，state,state包括LISTEN、SYN_SENT、	Average 、Minimum 、Maximum

			ESTABLISHED、SYN_RECV、FIN_WAIT1、CLOSE_WAIT、FIN_WAIT2、LAST_ACK、TIME_WAIT、CLOSING、CLOSED	
vm.ProcessCount	系统进程总数	count/min	instanceId	Average 、Minimum 、Maximum
vm.Process.number	进程数，添加进程监控后，会对指定关注的进程进行计数，显示正在运行的被关注进程总数	count/min	instanceId,processName	Average 、Minimum 、Maximum

注：安装新版agent后，查询vm.DiskIORead和vm.DiskIOWrite时，diskname的value需赋值为挂载点目录。

云数据库RDS监控项参考

- Project为acs_rds_dashboard。采样周期默认为300s，Period赋值为300或300的整数倍。
- 如果已在RDS控制台开通1分钟监控频率，则采样周期最小为60s。
- Dimensions中的instanceId赋值RDS实例的instanceId。

Metric	监控项描述	单位	Dimensions	Statistics
CpuUsage	CPU使用率	Percent	instanceId, type= CpuUsage	Average 、Minimum 、Maximum
DiskUsage	磁盘使用率	Percent	instanceId, type= DiskUsage	Average 、Minimum 、Maximum
IOPSUsage	IOPS使用率	Percent	instanceId, type= IOPSUsage	Average 、Minimum 、Maximum
ConnectionUsage	连接数使用率	Percent	instanceId, type= ConnectionUsage	Average 、Minimum 、Maximum
DataDelay	只读实例延迟	秒	instanceId，type=DataDelay	Average 、Minimum 、Maximum
MemoryUsage	内存使用率	Percent	instanceId, type= MemoryUsage	Average 、Minimum 、Maximum

MySQL_NetworkInNew	Mysql每秒网络入流量	bits/s	instanceId	Average、Minimum、Maximum
MySQL_NetworkOutNew	Mysql每秒网络出流量	bits/s	instanceId	Average、Minimum、Maximum
SQLServer_NetworkInNew	SQLServe每秒网络入流量	bits/s	instanceId	Average、Minimum、Maximum
SQLServer_NetworkOutNew	SQLServer每秒网络出流量	bits/s	instanceId	Average、Minimum、Maximum

负载均衡 Sever Load Balancer 监控项参考

- Project为acs_slb_dashboard，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值SLB实例的instanceId。
- Dimensions中的port赋值SLB实例的端口。
- Dimensions中的vip赋值SLB实例的服务地址。

Metric	描述	单位	Dimensions	Statistics
HeathyServerCount	后端健康ECS实例个数	Count	instanceId	Average、Minimum、Maximum
UnhealthyServerCount	后端异常ECS实例个数	Count	instanceId	Average、Minimum、Maximum
PacketTX	端口每秒流出数据包数	Count/Second	instanceId, port, vip	Average、Minimum、Maximum
PacketRX	端口每秒流入数据包数	Count/Second	instanceId, port, vip	Average、Minimum、Maximum
TrafficRXNew	端口每秒流入数据量	bits/s	instanceId, port, vip	Average、Minimum、Maximum
TrafficTXNew	端口每秒流出数据量	bits/s	instanceId, port, vip	Average、Minimum、Maximum
ActiveConnection	端口当前活跃连接数，既客户端正在访问SLB产生的连接	Count	instanceId, port, vip	Average、Minimum、Maximum
InactiveConnection	端口当前非活跃连接数，既访问SLB后未断开的	Count	instanceId, port, vip	Average、Minimum、Maximum

	空闲的连接			
NewConnection	端口当前新建连接数	Count	instanceId, port, vip	Average 、Minimum 、Maximum
MaxConnection	端口当前并发连接数	Count	instanceId, port, vip	Average 、Minimum 、Maximum
DropConnection	监听每秒丢失连接数	Count/Second	instanceId, port, vip	Average 、Minimum 、Maximum
DropPacketRX	监听每秒丢失入包数	Count/Second	instanceId, port, vip	Average 、Minimum 、Maximum
DropPacketTX	监听每秒丢失出包数	Count/Second	instanceId, port, vip	Average 、Minimum 、Maximum
DropTrafficRX	监听每秒丢失入bit数	bits/s	instanceId, port, vip	Average 、Minimum 、Maximum
DropTrafficTX	监听每秒丢失出bit数	bits/s	instanceId, port, vip	Average 、Minimum 、Maximum
InstanceActiveConnection	实例每秒活跃连接数	Count/Second	instanceId	Average 、Minimum 、Maximum
InstanceDropConnection	实例每秒丢失连接数	Count/Second	instanceId	Average 、Minimum 、Maximum
InstanceDropPacketRX	实例每秒丢失入包数	Count/Second	instanceId	Average 、Minimum 、Maximum
InstanceDropPacketTX	实例每秒丢失出包数	Count/Second	instanceId	Average 、Minimum 、Maximum
InstanceDropTrafficRX	实例每秒丢失入bit数	bits/s	instanceId	Average 、Minimum 、Maximum
InstanceDropTrafficTX	实例每秒丢失出bit数	bits/s	instanceId	Average 、Minimum 、Maximum
InstanceInactiveConnection	实例每秒非活跃连接数	Count/Second	instanceId	Average 、Minimum 、Maximum
InstanceMaxConnection	实例每秒最大并发连接数	Count/Second	instanceId	Average 、Minimum 、Maximum

InstanceNewConnection	实例每秒新建连接数	Count/Second	instanceId	Average、Minimum、Maximum
InstancePacketRX	实例每秒入包数	Count/Second	instanceId	Average、Minimum、Maximum
InstancePacketTX	实例每秒出包数	Count/Second	instanceId	Average、Minimum、Maximum
InstanceTrafficRX	实例每秒入bit数	bits/s	instanceId	Average、Minimum、Maximum
InstanceTrafficTX	实例每秒出bit数	bits/s	instanceId	Average、Minimum、Maximum

对象存储 OSS 监控项参考

参见OSS监控项参考

云数据库 Memcache 版 监控项参考

- Project为acs_ocs，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值memcache实例的instanceId。

Metric	描述	单位	Dimensions	Statistics
Evict	缓存每秒数据逐出量	Count/Second	instanceId	Average、Minimum、Maximum
HitRate	缓存命中率	Percent	instanceId	Average、Minimum、Maximum
IntranetIn	缓存输入带宽	Bytes/s	instanceId	Average、Minimum、Maximum
IntranetOut	缓存输出带宽	Bytes	instanceId	Average、Minimum、Maximum
ItemCount	缓存数据个数	Count	instanceId	Average、Minimum、Maximum
UsedMemCache	已用缓存	Bytes	instanceId	Average、Minimum、Maximum
UsedQps	已使用QPS	Count	instanceId	Average、

				Minimum 、Maximum

弹性公网 IP

- Project为acs_vpc_eip，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值EIP实例的instanceId。
- Dimensions中的ip 赋值EIP实例的IP地址。

Metric	描述	单位	Dimensions	Statistics
net_tx.rate	流出带宽	Byte/s	instanceId	
net_rx.rate	流入带宽	Byte/s	instanceId	
net.txPkgs	流出数据包数	Count	instanceId	Average 、Minimum 、Maximum
net.rxPkgs	流入数据包数	Count	instanceId	Average 、Minimum 、Maximum

云数据库 Redis 版

- Project为acs_kvstore，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值Redis实例的instanceId。

Metric	描述	单位	Dimensions	Statistics
MemoryUsage	已用容量百分比	percent	instanceId	Average 、Minimum 、Maximum
ConnectionUsage	已用连接数百分比	percent	instanceId	Average 、Minimum 、Maximum
IntranetInRatio	写入网络带宽使用率	percent	instanceId	Average 、Minimum 、Maximum
IntranetOutRatio	读取网络带宽使用率	percent	instanceId	Average 、Minimum 、Maximum
IntranetIn	写入网络速率	bits/s	instanceId	Average 、Minimum 、Maximum
IntranetOut	读取网络速率	bits/s	instanceId	Average 、Minimum 、Maximum

FailedCount	操作KVSTORE失败次数	Count/Second	instanceId	Average、Minimum、Maximum
CpuUsage	CPU使用率	percent	instanceId	Average、Minimum、Maximum
UsedMemory	已用内存容量	Bytes	instanceId	Average、Minimum、Maximum

消息和通知服务MNS

- Project为acs_mns_new，采样周期为300s，Period赋值为300或300的整数倍
- Dimensions中的region 赋值队列所在的region
- Dimensions中的queue赋值队列名称

Metric	描述	单位	Dimensions	Statistics
ActiveMessages	活跃消息数	Count	region,queue	Average、Minimum、Maximum
InactiveMessages	非活跃消息数	Count	region,queue	Average、Minimum、Maximum
DelayMessages	延时消息数	Count	region,queue	Average、Minimum、Maximum

内容分发网络CDN

- Project为acs_cdn，采样周期为300s，Period赋值为300或300的整数倍。
- Dimensions中的instanceId 赋值被加速的域名。

Metric	描述	单位	Dimensions	Statistics
QPS	每秒请求数QPS，时间粒度内的总访问次数/时间粒度	Count	instanceId	Average、Minimum、Maximum
BPS	网络带宽峰值BPS，单位时间内网络流量的最大值	bits/s	instanceId	Average、Minimum、Maximum
hitRate	字节命中率，时间粒度内请求的字节数命中缓存	Percent	instanceId	Average、Minimum、Maximum

	的概率			
code4xx	返回码4xx占比，时间粒度内http返回码4XX占全部返回码的百分比	Percent	instanceId	Average 、Minimum 、Maximum
code5xx	返回码5xx占比，时间粒度内http返回码5XX占全部返回码的百分比	Percent	instanceId	Average 、Minimum 、Maximum
InternetOut	下行流量	Bytes	Average 、Minimum 、Maximum	

云数据库 MongoDB版

- Project为acs_mongodb，采样周期为300s，Period赋值为300或300的整数倍。
- Dimensions中的role赋值”Primary”或”Secondary”,表示主、备实例。

Metric	监控项名称	监控项描述	单位	Dimensions	Statistics
CPUUtilization	CPU使用率	实例的CPU使用率	%	userId,instanceId、role	Average 、Minimum 、Maximum
MemoryUtilization	内存使用率	实例的内存使用率	%	userId,instanceId、role	Average 、Minimum 、Maximum
DiskUtilization	磁盘使用率	实例的磁盘使用率	%	userId,instanceId、role	Average 、Minimum 、Maximum
IOPSUtilization	IOPS使用率	实例的IOPS使用率	%	userId,instanceId、role	Average 、Minimum 、Maximum
Connection Utilization	连接数使用率	已经使用的连接数百分率	%	userId,instanceId、role	Average 、Minimum 、Maximum
QPS	平均每秒SQL查询数	Mongodb实例的平均每秒SQL查询数	个	userId,instanceId、role	Average 、Minimum 、Maximum
Connection Amount	连接数使用量	当前应用程序连接到Mongodb实例的数量	个	userId,instanceId、role	Average 、Minimum 、Maximum
InstanceDiskAmount	实例占用磁盘空间量	实例实际使用的磁盘空间总量	Bytes	userId,instanceId、role	Average 、Minimum 、Maximum

DataDiskAmount	数据占用磁盘空间量	数据占用的磁盘空间容量	Bytes	userId,instanceId、role	Average 、Minimum 、Maximum
LogDiskAmount	日志占用磁盘空间量	日志占用的磁盘空间容量	Bytes	userId,instanceId、role	Average 、Minimum 、Maximum
IntranetIn	网络入流量	实例的网络流入流量	Bytes	userId,instanceId、role	Average 、Minimum 、Maximum
IntranetOut	网络出流量	实例的网络流出流量	Bytes	userId,instanceId、role	Average 、Minimum 、Maximum
NumberRequests	请求数	发送到服务端的请求总量	个	userId,instanceId、role	Average 、Minimum 、Maximum
OpInsert	Insert操作次数	从mongodb实例最近一次启动到现在累计接收到的insert命令的次数	次	userId,instanceId、role	Average 、Minimum 、Maximum
OpQuery	Query操作次数	从mongodb实例最近一次启动到现在累计接收到的query命令的次数	次	userId,instanceId、role	Average 、Minimum 、Maximum
OpUpdate	Update操作次数	从mongodb实例最近一次启动到现在累计接收到的update命令的次数	次	userId,instanceId、role	Average 、Minimum 、Maximum
OpDelete	Delete操作次数	从mongodb实例最近一次启动到现在累计执行delete的操作次数	次	userId,instanceId、role	Average 、Minimum 、Maximum
OpGetmore	Getmore操作次数	从mongodb实例最近一次启动到现在累计执行getmore的操作次数	次	userId,instanceId、role	Average 、Minimum 、Maximum
OpComman	Command	从	次	userId,insta	Average 、

d	操作次数	mongodb实例最近一次启动到现在向数据库发出的command的累计次数		nceId、role	Minimum 、Maximum

高速通道

- Project为acs_express_connect ，采样周期为60s，Period赋值为60或60的整数倍。
- Dimensions中的instanceId赋值高速通道路由器接口ID。

Metric	监控项名称	单位	Dimensions
IntranetRX	网络流入流量	Bytes	instanceId
IntranetTX	网络流出流量	Bytes	instanceId
ReceiveBandwidth	网络流入带宽	bit/s	instanceId
TransportedBandwidth	网络流出带宽	bit/s	instanceId