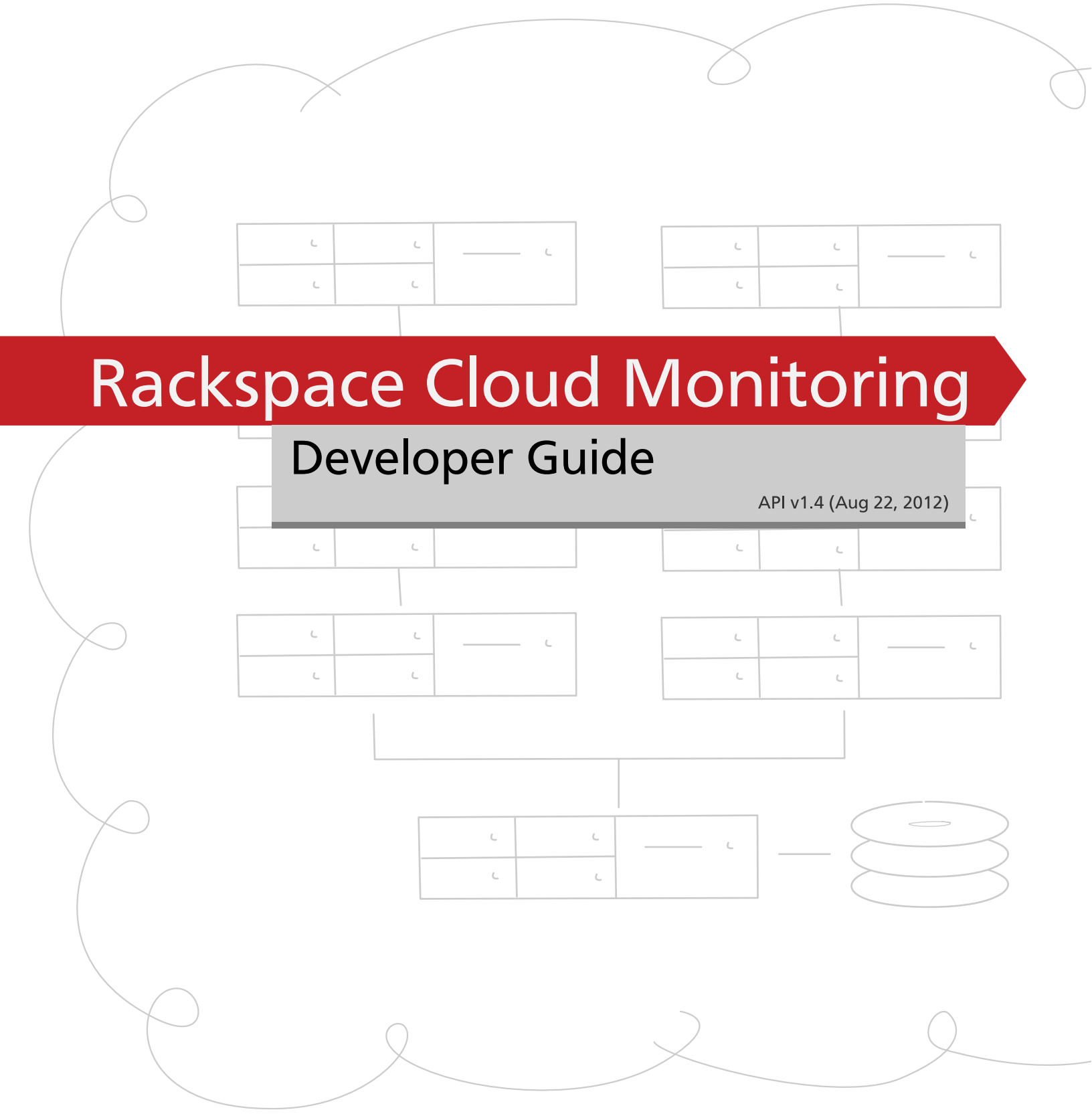


Rackspace Cloud Monitoring

Developer Guide

API v1.4 (Aug 22, 2012)

rackspace HOSTING

docs.rackspace.com/api

Rackspace Cloud Monitoring Developer Guide

API v1.4 (2012-08-22)

This document is intended for software developers interested in developing applications using the Rackspace Cloud Monitoring Application Programming Interface (API). The document is for informational purposes only and is provided "AS IS."

Table of Contents

List of Figures

List of Tables

List of Examples

1. Overview

Rackspace Cloud Monitoring analyzes cloud services and dedicated infrastructure using a simple, yet powerful API. The API currently includes monitoring for external services. The key benefits you receive from using this API include the following:

Use of Domain Specific Language (DSL)

The Rackspace Cloud Monitoring API uses a DSL, which makes it a powerful tool for configuring advanced monitoring features. For example, typically complex tasks, such as defining triggers on thresholds for metrics or performing an inverse string match become much easier with a concise, special purpose language created for defining alarms. For more information, see Alarms.

Monitoring from Multiple Datacenters

Rackspace Cloud Monitoring allows you to simultaneously monitor the performance of different resources from multiple datacenters and provides a clear picture of overall system health. It includes tunable parameters to interpret mixed results which help you to create deliberate and accurate alerting policies. See Alert Policies for more information.

Alarms and Notifications

When an alarm occurs on a monitored resource, Rackspace Cloud Monitoring sends you a notification so that you can take the appropriate action to either prevent an adverse situation from occurring or rectify a situation that has already occurred. These notifications are sent based on the severity of the alert as defined in the notification plan.

For more information about the basic foundations of this API, refer to Chapter 2, *Concepts* [4].

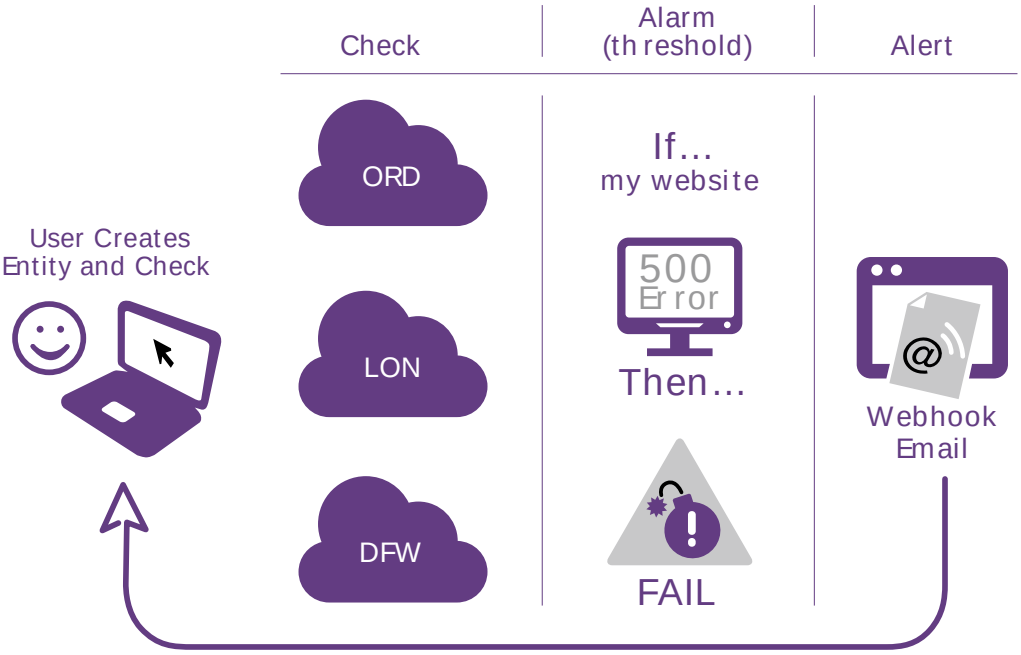
1.1. How Rackspace Cloud Monitoring Works

Rackspace Cloud Monitoring helps you to stay one step ahead of your customers by keeping a keen eye on all of your resources; from web sites to web servers, routers, load balancers, and more.

All you need to do is create an entity that represents the thing you want to monitor and then attach a check to it using one our predefined checks. For example, you might want to use the PING check to monitor your web site's public IP address.

You can run your checks from multiple monitoring zones. Each check has an alarm associated with it that serves as a threshold and processes the output of the check. When a specific condition is met, the alarm is triggered and your notification plan is put into action; sending you an email notification or a webhook to a URL.

Figure 1.1. Rackspace Cloud Monitoring Work Flow



1.2. Intended Audience

This document is intended for software developers interested in developing applications that use the Rackspace Cloud Monitoring product. It describes each API call, its associated options, and provides examples of successful and failed responses.

To use the information provided here, you should first have a general understanding of the Rackspace Monitoring Server service and have access to an active Rackspace account. You should also be familiar with:

- RESTful web services
- HTTP/1.1
- JSON and/or XML serialization formats

1.3. Document Change History

This version of the Developer Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Summary of Changes
Aug 22, 2012	Unlimited Availability
Aug 8, 2012	Fixed broken link.
Jun 17, 2012	Miscellaneous updates.
Mar 28, 2012	EAP

1.4. Additional Resources

You can download the most current version of this document from the Rackspace Cloud website at http://docs.rackspace.com/cm/api/v1.0/cm-devguide/content/index.html.

Try out the tutorial in the Rackspace Cloud Monitoring Getting Started Guide at: http://docs.rackspace.com/cm/api/v1.0/cm-getting-started/content/Introduction.html.

For information about Rackspace Cloud products, refer to http://www.rackspace.com/cloud. This site also offers links to official Rackspace support channels, including knowledge base articles, forums, phone, chat, and email.

You can also follow updates and announcements via twitter at http://www.twitter.com/rackspace.

This API uses standard HTTP 1.1 response codes as documented at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

2. Concepts

2.1. Monitoring Key Terms and Concepts

Entity

In Rackspace Cloud Monitoring, an entity is the object or *resource* that you want to monitor. It can be any object or device that you want to monitor. It's commonly a web server, but it might also be a website, a web page or a web service.

When you create an entity, you'll specify characteristics that describe *what* you are monitoring. At a minimum you must specify a name for the entity. The name is a user-friendly label or description that helps you identify the resource. You can also specify other attributes of the entity, such the entity's IP address, and any meta data that you'd like to associate with the entity.

Check

Once you've created an entity, you can configure one or more *checks* for it. A check is the foundational building block of the monitoring system, and is always associated with an entity. The check specifies the parts or pieces of the entity that you want to monitor, the monitoring frequency, how many monitoring zones are launching the check, and so on. Basically it contains the specific details of how you are monitoring the entity.

You can associate one or more checks with an entity. An entity must have at least one check, but by creating multiple checks for an entity, you can monitor several different aspects of a single resource.

For each check you create within the monitoring system, you'll designate a *check type*. The check type tells the monitoring system which method to use, PING, HTTP, SMTP, and so on, when investigating the monitored resource. Rackspace Cloud Monitoring check types are fully described here.

Note that if something happens to your resource, the check does not trigger a notification action. Instead, notifications are triggered by alarms that you create separately and associate with the check.

Monitoring Zones

When you create a check, you specify which monitoring zone(s) you want to launch the check from. A monitoring zone is the point of origin or "launch point" of the check. This concept of a monitoring zone is similar to that of a datacenter, however in the monitoring system, you can think of it more as a geographical region.

You can launch checks for a particular entity from multiple monitoring zones. This allows you to observe the performance of an entity from different regions of the world. It is also a way to prevent false alarms. For example, if the check from one

monitoring zone reports that an entity is down, a second or third monitoring zone might report that the entity is up and running. This gives you a better picture of an entity's overall health.

Collectors

A collector collects data from the monitoring zone and is mapped directly to an individual machine or a virtual machine. Monitoring zones contain many collectors, all of which will be within the IP address range listed in the response. Note that there may also be unallocated IP addresses or unrelated machines within that IP address range.

Alarms

An alarm contains a set of rules that determine when the monitoring system sends a notification. You can create multiple alarms for the different checks types associated with an entity. For example, if your entity is a web server that hosts your company's website, you can create one alarm to monitor the server itself, and another alarm to monitor the website.

The alarms language provides you with scoping parameters that let you pinpoint the value that will trigger the alarm. The scoping parameters are inherently flexible, so that you can set up multiple checks to trigger a single alarm. The alarm language supplies an adaptable triggering system that makes it easy for you to define different formulas for each alarm that monitors an entity's uptime. To learn how to use the alarm language to create robust monitors, see Alert Triggering and Alarms.

Notifications

A notification is an informational message that you receive from the monitoring system when an alarm is triggered. You can set up notifications to alert a single individual or an entire team. Rackspace Cloud Monitoring currently supports webhooks and email for sending notifications.

Notification Plans

A notification plan contains a set of notification rules to execute when an alarm is triggered. A notification plan can contain multiple notifications for each of the following states:

- Critical
- Warning
- Ok

Id

All objects in the monitoring system are identified by a uniquely generated Id, consisting of a two-character type prefix followed by a string of alphanumeric characters. You'll use an object's id when you want to perform certain operations on it. For example, when you want to create a check and associate it to an entity, you'll need to know the entity's id.

3. General API Information

The Rackspace Cloud Monitoring API provides a RESTful web service interface. All requests to authenticate and operate against the Rackspace Cloud Monitoring API are performed using SSL over HTTPS on TCP port 443.

3.1. Authentication

Every REST request against the Rackspace Cloud Monitoring API requires the inclusion of a specific authorization token, supplied by the `X-Auth-Token` HTTP header. You obtain this token by first using the Rackspace Cloud Authentication Service and supplying a valid username and API access key.

The Rackspace Cloud Authentication Service serves as the entry point to all Rackspace Cloud APIs and is itself a RESTful web service.

To access the Authentication Service, authenticate through:

https://auth.api.rackspacecloud.com/v1.1

3.1.1. Retrieving the Authentication Token

Verb	URI	Description
POST	/auth	Authenticate to receive a token and a service catalog.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), userDisabled (403), badRequest (400), authFault (500), serviceUnavailable (503)

The authenticate operation provides clients with an authentication token and a list of regional cloud endpoints.

Example 3.1. Auth Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<credentials xmlns="http://docs.rackspacecloud.com/auth/api/v1.1"
             username=❶"hub_cap"
             key=❷"a86850deb2742ec3cb41518e26aa2d89"/>
```

Example 3.2. Auth Request: JSON

```
{
    "credentials":{
        "username":❶"hub_cap",
        "key":❷"a86850deb2742ec3cb41518e26aa2d89"
    }
}
```

❶ The username supplied here is your common Rackspace Cloud username.
❷ The key is your API access key. The key can be obtained from the Rackspace Cloud Control Panel in the **Your Account** / **API Access** section.

Example 3.3. Auth Response: XML

```
<?xml version="1.0"?>
<container>
    <values>
        <entity created_at="1330044188322" id="enGTOR2C6u" updated_at=
"1330071481524">
            <labe❶l>Monitor Test</label>
            <ip_addresses>
                <default>192.168.0.1</default>
            </ip_addresses>
            <meta❷data/>
            <managed❸>false</managed>
            <uri/> ❹
            <agent_id/❺>
        </entity>
        <entity created_at="1327976441046" id="enHBMKJpe6" updated_at=
"1329412867854">
            <label>My Rackspace Server</label>
            <ip_addresses>
                <default>192.168.0.1</default>
            </ip_addresses>
            <metadata/>
            <managed>false</managed>
            <uri/>
            <agent_id/>
        </entity>
    </values>
    <metadata>
        <count>2</count>
        <limit>100</limit>
        <marker/❻>
        <next_href/>
    </metadata>
</container>
```

Example 3.4. Auth Response: JSON

```
{
    "auth":{
        "token":{
            "id":"asdasdasd-adsasdads-asdasdasd-adsadsasd",
            "expires":❶"2010-11-01T03:32:15-05:00"
        },
        "serviceCatalog":{
            "cloudFiles":[
                {
                    "region":❷"DFW",
                    "v1Default":❸true,
```

```
                    "publicURL":❹"https://storage.clouddrive.com/v1/
RackCloudFS_demo",
                    "internalURL":❺"https://storage-snet.clouddrive.com/v1/
RackCloudFS_demo"
                },
                {
                    "region":"ORD",
                    "publicURL":"https://otherstorage.clouddrive.com/v1/
RackCloudFS_demo",
                    "internalURL":"https://otherstorage-snet.clouddrive.com/
v1/RackCloudFS_demo"
                }
            ],
            "cloudFilesCDN":[
                {
                    "region":"DFW",
                    "v1Default":true,
                    "publicURL":"https://cdn.clouddrive.com/v1/
RackCloudFS_demo"
                },
                {
                    "region":"ORD",
                    "publicURL":"https://othercdn.clouddrive.com/v1/
RackCloudFS_demo"
                }
            ],
            "cloudServers":[ ❻
                {
                    "v1Default":true,
                    "publicURL":"https://servers.api.rackspacecloud.com/v1.0/
322781"
                }
            ]
        }
    }
}
```

❶ Tokens are valid for a finite duration. The `expires` attribute denotes the time after which the token will automatically become invalid. A token may be manually revoked before the time identified by the `expires` attribute; `expires` predicts a token's maximum possible lifespan but does not guarantee that it will reach that lifespan. Clients are encouraged to cache a token until it expires.

❷ A service may expose endpoints in different regions. Regional endpoints allow clients to provision resources in a manner that provides high availability.

❸ The `v1Default` attribute denotes that an endpoint is being returned in version 1.0 of the Cloud Authentication Service. The default value of `v1Default` is `false`; clients should assume the value is `false` when the attribute is missing. Auth 1.0 does not offer support for regional endpoints and therefore only returns one endpoint per service. Resources stored in endpoints where `v1Default` is `false` will not be seen by Auth 1.0 clients.

❹❺ An endpoint can be assigned public and internal URLs. A public URL is accessible from anywhere. Access to a public URL usually incurs traffic charges. Internal URLs are only accessible to services within the same region. Access to an internal URL is free of charge.

❻ Some services are not region-specific. These services supply a single non-regional endpoint and do not provide access to internal URLs.

3.1.2. Token Expiration

While an authentication token lasts, you can continue to perform requests, but once a token expires it returns an HTTP error code 401 Unauthorized. Given that an X-Auth-Token is good for 24 hours, long running or high request rate jobs should not try to authenticate at api.auth.rackspace.com on every request. You don't need to request another X-Auth-Token again until your existing X-Auth-Token expires. At that point you must obtain another authorization token. As a best practice example, here is some pseudo-code for re-authenticating. The best scalable process flow would be:

1. Begin requests by going to auth.api.rackspace.com for an X-Auth-Token.
2. Send request X-Storage-URL using the X-Auth-Token obtained in Step 1
3. Repeat step 2 using the same X-Auth-Token retrieved in Step 1 until either the job finishes or you get a result code of 401 (Unauthorized).
   - If the job finishes, you can allow the token to expire with no further action.
   - If result code is 401 then send a request to auth.api.rackspacecloud.com for a new X-Auth-Token.

A Python-based example of how to check for errors and re-authenticate upon receiving an error can be found in the OpenStack Swift project in client.py, which is freely available.

3.2. Endpoint Access

For Rackspace Cloud Monitoring, use the endpoint below to access the API:

https://monitoring.api.rackspacecloud.com/v1.0/*1234*

Replace the sample account ID number, *1234*, with your actual account number. Your account number is returned as part of the authentication service response, after the final '/' in the X-Server-Management-Url header. See Authentication for more information.

3.3. Request/Response Types

The Rackspace Cloud Monitoring API supports both the JSON and XML data serialization formats. The request format is specified using the `Content-Type` header and is required for operations that have a request body. The response format can be specified in requests using the `Accept` header. Note that it is possible for a response to be serialized using a format different from the request (see example below). If no response format is specified, JSON is the default.

Table 3.1. Response Types

Format	Accept Header	Default
JSON	application/json	Yes
JSONP*	application/json	No
XML	application/xml	No

*JSONP is the same as JSON except that the x-auth-token header is replaced with an HTTP query parameter with the same name. An additional HTTP query parameter JSONP indicates the method name to wrap the JSON response in.

In the request example below, notice that *Content-Type* is set to *application/json*, but *application/xml* is requested via the *Accept* header:

Example 3.5. JSON Request with Headers

```
POST https://monitoring.api.rackspacecloud.com/v1.0/<your_account_number>/
entities/enHBMKJpe6/test-check/
X-Auth-Token: <your_auth_token>
Content-Type: application/json
Accept: application/xml
'{ "details" : {  },
  "label" : "Website check 1",
  "monitoring_zones_poll" : [ "mzdfw" ],
  "period" : "60",
  "target_alias" : "default",
  "timeout" : 30,
  "type" : "remote.ping"
}'
```

Therefore an XML response format is returned:

Example 3.6. XML Response with Headers

```
<checks_data>
    <check_data>
        <timestamp>1332542815237</timestamp>
        <monitoring_zone_id>mzdfw</monitoring_zone_id>
        <available>false</available>
        <status>cnt=5,avail=0,min=-nan,max=-nan,avg=-nan</status>
        <metrics>
            <average>
                <type>n</type>
                <data/>
            </average>
            <minimum>
                <type>n</type>
                <data/>
            </minimum>
            <available>
                <type>n</type>
                <data>0.000000000000e+00</data>
            </available>
            <maximum>
                <type>n</type>
                <data/>
            </maximum>
            <count>
                <type>i</type>
                <data>5</data>
            </count>
        </metrics>
    </check_data>
```

```
</checks_data>
```

3.4. Versions

The Rackspace Cloud Monitoring API uses a URI versioning scheme. In the URI scheme, the first element of the path contains the target version identifier (e.g. https://monitoring.api.rackspacecloud.com/v1.0/...)

Example 3.7. Request with URI Versioning

```
GET /v1.0/entities HTTP/1.1
Host: monitoring.api.rackspacecloud.com/v1.0/12345/entities
Accept: application/xml
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

New features and functionality that do not break API-compatibility will be introduced in the current version of the API as extensions (see below) and the URI will remain unchanged. Features or functionality changes that would necessitate a break in API-compatibility will require a new version, which will result in URI version being updated accordingly. When new API versions are released, older versions will be marked as DEPRECATED. Providers should work with developers and partners to ensure there is adequate time to migrate to the new version before deprecated versions are discontinued.

3.5. Paginated Collections

To reduce load on the service, list operations will return a maximum number of items at a time. To navigate the collection, the parameters limit and marker can be set in the URI (e.g. *?limit=200&marker=enCCCCCC*). The marker parameter is the ID of a first item in the next page. This item can be found in the metadata object under the *next_key* tag. Items are sorted by the ID name in a lexicographic order. The limit parameter sets the page size. It defaults to *100* items per page and the maximum value is *1000*. Both parameters are optional. If the client requests a limit beyond that which is not supported an invalidLimit (400) fault may be thrown.

For convenience, collections contain a link to the next page (the *next_href* attribute in the metadata object). If there are no more objects, the *next_key* and *next_href* attributes will be empty. The following examples illustrate two pages in a collection of entities. The first page was retrieved via a **GET** to https://monitoring.api.rackspacecloud.com/v1.0/entities?limit=1. In these examples, the limit parameter sets the page size to a single item. Subsequent *next_href* link will honor the initial page size. Thus, a client may follow this link to traverse a paginated collection without having to input the limit parameter.

Example 3.8. List Entities, First Page: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
```

```
    <entity key="enAAAAA">
      <label>Brand New Entity</label>
      <ip_addresses>
        <a>127.0.0.4</a>
        <b>127.0.0.5</b>
        <c>127.0.0.6</c>
        <test>127.0.0.7</test>
      </ip_addresses>
      <metadata>
        <all>kinds</all>
        <of>stuff</of>
        <can>go</can>
        <here>null is not a valid value</here>
      </metadata>
    </entity>
  </values>
  <metadata>
    <count>1</count>
    <limit>1</limit>
    <marker/>
    <next_marker>enBBBB</next_marker>
    <next_href>https://cmbeta.api.rackspacecloud.com/v1.0/entities?limit=1&
marker=enBBBB</next_href>
  </metadata>
</container>
```

Example 3.9. List Entities, First Page: JSON

```
{
    "values": [
        {
            "key": "enAAAAA",
            "label": "Brand New Entity",
            "ip_addresses": {
                "a": "127.0.0.4",
                "b": "127.0.0.5",
                "c": "127.0.0.6",
                "test": "127.0.0.7"
            },
            "metadata": {
                "all": "kinds",
                "of": "stuff",
                "can": "go",
                "here": "null is not a valid value"
            }
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 1,
        "marker": null,
        "next_marker": "enBBBB",
        "next_href": "https://cmbeta.api.rackspacecloud.com/v1.0/entities?
limit=1&marker=enBBBB"
    }
}
```

Example 3.10. List Entities, Second Page: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <entity key="enBBBB">
      <label>Brand New Entity 2</label>
      <ip_addresses>
        <a>127.0.0.4</a>
        <b>127.0.0.5</b>
        <c>127.0.0.6</c>
        <test>127.0.0.7</test>
      </ip_addresses>
      <metadata>
        <all>kinds</all>
      </metadata>
    </entity>
  </values>
  <metadata>
    <count>1</count>
    <limit>1</limit>
    <marker>enBBBB</marker>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 3.11. List Entities, Second Page: JSON

```
{
    "values": [
        {
            "key": "enBBBB",
            "label": "Brand New Entity 2",
            "ip_addresses": {
                "a": "127.0.0.4",
                "b": "127.0.0.5",
                "c": "127.0.0.6",
                "test": "127.0.0.7"
            },
            "metadata": {
                "all": "kinds"
            }
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 1,
        "marker": "enBBBB",
        "next_marker": null,
        "next_href": null
    }
}
```

In the JSON representation, paginated collections contain a values property that contains the items in the collection. The link to the next page is located in the metadata object (*metadata.next_href* attribute). Clients must follow the *next_href* link to continue to retrieve additional entities belonging to an account.

3.6. Time Series Collections

Rackspace Cloud Monitoring tracks various time-indexed data sets, such as audits and alarm histories. When accessing a time series collection, two parameters beyond the base paginated collection API are available: *from* and *to*. These parameters take integer values, which are interpreted as timestamps expressed in milliseconds since 00:00:00 UTC on January 1, 1970. If the *to* value is not specified it defaults to the current time. Each time series collection specifies a default offset from the current time, which is used when the *from* value is not supplied. For example, if no *to* or *from* values are specified when retrieving an alarm history, then it will be treated as a query for the last 7 days of data.

The standard paginated collection API is available on top of the time series collection API. For example, performing a **GET** on https://monitoring.api.rackspacecloud.com/v1.0/audits?from=1320044400000&to=1320652800000 should retrieve audits from the first 7 days of November, 2011. However, since no *limit* is supplied a default of 100 is used. Thus, if more than 100 audits exist during that time period only the first 100 will be returned. Further audits within the same period can be retrieved using the *next_href* URL, or by constructing a URL by using the *next_marker* value to append a *marker* to your previous query. For example, to retrieve the next 100 objects beyond the previous query, you might use https://monitoring.api.rackspacecloud.com/v1.0/audits?from=1320044400000&to=1320652800000&marker=16c3de00-038e-11e1-b056-d3a1a3710f94.

To convert a particular time to UTC, you can use the **date +%s000** command or a website such as `http://www.epochconverter.com/`.

3.7. Audits

Every write operation performed against the API (**PUT**, **POST** or **DELETE**) generates an audit record that is stored for 30 days. Audits record a variety of information about the request including the method, URL, headers, query string, transaction ID, the request body and the response code. They also store information about the action performed including a JSON list of the previous state of any modified objects. For example, if you perform an update on an entity, this will record the state of the entity before modification.

You can also use *_who* and *_why* parameters in the query string which will be included in the transaction object.

Audits can be retrieved using the List Audits endpoint.

3.8. Resource Limits

The table below contains default limits for different resources. If you want to create a new resource and your account limit for this resource has been reached the monitoring server will return 400 "Limit has been reached".

Resource	Limit
checks	100
alarms	100

To view the limits that apply to your account, use the Get Limits endpoint.

3.9. Rate Limits

The following table specifies the default rate limits for different API operations.

URI	Limit
/*	50000 requests / 24 hours
/entities/*/test-check	500 requests / 24 hours
/entities/*/test-alarm	500 requests / 24 hours
/notifications/*/test, /test-notifications	200 requests / 24 hours

Each API response also contains the following headers:

Header name	Description
X-RateLimit-Limit	Limit in affect for the current request.
X-RateLimit-Used	How close you are to being rate limited.
X-RateLimit-Window	Rate limit window.
X-RateLimit-Type	Type of the rate limit which applies to the current request.

If you exceed the thresholds established for your account, you'll receive a 400, Limit has been reached response. HTTP responses are returned with a *`Reply-After`* header (number of seconds) to notify the client when it can attempt to try again.

To view the limits that apply to your account, use the Get Limits endpoint.

3.10. Faults

When an error occurs the system returns an HTTP error response code denoting the type of error and additional information in the fault response body.

The following examples show the XML and JSON response body for a `404` notFoundError:

Example 3.12. XML Not Found Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<fault><type>notFoundError</type><code>404</code><message>Item not found.
<message><details>Error Details...</details>
</fault>
```

Example 3.13. JSON Not Found Fault

```
{
    "type": "notFoundError",
    "code": "404",
    "message": "Item not found.",
    "details": "Error Details..."
}
```

The following table contains the possible fault types along with their associated error code and description.

3.10.1. Fault and Error Code Descriptions

Fault Element	Associated Error Codes	Description
badRequest	400	The system received an invalid value in a request
requiredNotFoundError	400	A required related object is not found in the system.
invalidLimit	400	Invalid limit has been specified.
childrenExistError	400	The system cannot perform the requested operation due to remaining child objects. Details attribute also contains a "type" field which specified a type of the existing child object.
alarmParseError	400	Alarm criteria cannot be parsed. Details attribute also contains the following fields: input, error_position, error_line, error_column, error_token, message.
invalidKeyPrefix	400	Provided key prefix is invalid.
cantContactAgent	400	Cannot contact the agent.
limitReached	400	Limit has been reached. Please contact cmbeta@rackspace.com to increase your limit.
rateLimitReached	400	Rate limit has been reached. Please wait before trying again.
indexCardinalityConstraintViolation	400	A resource with the specified index already exists in the system.
unauthorizedError	401	The system received a request from a user that is not authenticated.
forbiddenError	403	The system received a request that the user is forbidden to make.
notFoundError	404	The URL requested is not found in the system.
parentNotFoundError	404	The parent node requested is not found in the system.
requestTooLargeError	413	The response body is too large.
internalError	500	The system suffered an internal failure.
notImplementedError	501	The request is for a feature that has not yet been implemented.
systemFailureError	503	The system is experiencing heavy load or another system failure.

4. Service API Operations

4.1. Core Quick Reference

The following table lists the core service calls for the Rackspace Cloud Monitoring API:

Verb	URI	Description
	Account	
GET	/limits	View an account resource and rate limits.
GET	/account	Retrieve account attributes.
PUT	/account	Update specific account attributes.
	Audit	
GET	/audits	Retrieve API operation history for a given period of time (defaults to last seven days).
	Usage	
GET	/usage	Retrieves usage information for a given period of time (defaults to last seven days).
	Entities	
POST	/entities	Create an entity in the system.
GET	/entities	List all entities associated with your account.
GET	/entities/*entityId*	Examine the entity.
PUT	/entities/*entityId*	Update specific characteristics of an entity.
DELETE	/entities/*entityId*	Delete an entity.
	Checks	
POST	/entities/*entityId*/checks	Create a check.
POST	/entities/*entityId*/test-check	Test a check.
POST	/entities/*entityId*/test-check?debug=true	Test a check and include extra check type specific debugging information if available.
POST	/entities/*entityId*/checks/*checkId*/test	Test an existing check.
POST	/entities/*entityId*/checks/*checkId*/test?debug=true	Test an existing check and include extra check type specific debugging information if available.

Verb	URI	Description
GET	/entities/*entityId*/checks	List all checks associated with your account.
GET	/entities/*entityId*/checks/*checkId*	Get information for a single check.
PUT	/entities/*entityId*/checks/*checkId*	Update specific characteristics of a check.
DELETE	/entities/*entityId*/checks/*checkId*	Delete a check for a specific entity.
Check Types		
GET	/check_types	List all check types associated with your account.
GET	/check_types/*checkTypeId*	Get information for a single check type.
Monitoring Zones		
GET	/monitoring_zones	List all monitoring zones for your account.
GET	/monitoring_zones/*monitoringZoneId*	Get information for a single monitoring zone.
POST	/monitoring_zones/*monitoringZoneId*/traceroute	Perform a traceroute from a collector in the specified monitoring zones.
Alarms		
POST	/entities/*entityId*/alarms	Create an alarm for a specific entity.
POST	/entities/*entityId*/test-alarm	Test an alarm.
GET	/entities/*entityId*/alarms	List all alarms for your account.
GET	/entities/*entityId*/alarms/*alarmId*	Get information for a single alarm.
PUT	/entities/*entityId*/alarms/*alarmId*	Update a specific alarm.
DELETE	/entities/*entityId*/alarms/*alarmId*	Delete an alarm from the specified entity.
Alarm Notification History		
GET	/entities/	Discover which checks this alarm has a notification history for.

Verb	URI	Description
	entityId/alarms/*alarmId*/notification_history	
GET	/entities/*entityId*/alarms/*alarmId*/notification_history/*checkId*	List the notification history for a specific alarm and check.
GET	/entities/*entityId*/alarms/*alarmId*/notification_history/*checkId*/*uuid*	Retrieve a specific notification history event for this alarm and check.
Notification Plans		
POST	/notification_plans	Create a notification plan.
GET	/notification_plans	List all notifications plans for your account.
GET	/notification_plans/*notificationPlanId*	Get information for a single notification plan.
PUT	/notification_plans/*notificationPlanId*	Update a specific notification plan.
DELETE	/notification_plans/*notificationPlanId*	Delete a notification plan from the specified entity.
Notifications		
POST	/notifications	Create a notification.
GET	/notifications	List all notifications for your account.
GET	/notifications/*notificationId*	Get information for a single notification.
PUT	/notifications/*notificationId*	Update a specific notification.
POST	/notifications/*notificationId*/test	Test an existing notification.
POST	/test-notification	Test a notification.
DELETE	/notifications/*notificationId*	Delete a notification from the specified notification plan.
Notification Types		
GET	/notification_types	List all notifications for your account.
GET	/notification_types/*notificationTypeId*	Get information for a single notification type.
Changelogs		
GET	/changelogs/alarms	Lists alarm changelogs for this account.

Verb	URI	Description
Views		
GET	/views/overview	Return the overview view for this account.
GET	/views/overview?id=*entityId*&id=*entityId*	Return the overview view for this account, filtering returned entities by a list of entityIDs.
GET	/views/overview?uri=*URI*&uri=*URI*	Return the overview view for this account, filtering returned entities by a list of entity URIs.
Changelogs		
GET	/changelogs/alarms	Lists alarm changelogs for this account.
Alarm Examples		
GET	/alarm_examples	Lists examples for alarms.
GET	/alarm_examples/*alarmExampleId*	Get a specific alarm example.
POST	/alarm_examples/*alarmExampleId*	Use template parameters to evaluate alarm examples and return the bound criteria.

4.2. Account

4.2.1. Summary

An account contains attributes describing a customer's account. This description contains mostly read only data; however, a few properties can be modified with the API.

4.2.2. Get Account

This endpoint returns information about your account.

Verb	URI	Description
GET	/account	Returns account information.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.1. Get Account Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<account>
  <metadata>
    <key>aValue</key>
  </metadata>
  <webhook_token>token12345</webhook_token>
</account>
```

Example 4.2. Get Account Response: JSON

```
{
    "metadata": {
        "key": "aValue"
    },
    "webhook_token": "token12345"
}
```

4.2.3. Update Account

Verb	URI	Description
PUT	/account	Update properties on an account.

The following table contains the list of attributes that are allowed to be modified with an Account object.

4.2.3.1. Attributes

Name	Description	Validation
metadata	Arbitrary key/value pairs.	• Optional • Hash [String,String between 1 and 255 characters long:String,String between 1 and 255 characters long]
webhook_token		• Optional • String between 10 and 64 characters long

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Example 4.3. Update Account Request: XML

Normal Response Code: (204) This operation does not contain a response body.

Error Response Code: 500

```
<?xml version="1.0" encoding="utf-8"?>
<account>
  <webhook_token>test123456</webhook_token>
</account>
```

Example 4.4. Update Account Request: JSON

```
{
    "webhook_token":"test123456"
}
```

4.2.4. Get Limits

This endpoint returns resource and rate limits that apply to your account.

Verb	URI	Description
GET	/limits	Returns account resource limits.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.5. Get Limits Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<limit>
  <resource>
    <checks>150</checks>
    <alarms>50</alarms>
  </resource>
  <rate>
    <global>
      <limit>50000</limit>
      <used>500</used>
      <window>24 hours</window>
    </global>
    <test_check>
      <limit>500</limit>
      <used>5</used>
      <window>24 hours</window>
    </test_check>
    <test_alarm>
      <limit>500</limit>
      <used>2</used>
      <window>24 hours</window>
    </test_alarm>
  </rate>
</limit>
```

Example 4.6. Get Limits Response: JSON

```
{
    "resource": {
        "checks": 150,
        "alarms": 50
    },
    "rate": {
        "global": {
            "limit": 50000,
            "used": 500,
            "window": "24 hours"
        },
        "test_check": {
            "limit": 500,
            "used": 5,
```

```
            "window": "24 hours"
        },
        "test_alarm": {
            "limit": 500,
            "used": 2,
            "window": "24 hours"
        }
    }
}
```

4.2.5. List Audits

Rackspace Cloud Monitoring records every write operation in an audit. See Audits for more information on how audits are recorded.

Audits are accessible as a Time Series Collection. By default the API queries the last 7 days of audit information.

Verb	URI	Description
GET	/audits	Lists audits for this account.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.7. Audit List Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <audit>
      <headers>
        <host>127.0.0.1</host>
        <user-agent>ele-client/ae0a53c</user-agent>
        <accept>application/xml</accept>
        <content-type>application/xml</content-type>
        <x-auth-token>917ac3575c29e2f27e7236836a732615d62e416d4aead4d5</x-
auth-token>
        <content-length>254</content-length>
        <connection>close</connection>
      </headers>
      <url>/v1.0/entities</url>
      <app>entities</app>
      <query/>
      <txnId>.rh-nzQM.h-ele.r-Kxysr1TO.c-189.ts-1320879585331.v-ae0a53c</
txnId>
      <payload>{"label":"0NwhqrC05xKh22OEYZhcQdwkw","ip_addresses":
{"default":"10.0.0.2"},"metadata":{"all":"kinds","of":"stuff","can":"go",
"here":"null is not a valid value"}}</payload>
      <method>POST</method>
      <account_id>frank</account_id>
      <who/>
      <why/>
      <statusCode>201</statusCode>
    </audit>
    <audit>
      <headers>
```

```
        <host>127.0.0.1</host>
        <user-agent>ele-client/ae0a53c</user-agent>
        <accept>application/xml</accept>
        <content-type>application/xml</content-type>
        <x-auth-token>917ac3575c29e2f27e7236836a732615d62e416d4aead4d5</x-
auth-token>
        <content-length>51</content-length>
        <connection>close</connection>
      </headers>
      <url>/v1.0/entities/enuAc9qZRB</url>
      <app>entities</app>
      <query/>
      <txnId>.rh-nzQM.h-ele.r-lobnFBE4.c-195.ts-1320879585945.v-ae0a53c</
txnId>
      <payload>{"label":"testing.example.com"}</payload>
      <method>PUT</method>
      <account_id>frank</account_id>
      <who/>
      <why/>
      <statusCode>204</statusCode>
    </audit>
  </values>
  <metadata>
    <count>2</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.8. Audit List Response: JSON

```
{
    "values": [
        {
            "headers": {
                "host": "127.0.0.1",
                "user-agent": "ele-client/ae0a53c",
                "accept": "application/xml",
                "content-type": "application/xml",
                "x-auth-token":
 "917ac3575c29e2f27e7236836a732615d62e416d4aead4d5",
                "content-length": 254,
                "connection": "close"
            },
            "url": "/v1.0/entities",
            "app": "entities",
            "query": null,
            "txnId": ".rh-nzQM.h-ele.r-Kxysr1TO.c-189.ts-1320879585331.v-
ae0a53c",
            "payload": "{\"label\":\"0NwhqrC05xKh22OEYZhcQdwkw\",\
"ip_addresses\":{\"default\":\"10.0.0.2\"},\"metadata\":{\"all\":\"kinds\",\
"of\":\"stuff\",\"can\":\"go\",\"here\":\"null is not a valid value\"}}",
            "method": "POST",
            "account_id": "frank",
            "who": null,
            "why": null,
```

```
            "statusCode": 201
        },
        {
            "headers": {
                "host": "127.0.0.1",
                "user-agent": "ele-client/ae0a53c",
                "accept": "application/xml",
                "content-type": "application/xml",
                "x-auth-token":
 "917ac3575c29e2f27e7236836a732615d62e416d4aead4d5",
                "content-length": 51,
                "connection": "close"
            },
            "url": "/v1.0/entities/enuAc9qZRB",
            "app": "entities",
            "query": null,
            "txnId": ".rh-nzQM.h-ele.r-lobnFBE4.c-195.ts-1320879585945.v-
ae0a53c",
            "payload": "{\"label\":\"testing.example.com\"}",
            "method": "PUT",
            "account_id": "frank",
            "who": null,
            "why": null,
            "statusCode": 204
        }
    ],
    "metadata": {
        "count": 2,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.3. Entities

4.3.1. Summary

An entity is the target of what you are monitoring. For example, you can create an entity to monitor your website, a particular web service, or your Rackspace server or server instance. Note that an entity represents only one item in the monitoring system. For example, if you wanted to monitor each server in a cluster, you would create an entity for each of the servers. You would not create a single entity to represent the entire cluster.

An entity can have multiple checks associated with it. This allows you to check multiple services on the same host by creating multiple checks on the same entity, instead of multiple entities each with a single check.

When you create a new entity in the monitoring system, you specify the follow parameters:

- A meaningful name for the entity
- The IP address(es) for the entity (optional)
- The meta-data that the monitoring system uses if an alarm is triggered (optional)

These parameters are further described in Attributes. See Create Entities for an example of how to create an entity.

4.3.2. Attributes

Name	Description	Validation
label	Defines a name for the entity.	• String between 1 and 255 characters long • Non-empty string
agent_id	Agent to which this entity is bound to.	• Optional • String matching the regex /^[-\.\w]{1,255}$/
ip_addresses	Hash of IP addresses that can be referenced by checks on this entity.	• Optional • Hash [String,String between 1 and 64 characters long:IPv4 or IPv6 address]
metadata	Arbitrary key/value pairs that are passed during the alerting phase.	• Optional • Hash [String,String between 1 and 255 characters long:String,String between 1 and 255 characters long]

4.3.3. Create Entity

Verb	URI	Description
POST	/entities	Creates a new entity. Specify the entity's characteristics using a valid set of parameters from the table shown in the Attributes section above.

Normal Response Code: (201) 'Location' header contains a link to the newly created entity.

Error Response Codes: 400, 401, 403, 500, 503

Once you have created an entity, you will typically create a check for it. See Create Check.

Example 4.9. Entity Create Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<entity>
  <label>Brand New Entity</label>
  <ip_addresses>
    <default>127.0.0.4</default>
    <b>127.0.0.5</b>
    <c>127.0.0.6</c>
    <test>127.0.0.7</test>
  </ip_addresses>
  <metadata>
    <all>kinds</all>
    <of>stuff</of>
    <can>go</can>
    <here>null is not a valid value</here>
  </metadata>
</entity>
```

Example 4.10. Entity Create Request: JSON

```
{
    "label": "Brand New Entity",
    "ip_addresses": {
        "default": "127.0.0.4",
        "b": "127.0.0.5",
        "c": "127.0.0.6",
        "test": "127.0.0.7"
    },
    "metadata": {
        "all": "kinds",
        "of": "stuff",
        "can": "go",
        "here": "null is not a valid value"
    }
}
```

4.3.4. List Entities

Verb	URI	Description
GET	/entities	Lists the entities for this particular account.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.11. List Entities Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <entity id="enAAAAA">
      <label>Brand New Entity</label>
      <ip_addresses>
        <default>127.0.0.4</default>
        <b>127.0.0.5</b>
        <c>127.0.0.6</c>
        <test>127.0.0.7</test>
      </ip_addresses>
      <metadata>
        <all>kinds</all>
        <of>stuff</of>
        <can>go</can>
        <here>null is not a valid value</here>
      </metadata>
    </entity>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
```

```
  </metadata>
</container>
```

Example 4.12. List Entities Response: JSON

```
{
    "values": [
        {
            "id": "enAAAAA",
            "label": "Brand New Entity",
            "ip_addresses": {
                "default": "127.0.0.4",
                "b": "127.0.0.5",
                "c": "127.0.0.6",
                "test": "127.0.0.7"
            },
            "metadata": {
                "all": "kinds",
                "of": "stuff",
                "can": "go",
                "here": "null is not a valid value"
            }
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.3.5. Get Entity

Verb	URI	Description
GET	/entities/*entityId*	Retrieves the current state of an entity.

This operation allows you to retrieve the current state of an entity.

Normal Response Code: 200

Error Response Codes: 401, 403, 404, 500, 503

Example 4.13. Get Entity Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<entity id="enAAAAA">
  <label>Brand New Entity</label>
  <ip_addresses>
    <default>127.0.0.4</default>
    <b>127.0.0.5</b>
    <c>127.0.0.6</c>
    <test>127.0.0.7</test>
```

```
  </ip_addresses>
  <metadata>
    <all>kinds</all>
    <of>stuff</of>
    <can>go</can>
    <here>null is not a valid value</here>
  </metadata>
</entity>
```

Example 4.14. Get Entity Response: JSON

```
{
    "id": "enAAAAA",
    "label": "Brand New Entity",
    "ip_addresses": {
        "default": "127.0.0.4",
        "b": "127.0.0.5",
        "c": "127.0.0.6",
        "test": "127.0.0.7"
    },
    "metadata": {
        "all": "kinds",
        "of": "stuff",
        "can": "go",
        "here": "null is not a valid value"
    }
}
```

4.3.6. Update Entity

Verb	URI	Description
PUT	/entities/*entityId*	Updates an entity specified by the entityId. Since partial updates to an entity are acceptable, you may specify only the parameters you would like to update.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Example 4.15. Entity Update Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<entity>
  <label>Brand New Entity 2</label>
</entity>
```

Example 4.16. Entity Update Request: JSON

```
{
    "label": "Brand New Entity 2"
}
```

4.3.7. Delete Entity

Verb	URI	Description
DELETE	/entities/*entityId*	Deletes an entity from your account. Also deletes any checks and alarms defined for that entity.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 401, 403, 404, 500, 503

4.4. Checks

4.4.1. Summary

A check is one of the foundational building blocks of the monitoring system. The check determines the parts or pieces of the entity that you want to monitor, the monitoring frequency, how many monitoring zones are originating the check, and so on. When you create a new check in the monitoring system, you specify the following information:

- A name for the check
- The check's parent entity
- The type of check you're creating
- Details of the check
- The monitoring zones that will launch the check

The attributes you use for creating checks, including optional parameters, are described in the following Attributes section. Create Check [32] provides an example of how to create a new check.

The check, as created, will not trigger alert messages until you create an alarm to generate notifications, to enable the creation of a single alarm that acts upon multiple checks (e.g. alert if any of ten different servers stops responding) or multiple alarms off of a single check. (e.g. ensure both that a HTTPS server is responding and that it has a valid certificate).

Note

Checks are strictly associated with a parent entity, therefore the REST URLs for checks are underneath the entity that they are associated with.

4.4.2. Attributes

Name	Description	Validation
Attributes used for all checks		

Name	Description	Validation
type	The type of check.	• Immutable • String • One of (remote.dns, remote.ssh, remote.smtp, remote.http, remote.tcp, remote.ping, remote.ftp-banner, remote.imap-banner, remote.pop3-banner, remote.smtp-banner, remote.postgresql-banner, remote.telnet-banner, remote.mysql-banner, remote.mssql-banner)
details	Details specific to the check type.	• Optional • Hash [String,String between 1 and 255 characters long:Optional]
disabled	Disables the check.	• Optional • Boolean
label	A friendly label for a check.	• Optional • String between 1 and 255 characters long
metadata	Arbitrary key/value pairs.	• Optional • Hash [String,String between 1 and 255 characters long:String,String between 1 and 255 characters long]
period	The period in seconds for a check. The value must be greater than the minimum period set on your account.	• Optional • Integer • Value (30..1800)
timeout	The timeout in seconds for a check. This has to be less than the period.	• Optional • Integer • Value (2..1800)
Attributes used for remote checks		
monitoring_zones_poll	List of monitoring zones to poll from. Note: This argument is only required for remote (non-agent) checks	• Optional • Array [String]
target_alias	A key in the entity's 'ip_addresses' hash used to resolve this check to an IP address. This parameter is mutually exclusive with target_hostname.	• Optional • String between 1 and 64 characters long
target_hostname	The hostname this check should target. This parameter is mutually exclusive with target_alias.	• Optional • Valid hostname, IPv4 or IPv6 address
target_resolver	Determines how to resolve the check target.	• Optional • One of (IPv4, IPv6)

Note

target_alias and target_hostname are mutually exclusive; one, but not both, must be provided.

4.4.3. Create Check

Verb	URI	Description
POST	/entities/*entityId*/checks	Create a new check and associate it with an entity using the parameters listed in Attributes.

Normal Response Code: (201) 'Location' header contains a link to the newly created check.

Error Response Codes: 400, 401, 403, 500, 503

Example 4.17. Check Create Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<check>
  <label>Website check 1</label>
  <type>remote.http</type>
  <details>
    <url>http://www.foo.com</url>
    <method>GET</method>
  </details>
  <monitoring_zones_poll>
    <monitoring_zone_id>mzA</monitoring_zone_id>
  </monitoring_zones_poll>
  <timeout>30</timeout>
  <period>100</period>
  <target_alias>default</target_alias>
</check>
```

Example 4.18. Check Create Request: JSON

```
{
    "label": "Website check 1",
    "type": "remote.http",
    "details": {
        "url": "http://www.foo.com",
        "method": "GET"
    },
    "monitoring_zones_poll": [
        "mzA"
    ],
    "timeout": 30,
    "period": 100,
    "target_alias": "default"
}
```

4.4.4. Test Check

Verb	URI	Description
POST	/entities/*entityId*/test-check/	Test a check before creating it.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 404, 500, 503

This operation causes Rackspace Cloud Monitoring to attempt to run the check on the specified monitoring zones and return the results. This allows you to test the check parameters in a single step before the check is actually created in the system.

Note

You can copy the results of a test check response and paste it directly into a test alarm.

Example 4.19. Test Check Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<test_check>
  <type>remote.http</type>
  <details>
    <url>http://www.foo.com</url>
    <method>GET</method>
  </details>
  <monitoring_zones_poll>
    <monitoring_zone_id>mzA</monitoring_zone_id>
  </monitoring_zones_poll>
  <timeout>30</timeout>
  <target_alias>default</target_alias>
</test_check>
```

Example 4.20. Test Check Request: JSON

```
{
    "type": "remote.http",
    "details": {
        "url": "http://www.foo.com",
        "method": "GET"
    },
    "monitoring_zones_poll": [
        "mzA"
    ],
    "timeout": 30,
    "target_alias": "default"
}
```

Example 4.21. Test Check Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<checks_data>
  <check_data>
    <timestamp>1319222001982</timestamp>
    <monitoring_zone_id>mzxJ4L2IU</monitoring_zone_id>
    <available>true</available>
```

```
    <status>code=200,rt=0.257s,bytes=0</status>
    <metrics>
      <bytes>
        <type>i</type>
        <data>0</data>
      </bytes>
      <tt_firstbyte>
        <type>I</type>
        <data>257</data>
      </tt_firstbyte>
      <tt_connect>
        <type>I</type>
        <data>128</data>
      </tt_connect>
      <code>
        <type>s</type>
        <data>200</data>
      </code>
      <duration>
        <type>I</type>
        <data>257</data>
      </duration>
    </metrics>
  </check_data>
</checks_data>
```

Example 4.22. Test Check Response: JSON

```
[
    {
        "timestamp": 1319222001982,
        "monitoring_zone_id": "mzxJ4L2IU",
        "available": true,
        "status": "code=200,rt=0.257s,bytes=0",
        "metrics": {
            "bytes": {
                "type": "i",
                "data": "0"
            },
            "tt_firstbyte": {
                "type": "I",
                "data": "257"
            },
            "tt_connect": {
                "type": "I",
                "data": "128"
            },
            "code": {
                "type": "s",
                "data": "200"
            },
            "duration": {
                "type": "I",
                "data": "257"
            }
        }
    }
]
```

4.4.4.1. Test Check And Include Debug Information

Verb	URI	Description
POST	/entities/*entityId*/test-check?debug=true	Test a check and include extra check type-specific debugging information, if available.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 404, 500, 503

This operation causes Rackspace Cloud Monitoring to attempt to run the check on the specified monitoring zones and return the results. This allows you to test the check parameters in a single step before the check is actually created in the system. This call also includes debug information. Currently debug information is only available for the remote.http check and includes the response body.

Note

Only first 512 KB of the response body is read. If the response body is longer, it is truncated to 512KB.

Example 4.23. Test Check Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<test_check>
  <type>remote.http</type>
  <details>
    <url>http://www.foo.com</url>
    <method>GET</method>
  </details>
  <monitoring_zones_poll>
    <monitoring_zone_id>mzA</monitoring_zone_id>
  </monitoring_zones_poll>
  <timeout>30</timeout>
  <target_alias>default</target_alias>
</test_check>
```

Example 4.24. Test Check Request: JSON

```
{
    "type": "remote.http",
    "details": {
        "url": "http://www.foo.com",
        "method": "GET"
    },
    "monitoring_zones_poll": [
        "mzA"
    ],
    "timeout": 30,
    "target_alias": "default"
}
```

Example 4.25. Test Check With Debug Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<checks_data>
  <check_data>
    <timestamp>1319222001982</timestamp>
    <monitoring_zone_id>mzxJ4L2IU</monitoring_zone_id>
    <available>true</available>
    <status>code=200,rt=0.257s,bytes=0</status>
    <metrics>
      <bytes>
        <type>i</type>
        <data>0</data>
      </bytes>
      <tt_firstbyte>
        <type>I</type>
        <data>257</data>
      </tt_firstbyte>
      <tt_connect>
        <type>I</type>
        <data>128</data>
      </tt_connect>
      <code>
        <type>s</type>
        <data>200</data>
      </code>
      <duration>
        <type>I</type>
        <data>257</data>
      </duration>
    </metrics>
    <debug_info>
      <body>&lt;html&gt;&lt;body&gt;My shiny website&lt;/body&gt;&lt;/html&
gt;</body>
    </debug_info>
  </check_data>
</checks_data>
```

Example 4.26. Test Check With Debug Response: JSON

```
[
    {
        "timestamp": 1319222001982,
        "monitoring_zone_id": "mzxJ4L2IU",
        "available": true,
        "status": "code=200,rt=0.257s,bytes=0",
        "metrics": {
            "bytes": {
                "type": "i",
                "data": "0"
            },
            "tt_firstbyte": {
                "type": "I",
                "data": "257"
            },
            "tt_connect": {
                "type": "I",
```

```
                "data": "128"
            },
            "code": {
                "type": "s",
                "data": "200"
            },
            "duration": {
                "type": "I",
                "data": "257"
            }
        },
        "debug_info": {
            "body": "<html><body>My shiny website</body></html>"
        }
    }
]
```

4.4.5. Test Existing Check

Verb	URI	Description
POST	/entities/*entityId*/checks/*checkId*/test	Test a check inline.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 404, 500, 503

This operation causes Rackspace Cloud Monitoring to attempt to run the check on the specified monitoring zones and return the results. This allows you to test the check parameters.

Note

You can copy the results of a test check response and paste it directly into a test alarm.

Note

There is no request body, just posting to the url executes the request.

Example 4.27. Test Existing Check Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<checks_data>
  <check_data>
    <timestamp>1319222001982</timestamp>
    <monitoring_zone_id>mzxJ4L2IU</monitoring_zone_id>
    <available>true</available>
    <status>code=200,rt=0.257s,bytes=0</status>
    <metrics>
      <bytes>
        <type>i</type>
        <data>0</data>
      </bytes>
      <tt_firstbyte>
        <type>I</type>
```

```
        <data>257</data>
      </tt_firstbyte>
      <tt_connect>
        <type>I</type>
        <data>128</data>
      </tt_connect>
      <code>
        <type>s</type>
        <data>200</data>
      </code>
      <duration>
        <type>I</type>
        <data>257</data>
      </duration>
    </metrics>
  </check_data>
</checks_data>
```

Example 4.28. Test Existing Check Response: JSON

```
[
    {
        "timestamp": 1319222001982,
        "monitoring_zone_id": "mzxJ4L2IU",
        "available": true,
        "status": "code=200,rt=0.257s,bytes=0",
        "metrics": {
            "bytes": {
                "type": "i",
                "data": "0"
            },
            "tt_firstbyte": {
                "type": "I",
                "data": "257"
            },
            "tt_connect": {
                "type": "I",
                "data": "128"
            },
            "code": {
                "type": "s",
                "data": "200"
            },
            "duration": {
                "type": "I",
                "data": "257"
            }
        }
    }
]
```

4.4.6. List Checks

Verb	URI	Description
GET	/entities/*entityId*/checks/	Lists the checks associated with a given entityId.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.29. List Checks Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <check id="chAAAA">
      <label>Website check 1</label>
      <type>remote.http</type>
      <details>
        <url>http://www.foo.com</url>
        <method>GET</method>
      </details>
      <monitoring_zones_poll>
        <monitoring_zone_id>mzA</monitoring_zone_id>
      </monitoring_zones_poll>
      <timeout>30</timeout>
      <period>100</period>
      <target_alias>default</target_alias>
    </check>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.30. List Checks Response: JSON

```
{
    "values": [
        {
            "id": "chAAAA",
            "label": "Website check 1",
            "type": "remote.http",
            "details": {
                "url": "http://www.foo.com",
                "method": "GET"
            },
            "monitoring_zones_poll": [
                "mzA"
            ],
            "timeout": 30,
            "period": 100,
            "target_alias": "default"
        }
    ],
    "metadata": {
```

```
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.4.7. Get Check

Verb	URI	Description
GET	/entities/*entityId*/checks/*checkId*	Returns the specified check.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 404, 413, 500, 503

Example 4.31. Get Check Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<check id="chAAAA">
  <label>Website check 1</label>
  <type>remote.http</type>
  <details>
    <url>http://www.foo.com</url>
    <method>GET</method>
  </details>
  <monitoring_zones_poll>
    <monitoring_zone_id>mzA</monitoring_zone_id>
  </monitoring_zones_poll>
  <timeout>30</timeout>
  <period>100</period>
  <target_alias>default</target_alias>
</check>
```

Example 4.32. Get Check Response: JSON

```
{
    "id": "chAAAA",
    "label": "Website check 1",
    "type": "remote.http",
    "details": {
        "url": "http://www.foo.com",
        "method": "GET"
    },
    "monitoring_zones_poll": [
        "mzA"
    ],
    "timeout": 30,
    "period": 100,
    "target_alias": "default"
```

```
}
```

4.4.8. Update Checks

Verb	URI	Description
PUT	/entities/*entityId*/checks/*checkId*	Updates a check with the specified checkId.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Note

Updating monitoring zones can effect the billing of the product.

Example 4.33. Check Update Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<check>
  <label>New check label</label>
  <timeout>10</timeout>
</check>
```

Example 4.34. Check Update Request: JSON

```
{
    "label": "New check label",
    "timeout": 10
}
```

4.4.9. Delete Checks

Verb	URI	Description
DELETE	/entities/*entityId*/checks/*checkId*	Deletes a check from your account.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 401, 403, 404, 500, 503

4.5. Check Types

4.5.1. Summary

Each check within the Rackspace Cloud Monitoring has a designated check type. The check type instructs the monitoring system how to check the monitored resource.

Note

Users cannot create, update or delete check types.

Check types for commonly encountered web protocols, such as HTTP (remote.http), IMAP (remote.imap-banner) , SMTP (remote.stmp), and DNS (remote.dns) are provided. Monitoring commonly encountered infrastructure servers like MySQL (remote.mysql-banner) and PostgreSQL (remote.postgresql-banner) are also available. Monitoring custom server uptime can be accomplished with the remote.tcp banner check to check for a protocol-defined banner at the beginning of a connection. Gathering metrics from server software to create alerts against can be accomplished using the remote.http check type and the 'extract' attribute to define the format.

Checks generate metrics that alarms will alert based upon. The metrics generated often times depend on the check's parameters.. for example, using the 'extract' attribute on the remote.http check, however the default metrics will always be present. To determine the exact metrics available, the Test Check API is provided

4.5.2. Attributes

Name	Description	Validation
type	The name of the supported check type.	• String
fields	Check type fields.	• Optional • Array [Optional]

4.5.3. Available Check Types and Fields

4.5.3.1. remote.dns

Field	Description	Validation
query	DNS Query	• String between 1 and 255 characters long
record_type	DNS Record Type	• String matching the regex /^(A|AAAA|TXT|MX|SOA|CNAME|PTR|NS|MB|MD|MF|MG|MR||)+$/
port	Port number (default: 53)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive

The remote.dns check will run a DNS check against a given target. This check should assist in verifying functionality of a DNS server, for example ensuring that it is publishing the domains you think that it should be publishing.

4.5.3.1.1. standard metrics

Metric	Description	Type
answer	The list of space seperated IP addresses for the specified name resolution. (ex: "74.125.225.80, 74.125.225.81, 74.125.225.82, 74.125.225.83, 74.125.225.84")	String
rtt	The roundtrip time to execute a remote.dns check. (ex: 17.814)	Double

Metric	Description	Type
ttl	The TTL returned by the dns query. (ex: 300)	Uint32

4.5.3.2. remote.ssh

Field	Description	Validation
port	Port number (default: 22)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive

The remote.ssh check will attempt to SSH to a target.

4.5.3.2.1. standard metrics

Metric	Description	Type
duration	The time took to finish executing the check in milliseconds. (ex: 59)	Uint32
fingerprint	The ssh fingerprint used to verify identity. (ex: "4ea6a02e7e52469c3274d59f61bf2f14")	String

4.5.3.3. remote.smtp

Field	Description	Validation
ehlo	EHLO parameter	• Optional • String between 1 and 255 characters long
from	From parameter	• Optional • String between 1 and 255 characters long
payload	Specifies the payload	• Optional • String between 1 and 1024 characters long
port	Port number (default: 25)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
starttls	Should the connection be upgraded to TLS/SSL	• Optional • Boolean
to	To parameter	• Optional • String between 1 and 255 characters long

The remote.smtp check will attempt to connect to a SMTP mail server, send an email from the 'from' parameter, to the 'to' parameter, with a payload specified by the 'payload' parameter setting the EHLO from host to the value in 'ehlo'

4.5.3.4. remote.http

Field	Description	Validation
url	Target URL	• URL • String between 1 and 8096 characters long
auth_password	Optional auth password	• Optional • String between 1 and 255 characters long
auth_user	Optional auth user	• Optional • String between 1 and 255 characters long
body	Body match regular expression (body is limited to 100k)	• Optional • String between 1 and 255 characters long
body_matches	Body match regular expressions (body is limited to 100k)	• Optional • Hash [String,String between 1 and 50 characters long,String matching the regex /^[-_ a-z0-9]+$/i:String,String between 1 and 255 characters long] • Array or object with number of items between 0 and 4
extract	Regex to capture key/value pairs.	• Optional • String between 1 and 255 characters long
follow_redirects	Follow redirects (default: true)	• Optional • Boolean
headers	Arbitrary headers which are sent with the request.	• Optional • Hash [String,String between 1 and 50 characters long:String,String between 1 and 50 characters long] • Array or object with number of items between 0 and 10 • A value which is not one of: content-length, user-agent, host, connection, keep-alive, transfer-encoding, upgrade
method	HTTP method (default: GET)	• Optional • String • One of (HEAD, GET, POST, PUT, DELETE, INFO)
payload	Specify a request body (limited to 1024 characters). If following a redirect, payload will only be sent to first location	• Optional • String between 0 and 1024 characters long
ssl	Enable SSL	• Optional • Boolean

The remote.http check will try to connect to the server and retrieve the specified URL using the specified method, optionally with the password and user for authentication, using SSL, and checking the body with a regex. This can be used to test that a web application running on a server is responding without generating error messages. It can also test if the SSL certificate is valid and extract metrics from a properly formatted page

4.5.3.4.1. standard metrics

Metric	Description	Type
body_match	The string representing the body match specified in a remote.http check. (ex: "<!doctype html><html><head><meta http-equiv="content-type" content="text/html; c")	String
bytes	The number of bytes returned from a response payload. (ex: 10577)	Int32
cert_end	The absolute timestamp in seconds for the certificate expiration. This is only available performing a check on an HTTPS server. (ex: 1324252799)	Uint32
cert_end_in	The relative timestamp in seconds til certification expiration. This is only available performing a check on an HTTPS server. (ex: 4003810)	Int32
cert_error	A string describing a certificate error in our validation. This is only available performing a check on an HTTPS server. (ex: "certificate not trusted")	String
cert_issuer	The issue string for the certificate. This is only available performing a check on an HTTPS server. (ex: "/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA")	String
cert_start	The absolute timestamp of the issue of the certificate. This is only available performing a check on an HTTPS server. (ex: 1261094400)	Uint32
cert_subject	The subject of the certificate. This is only available performing a check on an HTTPS server. (ex: "/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com")	String
code	The status code returned. (ex: "200")	String
duration	The time took to finish executing the check in milliseconds. (ex: 237)	Uint32
tt_connect	The time to connect measured in milliseconds. (ex: 50)	Uint32
tt_firstbyte	The time to first byte measured in milliseconds. (ex: 237)	Uint32

4.5.3.5. remote.tcp

Field	Description	Validation
port	Port number	• Whole number (may be zero padded)

Field	Description	Validation
		• Integer between 1-65535 inclusive
banner_match	Banner match regex.	• Optional • String
body_match	Body match regex. Key/Values are captured when matches are specified within the regex. Note: Maximum body size 1024 bytes.	• Optional • String
send_body	Send a body. If a banner is provided the body is sent after the banner is verified.	• Optional • String
ssl	Enable SSL	• Optional • Boolean

The remote.tcp check will attempt to connect to a host and port, and optionally issue a banner match to ensure that the service is responding as specified. This can be used to test services that are not covered by the existing HTTP, SMTP, SSH, MySQL, etc. checks.

4.5.3.5.1. standard metrics

Metric	Description	Type
banner	The string sent from the server on connect. (ex: "SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7 ")	String
banner_match	The matched string from the **banner_match** regular expression specified during check creation. (ex: "SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7 ")	String
duration	The time took to finish executing the check in milliseconds. (ex: 10)	Uint32
tt_connect	The time to connect measured in milliseconds. (ex: 0)	Uint32
tt_firstbyte	The time to first byte measured in milliseconds. (ex: 10)	Uint32

4.5.3.6. remote.ping

Field	Description	Validation
count	Number of pings to send within a single check	• Optional • Whole number (may be zero padded) • Value (1..50)

The remote.ping check will attempt to ping a server.

4.5.3.6.1. standard metrics

Metric	Description	Type
available	The whole number representing the percent of pings that returned back for a remote.ping check. (ex: 60)	Double

Metric	Description	Type
average	The average response time for all ping packets sent out and later retrieved. (ex: 0.0002513333359578)	Double
count	The number of ping ping's returned. (ex: 5)	Int32
maximum	The maximum roundtrip time of an ICMP packet. (ex: 0.0002770000137389)	Double
minimum	The minimum roundtrip time of an ICMP packet. (ex: 0.0002329999988433)	Double

4.5.3.7. remote.ftp-banner

Field	Description	Validation
port	Port number (default: 21)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive

The remote.ftp-banner check will attempt to connect to a FTP server and verify that it responds to the connection.

4.5.3.7.1. standard metrics

Metric	Description	Type
banner	The string sent from the server on connect. (ex: "220- ")	String
banner_match	The matched string from the **banner_match** regular expression specified during check creation. (ex: "220- ")	String
duration	The time took to finish executing the check in milliseconds. (ex: 119)	Uint32
tt_connect	The time to connect measured in milliseconds. (ex: 59)	Uint32
tt_firstbyte	The time to first byte measured in milliseconds. (ex: 119)	Uint32

4.5.3.8. remote.imap-banner

Field	Description	Validation
port	Port number (default: 143)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.imap-banner check will attempt to connect to an IMAP server and verify that it response to the connection

4.5.3.9. remote.pop3-banner

Field	Description	Validation
port	Port number (default: 110)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.pop3-banner check will attempt to connect to a POP3 mailbox server and verify that it responds to the connection.

4.5.3.10. remote.smtp-banner

Field	Description	Validation
port	Port number (default: 25)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.smtp-banner check will attempt to connect to a SMTP mail server and verify that a HELO/EHLO is received.

4.5.3.10.1. standard metrics

Metric	Description	Type
tt_connect	The time to connect measured in milliseconds. (ex: 34)	Uint32

4.5.3.11. remote.postgresql-banner

Field	Description	Validation
port	Port number (default: 5432)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.postgresql-banner check will attempt to connect to a PostgreSQL database server and verify that it is accepting connections.

4.5.3.12. remote.telnet-banner

Field	Description	Validation
banner_match	Banner to check	• Optional

Field	Description	Validation
		• String between 1 and 255 characters long
port	Port number (default: 23)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.telnet-banner check will attempt to connect to a Telnet (or similar protocol) server and verify that an appropriate banner is received.

4.5.3.13. remote.mysql-banner

Field	Description	Validation
port	Port number (default: 3306)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.mysql-banner check will attempt to connect to a MySQL database server and verify that it is accepting connections.

4.5.3.14. remote.mssql-banner

Field	Description	Validation
port	Port number (default: 1433)	• Optional • Whole number (may be zero padded) • Integer between 1-65535 inclusive
ssl	Enable SSL	• Optional • Boolean

The remote.mssql-banner check will attempt to connect to a Microsoft SQL database server and verify that it is accepting connections.

4.5.4. Create Check Type

Users cannot create check types.

4.5.5. List Check Types

Verb	URI	Description
GET	/check_types	List all the available check types.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.35. List Check Types: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <check_type id="remote.dns">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 53)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>query</name>
          <description>DNS Query</description>
          <optional>false</optional>
        </field>
        <field>
          <name>record_type</name>
          <description>DNS Record Type</description>
          <optional>false</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.ssh">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 22)</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.smtp">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 25)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>ehlo</name>
          <description>EHLO parameter</description>
          <optional>true</optional>
        </field>
        <field>
          <name>from</name>
          <description>From parameter</description>
          <optional>true</optional>
        </field>
        <field>
          <name>to</name>
```

```
          <description>To parameter</description>
          <optional>true</optional>
        </field>
        <field>
          <name>payload</name>
          <description>Specifies the payload</description>
          <optional>true</optional>
        </field>
        <field>
          <name>starttls</name>
          <description>Should the connection be upgraded to TLS/SSL</
description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.http">
      <type>remote</type>
      <fields>
        <field>
          <name>url</name>
          <description>Target URL</description>
          <optional>false</optional>
        </field>
        <field>
          <name>body</name>
          <description>Body match regular expression (body is limited to
 100k)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>headers</name>
          <description>Arbitrary headers which are sent with the request.</
description>
          <optional>true</optional>
        </field>
        <field>
          <name>body_matches</name>
          <description>Body match regular expressions (body is limited to
 100k)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
        <field>
          <name>method</name>
          <description>HTTP method (default: GET)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>auth_user</name>
          <description>Optional auth user</description>
          <optional>true</optional>
        </field>
        <field>
          <name>auth_password</name>
          <description>Optional auth password</description>
```

```
          <optional>true</optional>
        </field>
        <field>
          <name>extract</name>
          <description>Regex to capture key/value pairs.</description>
          <optional>true</optional>
        </field>
        <field>
          <name>follow_redirects</name>
          <description>Follow redirects (default: true)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>payload</name>
          <description>Specify a request body (limited to 1024 characters).
 If following a redirect, payload will only be sent to first location</
description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.tcp">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number</description>
          <optional>false</optional>
        </field>
        <field>
          <name>banner_match</name>
          <description>Banner match regex.</description>
          <optional>true</optional>
        </field>
        <field>
          <name>send_body</name>
          <description>Send a body. If a banner is provided the body is sent
 after the banner is verified.</description>
          <optional>true</optional>
        </field>
        <field>
          <name>body_match</name>
          <description>Body match regex. Key/Values are captured when matches
 are specified within the regex. Note: Maximum body size 1024 bytes.</
description>
          <optional>true</optional>
        </field>
        <field>
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.ping">
      <type>remote</type>
      <fields>
        <field>
          <name>count</name>
```

```
          <description>Number of pings to send within a single check</
description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.ftp-banner">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 21)</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.imap-banner">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 143)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.pop3-banner">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 110)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
    <check_type id="remote.smtp-banner">
      <type>remote</type>
      <fields>
        <field>
          <name>port</name>
          <description>Port number (default: 25)</description>
          <optional>true</optional>
        </field>
        <field>
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
      </fields>
```

```
  </check_type>
  <check_type id="remote.postgresql-banner">
    <type>remote</type>
    <fields>
      <field>
        <name>port</name>
        <description>Port number (default: 5432)</description>
        <optional>true</optional>
      </field>
      <field>
        <name>ssl</name>
        <description>Enable SSL</description>
        <optional>true</optional>
      </field>
    </fields>
  </check_type>
  <check_type id="remote.telnet-banner">
    <type>remote</type>
    <fields>
      <field>
        <name>port</name>
        <description>Port number (default: 23)</description>
        <optional>true</optional>
      </field>
      <field>
        <name>banner_match</name>
        <description>Banner to check</description>
        <optional>true</optional>
      </field>
      <field>
        <name>ssl</name>
        <description>Enable SSL</description>
        <optional>true</optional>
      </field>
    </fields>
  </check_type>
  <check_type id="remote.mysql-banner">
    <type>remote</type>
    <fields>
      <field>
        <name>port</name>
        <description>Port number (default: 3306)</description>
        <optional>true</optional>
      </field>
      <field>
        <name>ssl</name>
        <description>Enable SSL</description>
        <optional>true</optional>
      </field>
    </fields>
  </check_type>
  <check_type id="remote.mssql-banner">
    <type>remote</type>
    <fields>
      <field>
        <name>port</name>
        <description>Port number (default: 1433)</description>
        <optional>true</optional>
      </field>
      <field>
```

```
          <name>ssl</name>
          <description>Enable SSL</description>
          <optional>true</optional>
        </field>
      </fields>
    </check_type>
  </values>
  <metadata>
    <count>14</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.36. List Check Types: JSON

```
{
    "values": [
        {
            "id": "remote.dns",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 53)",
                    "optional": true
                },
                {
                    "name": "query",
                    "description": "DNS Query",
                    "optional": false
                },
                {
                    "name": "record_type",
                    "description": "DNS Record Type",
                    "optional": false
                }
            ]
        },
        {
            "id": "remote.ssh",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 22)",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.smtp",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
```

```
                    "description": "Port number (default: 25)",
                    "optional": true
                },
                {
                    "name": "ehlo",
                    "description": "EHLO parameter",
                    "optional": true
                },
                {
                    "name": "from",
                    "description": "From parameter",
                    "optional": true
                },
                {
                    "name": "to",
                    "description": "To parameter",
                    "optional": true
                },
                {
                    "name": "payload",
                    "description": "Specifies the payload",
                    "optional": true
                },
                {
                    "name": "starttls",
                    "description": "Should the connection be upgraded to TLS/
SSL",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.http",
            "type": "remote",
            "fields": [
                {
                    "name": "url",
                    "description": "Target URL",
                    "optional": false
                },
                {
                    "name": "body",
                    "description": "Body match regular expression (body is
 limited to 100k)",
                    "optional": true
                },
                {
                    "name": "headers",
                    "description": "Arbitrary headers which are sent with the
 request.",
                    "optional": true
                },
                {
                    "name": "body_matches",
                    "description": "Body match regular expressions (body is
 limited to 100k)",
                    "optional": true
                },
                {
                    "name": "ssl",
```

```
                    "description": "Enable SSL",
                    "optional": true
                },
                {
                    "name": "method",
                    "description": "HTTP method (default: GET)",
                    "optional": true
                },
                {
                    "name": "auth_user",
                    "description": "Optional auth user",
                    "optional": true
                },
                {
                    "name": "auth_password",
                    "description": "Optional auth password",
                    "optional": true
                },
                {
                    "name": "extract",
                    "description": "Regex to capture key/value pairs.",
                    "optional": true
                },
                {
                    "name": "follow_redirects",
                    "description": "Follow redirects (default: true)",
                    "optional": true
                },
                {
                    "name": "payload",
                    "description": "Specify a request body (limited to 1024
characters). If following a redirect, payload will only be sent to first
location",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.tcp",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number",
                    "optional": false
                },
                {
                    "name": "banner_match",
                    "description": "Banner match regex.",
                    "optional": true
                },
                {
                    "name": "send_body",
                    "description": "Send a body. If a banner is provided the
body is sent after the banner is verified.",
                    "optional": true
                },
                {
                    "name": "body_match",
```

```
                    "description": "Body match regex. Key/Values are captured
when matches are specified within the regex. Note: Maximum body size 1024
bytes.",
                    "optional": true
                },
                {
                    "name": "ssl",
                    "description": "Enable SSL",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.ping",
            "type": "remote",
            "fields": [
                {
                    "name": "count",
                    "description": "Number of pings to send within a single
check",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.ftp-banner",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 21)",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.imap-banner",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 143)",
                    "optional": true
                },
                {
                    "name": "ssl",
                    "description": "Enable SSL",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.pop3-banner",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 110)",
                    "optional": true
                },
```

```
            {
                "name": "ssl",
                "description": "Enable SSL",
                "optional": true
            }
        ]
    },
    {
        "id": "remote.smtp-banner",
        "type": "remote",
        "fields": [
            {
                "name": "port",
                "description": "Port number (default: 25)",
                "optional": true
            },
            {
                "name": "ssl",
                "description": "Enable SSL",
                "optional": true
            }
        ]
    },
    {
        "id": "remote.postgresql-banner",
        "type": "remote",
        "fields": [
            {
                "name": "port",
                "description": "Port number (default: 5432)",
                "optional": true
            },
            {
                "name": "ssl",
                "description": "Enable SSL",
                "optional": true
            }
        ]
    },
    {
        "id": "remote.telnet-banner",
        "type": "remote",
        "fields": [
            {
                "name": "port",
                "description": "Port number (default: 23)",
                "optional": true
            },
            {
                "name": "banner_match",
                "description": "Banner to check",
                "optional": true
            },
            {
                "name": "ssl",
                "description": "Enable SSL",
                "optional": true
            }
        ]
    },
```

```
        {
            "id": "remote.mysql-banner",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 3306)",
                    "optional": true
                },
                {
                    "name": "ssl",
                    "description": "Enable SSL",
                    "optional": true
                }
            ]
        },
        {
            "id": "remote.mssql-banner",
            "type": "remote",
            "fields": [
                {
                    "name": "port",
                    "description": "Port number (default: 1433)",
                    "optional": true
                },
                {
                    "name": "ssl",
                    "description": "Enable SSL",
                    "optional": true
                }
            ]
        }
    ],
    "metadata": {
        "count": 14,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.5.6. Get Check Type

Verb	URI	Description
GET	/check_types/*checkTypeId*	Retrieve information for a single check type.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.37. Get Check Type: XML

```
<?xml version="1.0" encoding="utf-8"?>
<check_type id="remote.dns">
  <type>remote</type>
  <fields>
    <field>
      <name>port</name>
      <description>Port number (default: 53)</description>
      <optional>true</optional>
    </field>
    <field>
      <name>query</name>
      <description>DNS Query</description>
      <optional>false</optional>
    </field>
    <field>
      <name>record_type</name>
      <description>DNS Record Type</description>
      <optional>false</optional>
    </field>
  </fields>
</check_type>
```

Example 4.38. Get Check Type: JSON

```
{
    "id": "remote.dns",
    "type": "remote",
    "fields": [
        {
            "name": "port",
            "description": "Port number (default: 53)",
            "optional": true
        },
        {
            "name": "query",
            "description": "DNS Query",
            "optional": false
        },
        {
            "name": "record_type",
            "description": "DNS Record Type",
            "optional": false
        }
    ]
}
```

4.5.7. Update Check Type

Users cannot update check types.

4.5.8. Delete Check Types

Users cannot delete check types.

4.6. Alarms

4.6.1. Summary

Alarms bind alerting rules, entities, and notification plans into a logical unit. Alarms are responsible for examining the state of one or more checks and executing a notification plan. You create alerting rules by using the alarm DSL. For information about using the alarm language, refer to the reference documentation.

Note

Criteria is optional. This means you rely on the check successfully running or failing to set the state of the alarm. It is a convenient shortcut for setting a simple alarm with a notification plan. For example, if you set a ping check on a server, it won't alert until the pings actually fail to be returned, whereas adding criteria would enable the alert to trigger if the ping round trip time went past a certain threshold.

4.6.2. Attributes

Name	Description	Validation
notification_plan_id	The id of the notification plan to execute when the state changes.	• Valid Notification Plan • Non-empty string
check_id	The ID of the check to alert on. This paramater is mutually exclusive with check_type.	• Optional • Immutable • String
check_type	The type of check to alert on. This paramater is mutually exclusive with check_id.	• Optional • Immutable • String • One of (remote.dns, remote.ssh, remote.smtp, remote.http, remote.tcp, remote.ping, remote.ftp-banner, remote.imap-banner, remote.pop3-banner, remote.smtp-banner, remote.postgresql-banner, remote.telnet-banner, remote.mysql-banner, remote.mssql-banner)
criteria	The alarm DSL for describing alerting conditions and their output states.	• Optional • String between 1 and 16384 characters long • Valid Alarm
label	A friendly label for an alarm.	• Optional • String between 1 and 255 characters long
metadata	Arbitrary key/value pairs.	• Optional

Name	Description	Validation
		• Hash [String,String between 1 and 255 characters long:String,String between 1 and 255 characters long]

Note

check_id and check_type are mutually exclusive. Either check_id or check_type must be set but not both.

4.6.3. Create Alarm

Verb	URI	Description
POST	/entities/*entityId*/alarms	Creates a new alarm for the specified entity. Specify the alarm's characteristics using a valid set of parameters from the table shown in the Attributes section above.

Normal Response Code: (201) 'Location' header contains a link to the newly created alarm.

Error Response Codes: 400, 401, 403, 500, 503

Example 4.39. Alarm Create Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<alarm>
  <check_type>remote.http</check_type>
  <criteria>if (metric["t"] &gt;= 2.1) { return OK } return CRITICAL</
criteria>
  <notification_plan_id>npAAAAA</notification_plan_id>
</alarm>
```

Example 4.40. Alarm Create Request: JSON

```
{
    "check_type": "remote.http",
    "criteria": "if (metric[\"t\"] >= 2.1) { return OK } return CRITICAL",
    "notification_plan_id": "npAAAAA"
}
```

4.6.4. Test Alarm

Verb	URI	Description
POST	/entities/*entityId*/test-alarm	Test run an alarm.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 413, 500, 503

Example 4.41. Test Alarm Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<test_run_alarm>
  <criteria>if (metric["code"] == "404") { return CRITICAL, "not found" }
 return OK</criteria>
  <checks_data>
    <check_data>
      <timestamp>1319222001982</timestamp>
      <monitoring_zone_id>mzxJ4L2IU</monitoring_zone_id>
      <available>true</available>
      <status>code=200,rt=0.257s,bytes=0</status>
      <metrics>
        <bytes>
          <type>i</type>
          <data>0</data>
        </bytes>
        <tt_firstbyte>
          <type>I</type>
          <data>257</data>
        </tt_firstbyte>
        <tt_connect>
          <type>I</type>
          <data>128</data>
        </tt_connect>
        <code>
          <type>s</type>
          <data>200</data>
        </code>
        <duration>
          <type>I</type>
          <data>257</data>
        </duration>
      </metrics>
    </check_data>
  </checks_data>
</test_run_alarm>
```

Example 4.42. Test Alarm Request: JSON

```
{
    "criteria": "if (metric[\"code\"] == \"404\") { return CRITICAL, \"not
 found\" } return OK",
    "check_data": [
        {
            "timestamp": 1319222001982,
            "monitoring_zone_id": "mzxJ4L2IU",
            "available": true,
            "status": "code=200,rt=0.257s,bytes=0",
            "metrics": {
                "bytes": {
                    "type": "i",
                    "data": "0"
                },
                "tt_firstbyte": {
                    "type": "I",
                    "data": "257"
                },
                "tt_connect": {
                    "type": "I",
```

```
                    "data": "128"
                },
                "code": {
                    "type": "s",
                    "data": "200"
                },
                "duration": {
                    "type": "I",
                    "data": "257"
                }
            }
        }
    ]
}
```

Example 4.43. Test Alarm Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<alarm_data>
  <alarm_data>
    <timestamp>1319224500831</timestamp>
    <state>OK</state>
    <status>Matched default return statement</status>
  </alarm_data>
</alarm_data>
```

Example 4.44. Test Alarm Response: JSON

```
[
    {
        "timestamp": 1319224500831,
        "state": "OK",
        "status": "Matched default return statement"
    }
]
```

4.6.5. List Alarms

Verb	URI	Description
GET	/entities/*entityId*/alarms	List the alarms on the specified entity.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.45. List Alarms Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
```

```
    <alarm id="alAAAA">
      <check_type>remote.http</check_type>
      <criteria>if (metric["t"] &gt;= 2.1) { return OK } return CRITICAL</
criteria>
    </alarm>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.46. List Alarms Response: JSON

```
{
    "values": [
        {
            "id": "alAAAA",
            "check_type": "remote.http",
            "criteria": "if (metric[\"t\"] >= 2.1) { return OK } return
 CRITICAL"
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.6.6. Get Alarm

Verb	URI	Description
GET	/entities/*entityId*/alarms/*alarmId*	Get information for a single alarm.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.47. Get Alarm Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<alarm id="alAAAA">
  <check_type>remote.http</check_type>
  <criteria>if (metric["t"] &gt;= 2.1) { return OK } return CRITICAL</
criteria>
</alarm>
```

Example 4.48. Get Alarm Response: JSON

```
{
    "id": "alAAAA",
    "check_type": "remote.http",
    "criteria": "if (metric[\"t\"] >= 2.1) { return OK } return CRITICAL"
}
```

4.6.7. Update Alarm

Verb	URI	Description
PUT	/entities/*entityId*/alarms/*alarmId*	Update an alarm with the specified alarmId. Partial updates to an alarm are acceptable. You may specify only the parameters you would like to update.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Example 4.49. Update Alarm Request: XML

Normal Response Code: (204) This operation does not contain a response body.

Error Response Code: 500

```
<?xml version="1.0" encoding="utf-8"?>
<alarm>
  <criteria>if (metric["average"] &lt; 100) { return OK } return WARNING</
criteria>
</alarm>
```

Example 4.50. Alarm Update Request: JSON

```
{
    "criteria": "if (metric[\"average\"] < 100) { return OK } return WARNING"
}
```

4.6.8. Delete Alarm

Verb	URI	Description
DELETE	/entities/*entityId*/alarms/*alarmId*	Delete an alarm from your account.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 401, 403, 404, 500, 503

4.7. Notification Plans

4.7.1. Summary

A notification plan contains a set of notification actions that Rackspace Cloud Monitoring executes when triggered by an alarm. Rackspace Cloud Monitoring currently supports webhook and email notifications.

Each notification state can contain multiple notification actions. For example, you can create a notification plan that hits a webhook/email to notify your operations team if a warning occurs. However, if the warning escalates to an Error, the notification plan could be configured to hit a different webhook/email that triggers both email and SMS messages to the operations team. The notification plan supports the following states:

- Critical
- Warning
- OK

A notification plan, npTechnicalContactsEmail, is provided by default which will email all of the technical contacts on file for an account whenever there is a state change.

4.7.2. Attributes

Name	Description	Validation
label	Friendly name for the notification plan.	• String between 1 and 255 characters long
critical_state	The notification list to send to when the state is CRITICAL.	• Optional • Array [Valid Notification]
ok_state	The notification list to send to when the state is OK.	• Optional • Array [Valid Notification]
warning_state	The notification list to send to when the state is WARNING.	• Optional • Array [Valid Notification]

4.7.3. Create Notification Plan

Verb	URI	Description
POST	/notification_plans	Create a notification plan.

Normal Response Code: (201) 'Location' header contains a link to the newly created notification plan.

Error Response Codes: 400, 401, 403, 500, 503

Example 4.51. Notification Plan Create Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_plan>
  <label>Notification Plan 1</label>
  <critical_state>
```

```
    <notification_id>ntAAAA</notification_id>
  </critical_state>
  <warning_state>
    <notification_id>ntCCCCC</notification_id>
  </warning_state>
  <ok_state>
    <notification_id>ntBBBB</notification_id>
  </ok_state>
</notification_plan>
```

Example 4.52. Notification Plan Create Request: JSON

```
{
    "label": "Notification Plan 1",
    "critical_state": [
        "ntAAAA"
    ],
    "warning_state": [
        "ntCCCCC"
    ],
    "ok_state": [
        "ntBBBB"
    ]
}
```

4.7.4. List Notification Plans

Verb	URI	Description
GET	/notification_plans	Lists the notification plans for this particular account.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.53. List Notification Plans Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <notification_plan>
      <label>Notification Plan 1</label>
      <critical_state>
        <notification_id>ntAAAA</notification_id>
      </critical_state>
      <warning_state>
        <notification_id>ntCCCCC</notification_id>
      </warning_state>
      <ok_state>
        <notification_id>ntBBBB</notification_id>
      </ok_state>
    </notification_plan>
  </values>
```

```
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.54. List List Notification Plans Response: JSON

```
{
    "values": [
        {
            "label": "Notification Plan 1",
            "critical_state": [
                "ntAAAA"
            ],
            "warning_state": [
                "ntCCCCC"
            ],
            "ok_state": [
                "ntBBBB"
            ]
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.7.5. Get Notification Plan

Verb	URI	Description
GET	/notification_plans/*notificationPlanId*	Get information for a single notification plan.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.55. Get Notification Plan Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_plan>
  <label>Notification Plan 1</label>
  <critical_state>
    <notification_id>ntAAAA</notification_id>
  </critical_state>
  <warning_state>
```

```
    <notification_id>ntCCCCC</notification_id>
  </warning_state>
  <ok_state>
    <notification_id>ntBBBB</notification_id>
  </ok_state>
</notification_plan>
```

Example 4.56. Get List Notification Plan Response: JSON

```
{
    "label": "Notification Plan 1",
    "critical_state": [
        "ntAAAA"
    ],
    "warning_state": [
        "ntCCCCC"
    ],
    "ok_state": [
        "ntBBBB"
    ]
}
```

4.7.6. Update Notification Plans

Verb	URI	Description
PUT	/notification_plans/*notificationPlanId*	Update a notification plan with the specified notificationPlanId. Partial updates to a notification plan are acceptable. You may specify only the parameters you would like to update.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Example 4.57. Notification Plan Update Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_plan>
  <critical_state>
    <notification_id>ntBBBB</notification_id>
  </critical_state>
  <warning_state/>
</notification_plan>
```

Example 4.58. Notification Plan Update Request: JSON

```
{
    "critical_state": [
        "ntBBBB"
    ],
    "warning_state": []
}
```

4.7.7. Delete Notification Plans

Verb	URI	Description
DELETE	/notification_plans/*notificationPlanId*	Deletes a notification plan.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 401, 403, 404, 500, 503

4.8. Monitoring Zones

4.8.1. Summary

A monitoring zone is a location that Rackspace Cloud Monitoring collects data from. Examples of monitoring zones are "US West", "DFW1" or "ORD1". It is an abstraction for a general location from which data is collected.

An "endpoint," also known as a "collector," collects data from the monitoring zone. The endpoint is mapped directly to an individual machine or a virtual machine. A monitoring zone contains many endpoints, all of which will be within the IP address range listed in the response. The opposite is not true, however, as there may be unallocated IP addresses or unrelated machines within that IP address range.

A check references a list of monitoring zones it should be run from.

Note

Users cannot create, update or delete monitoring zones.

4.8.2. Attributes

Name	Description	Validation
country_code	Country Code	• String longer than 2 characters
label	Label	• String • Non-empty string
source_ips	Source IP list	• Array [String]

4.8.3. Create Monitoring Zone

Users cannot create monitoring zones.

4.8.4. List Monitoring Zones

Verb	URI	Description
GET	/monitoring_zones	List the monitoring zones.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.59. List Monitoring Zones Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <monitoring_zone id="mzAAAAA">
      <label>US South (Atlanta) - 5</label>
      <country_code>US</country_code>
      <source_ips>
        <ip>1.2.0.0/24</ip>
      </source_ips>
    </monitoring_zone>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.60. List Monitoring Zones Response: JSON

```
{
    "values": [
        {
            "id": "mzAAAAA",
            "label": "US South (Atlanta) - 5",
            "country_code": "US",
            "source_ips": [
                "1.2.0.0/24"
            ]
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.8.5. Get Monitoring Zone

Verb	URI	Description
GET	/monitoring_zones/*monitoringZoneId*	Get information for a single monitoring zone.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.61. Get Monitoring Zone Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<monitoring_zone id="mzAAAAA">
  <label>US South (Atlanta) - 5</label>
  <country_code>US</country_code>
  <source_ips>
    <ip>1.2.0.0/24</ip>
  </source_ips>
</monitoring_zone>
```

Example 4.62. Get Monitoring Zone Response: JSON

```
{
    "id": "mzAAAAA",
    "label": "US South (Atlanta) - 5",
    "country_code": "US",
    "source_ips": [
        "1.2.0.0/24"
    ]
}
```

4.8.6. Update Monitoring Zone

Users cannot update monitoring zones.

4.8.7. Delete Monitoring Zone

Users cannot delete monitoring zones.

4.8.8. Perform a "traceroute" from a Monitoring Zone

Verb	URI	Description
POST	/monitoring_zones/*monitoringZoneId*/traceroute	Perform a traceroute from a collector in the specified monitoring zones.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

This API lets you run a Traceroute from a Monitoring Zone to a Hostname or IP address. Like all Cloud Monitoring features, the Traceroute API is fully dual stack, supporting both IPv4 and IPv6. The Traceroute API can be used to debug networking issues between the Cloud Monitoring collectors and your infrastructure.

4.8.8.1. Attributes

Name	Description	Validation
target	Traceroute target IP or hostname.	• Valid hostname, IPv4 or IPv6 address
target_resolver	Target resolver type. Only applicable if hostname is provided.	• Optional • One of (IPv4, IPv6)

Example 4.63. Traceroute Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<traceroute>
  <target>google.com</target>
  <target_resolver>IPv4</target_resolver>
</traceroute>
```

Example 4.64. Traceroute Request: JSON

```
{
    "target": "google.com",
    "target_resolver": "IPv4"
}
```

Example 4.65. Traceroute Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<traceroute>
  <hops>
    <hop>
      <number>1</number>
      <rtts>3.025</rtts>
      <rtts>3.116</rtts>
      <rtts>3.189</rtts>
      <ip>50.57.208.106</ip>
    </hop>
    <hop>
      <number>2</number>
      <rtts>0.675</rtts>
      <rtts>0.943</rtts>
      <rtts>1.083</rtts>
      <ip>50.56.6.34</ip>
    </hop>
    <hop>
      <number>3</number>
      <rtts>0.558</rtts>
      <rtts>0.824</rtts>
      <ip>50.56.6.16</ip>
    </hop>
    <hop>
      <number>3</number>
      <rtts>0.606</rtts>
      <ip>50.56.6.18</ip>
    </hop>
    <hop>
      <number>4</number>
      <rtts>0.239</rtts>
      <rtts>0.252</rtts>
      <rtts>0.247</rtts>
      <ip>184.106.126.139</ip>
    </hop>
    <hop>
      <number>5</number>
      <rtts>3.418</rtts>
```

```
      <rtts>3.444</rtts>
      <rtts>3.453</rtts>
      <ip>69.31.110.241</ip>
    </hop>
    <hop>
      <number>6</number>
      <rtts>1.219</rtts>
      <rtts>1.231</rtts>
      <ip>69.31.110.249</ip>
    </hop>
    <hop>
      <number>6</number>
      <rtts>1.3</rtts>
      <ip>69.31.110.253</ip>
    </hop>
    <hop>
      <number>7</number>
      <rtts>2.463</rtts>
      <rtts>2.224</rtts>
      <rtts>2.196</rtts>
      <ip>206.223.119.21</ip>
    </hop>
    <hop>
      <number>8</number>
      <rtts>1.831</rtts>
      <rtts>2.041</rtts>
      <rtts>1.814</rtts>
      <ip>209.85.254.130</ip>
    </hop>
    <hop>
      <number>9</number>
      <rtts>2.558</rtts>
      <rtts>1.977</rtts>
      <rtts>3.105</rtts>
      <ip>72.14.237.133</ip>
    </hop>
    <hop>
      <number>10</number>
      <rtts>51.028</rtts>
      <ip>216.239.46.214</ip>
    </hop>
    <hop>
      <number>10</number>
      <rtts>29.526</rtts>
      <ip>216.239.46.216</ip>
    </hop>
    <hop>
      <number>10</number>
      <rtts>48.987</rtts>
      <ip>216.239.46.214</ip>
    </hop>
    <hop>
      <number>11</number>
      <rtts>105.6</rtts>
      <ip>216.239.46.219</ip>
    </hop>
    <hop>
      <number>11</number>
      <rtts>128.521</rtts>
      <ip>216.239.43.5</ip>
```

```
    </hop>
    <hop>
      <number>11</number>
      <rtts>105.548</rtts>
      <ip>216.239.46.219</ip>
    </hop>
    <hop>
      <number>12</number>
      <rtts>109.492</rtts>
      <ip>72.14.235.175</ip>
    </hop>
    <hop>
      <number>12</number>
      <rtts>106.523</rtts>
      <ip>72.14.235.173</ip>
    </hop>
    <hop>
      <number>12</number>
      <rtts>105.952</rtts>
      <ip>72.14.235.175</ip>
    </hop>
    <hop>
      <number>13</number>
      <rtts>106.482</rtts>
      <ip>216.239.43.233</ip>
    </hop>
    <hop>
      <number>13</number>
      <rtts>106.92</rtts>
      <rtts>106.681</rtts>
      <ip>209.85.253.20</ip>
    </hop>
    <hop>
      <number>14</number>
      <rtts>129.616</rtts>
      <ip>72.14.236.191</ip>
    </hop>
    <hop>
      <number>14</number>
      <rtts>106.329</rtts>
      <ip>209.85.252.83</ip>
    </hop>
    <hop>
      <number>14</number>
      <rtts>106.253</rtts>
      <ip>216.239.49.45</ip>
    </hop>
    <hop>
      <number>15</number>
      <ip>*</ip>
    </hop>
    <hop>
      <number>16</number>
      <rtts>106.101</rtts>
      <rtts>106.075</rtts>
      <rtts>106.48</rtts>
      <ip>173.194.78.139</ip>
    </hop>
  </hops>
</traceroute>
```

Example 4.66. Traceroute Response: JSON


```
{
    "result": [
        {
            "number": 1,
            "rtts": [
                3.025,
                3.116,
                3.189
            ],
            "ip": "50.57.208.106"
        },
        {
            "number": 2,
            "rtts": [
                0.675,
                0.943,
                1.083
            ],
            "ip": "50.56.6.34"
        },
        {
            "number": 3,
            "rtts": [
                0.558,
                0.824
            ],
            "ip": "50.56.6.16"
        },
        {
            "number": 3,
            "rtts": [
                0.606
            ],
            "ip": "50.56.6.18"
        },
        {
            "number": 4,
            "rtts": [
                0.239,
                0.252,
                0.247
            ],
            "ip": "184.106.126.139"
        },
        {
            "number": 5,
            "rtts": [
                3.418,
                3.444,
                3.453
            ],
            "ip": "69.31.110.241"
        },
        {
            "number": 6,
```

```
            "rtts": [
                1.219,
                1.231
            ],
            "ip": "69.31.110.249"
        },
        {
            "number": 6,
            "rtts": [
                1.3
            ],
            "ip": "69.31.110.253"
        },
        {
            "number": 7,
            "rtts": [
                2.463,
                2.224,
                2.196
            ],
            "ip": "206.223.119.21"
        },
        {
            "number": 8,
            "rtts": [
                1.831,
                2.041,
                1.814
            ],
            "ip": "209.85.254.130"
        },
        {
            "number": 9,
            "rtts": [
                2.558,
                1.977,
                3.105
            ],
            "ip": "72.14.237.133"
        },
        {
            "number": 10,
            "rtts": [
                51.028
            ],
            "ip": "216.239.46.214"
        },
        {
            "number": 10,
            "rtts": [
                29.526
            ],
            "ip": "216.239.46.216"
        },
        {
            "number": 10,
            "rtts": [
                48.987
            ],
            "ip": "216.239.46.214"
```

```
        },
        {
            "number": 11,
            "rtts": [
                105.6
            ],
            "ip": "216.239.46.219"
        },
        {
            "number": 11,
            "rtts": [
                128.521
            ],
            "ip": "216.239.43.5"
        },
        {
            "number": 11,
            "rtts": [
                105.548
            ],
            "ip": "216.239.46.219"
        },
        {
            "number": 12,
            "rtts": [
                109.492
            ],
            "ip": "72.14.235.175"
        },
        {
            "number": 12,
            "rtts": [
                106.523
            ],
            "ip": "72.14.235.173"
        },
        {
            "number": 12,
            "rtts": [
                105.952
            ],
            "ip": "72.14.235.175"
        },
        {
            "number": 13,
            "rtts": [
                106.482
            ],
            "ip": "216.239.43.233"
        },
        {
            "number": 13,
            "rtts": [
                106.92,
                106.681
            ],
            "ip": "209.85.253.20"
        },
        {
            "number": 14,
```

```
            "rtts": [
                129.616
            ],
            "ip": "72.14.236.191"
        },
        {
            "number": 14,
            "rtts": [
                106.329
            ],
            "ip": "209.85.252.83"
        },
        {
            "number": 14,
            "rtts": [
                106.253
            ],
            "ip": "216.239.49.45"
        },
        {
            "number": 15,
            "rtts": [],
            "ip": "*"
        },
        {
            "number": 16,
            "rtts": [
                106.101,
                106.075,
                106.48
            ],
            "ip": "173.194.78.139"
        }
    ]
}
```

4.9. Alarm Notification History

4.9.1. Summary

The monitoring service keeps a record of notifications sent for each alarm. This history is further subdivided by the check on which the notification occurred. Every attempt to send a notification is recorded, making this history a valuable tool in diagnosing issues with unreceived notifications, in addition to offering a means of viewing the history of an alarm's statuses.

Alarm notification history is accessible as a Time Series Collection. By default alarm notification history is stored for 30 days and the the API queries the last 7 days of information.

4.9.2. Discover Alarm Notification History

Verb	URI	Description
GET	/entities/*entityId*/alarms/*alarmId*/ notification_history	List checks for which alarm notification history is available.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.67. Alarm Notification History Discovery Response: XML

```
<?xml version='1.0' encoding='utf-8'?>
<alarm_checks_map>
  <check_ids>
    <check_id>chOne</check_id>
    <check_id>chTwo</check_id>
  </check_ids>
</alarm_checks_map>
```

Example 4.68. Alarm Notification History Discovery Response: JSON

```
{
    "check_ids": [
        "chOne",
        "chTwo"
    ]
}
```

4.9.3. List Alarm Notification History

Verb	URI	Description
GET	/entities/*entityId*/alarms/*alarmId*/notification_history/*checkId*	Lists alarm notification history for a given entity, alarm and check.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.69. Alarm Notification History List Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <alarm_notification_history_item>
      <timestamp>1320885544875</timestamp>
      <notification_plan_id>npOne</notification_plan_id>
      <transaction_id>sometransaction</transaction_id>
      <status>matched return statement on line 6</status>
      <state>WARNING</state>
      <previous_state>OK</previous_state>
      <notification_results>
        <notification_result>
          <notification_id>ntOne</notification_id>
          <notification_type>webhook</notification_type>
          <notification_details>
            <url>http://admin.example.com/notify/</url>
          </notification_details>
          <in_progress>false</in_progress>
          <message>Success: Webhook successfully executed</message>
```

```
          <success>true</success>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>Unexpected status code "404". Expected one of: /2[0-9]
{2}/</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>Success: Webhook successfully executed</message>
            <from>ele</from>
          </attempts>
        </notification_result>
        <notification_result>
          <notification_id>ntFive</notification_id>
          <notification_type>webhook</notification_type>
          <notification_details>
            <url>https://down.example.com/notify/</url>
          </notification_details>
          <in_progress>false</in_progress>
          <success>false</success>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <attempts>
```

```
            <message>ECONNREFUSED, Connection refused</message>
            <from>ele</from>
          </attempts>
          <message>Notification failed after 10 attempts</message>
        </notification_result>
      </notification_results>
    </alarm_notification_history_item>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.70. Alarm Notification History List Response: JSON

```
{
    "values": [
        {
            "timestamp": 1320885544875,
            "notification_plan_id": "npOne",
            "transaction_id": "sometransaction",
            "status": "matched return statement on line 6",
            "state": "WARNING",
            "previous_state": "OK",
            "notification_results": [
                {
                    "notification_id": "ntOne",
                    "notification_type": "webhook",
                    "notification_details": {
                        "url": "http://admin.example.com/notify/"
                    },
                    "in_progress": false,
                    "message": "Success: Webhook successfully executed",
                    "success": true,
                    "attempts": [
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "Unexpected status code \"404\".
 Expected one of: /2[0-9]{2}/",
                            "from": "ele"
                        },
                        {
                            "message": "Success: Webhook successfully
 executed",
                            "from": "ele"
                        }
                    ]
                },
                {
                    "notification_id": "ntFive",
                    "notification_type": "webhook",
```

```
                    "notification_details": {
                        "url": "https://down.example.com/notify/"
                    },
                    "in_progress": false,
                    "success": false,
                    "attempts": [
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        },
                        {
                            "message": "ECONNREFUSED, Connection refused",
                            "from": "ele"
                        }
                    ],
                    "message": "Notification failed after 10 attempts"
                }
            ]
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.9.4. Get Alarm Notification History

Verb	URI	Description
GET	/entities/*entityId*/alarms/*alarmId*/notification_history/*checkId*/*uuid*	Retrieve a single alarm notification history item.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.71. Get Alarm Notification History Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<alarm_notification_history_item>
  <timestamp>1320885544875</timestamp>
  <notification_plan_id>npOne</notification_plan_id>
  <transaction_id>sometransaction</transaction_id>
  <status>matched return statement on line 6</status>
  <state>WARNING</state>
  <previous_state>OK</previous_state>
  <notification_results>
    <notification_result>
      <notification_id>ntOne</notification_id>
      <notification_type>webhook</notification_type>
      <notification_details>
        <url>http://admin.example.com/notify/</url>
      </notification_details>
      <in_progress>false</in_progress>
      <message>Success: Webhook successfully executed</message>
      <success>true</success>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>Unexpected status code "404". Expected one of: /2[0-9]{2}/</
message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>Success: Webhook successfully executed</message>
        <from>ele</from>
      </attempts>
    </notification_result>
    <notification_result>
      <notification_id>ntFive</notification_id>
      <notification_type>webhook</notification_type>
      <notification_details>
        <url>https://down.example.com/notify/</url>
      </notification_details>
      <in_progress>false</in_progress>
      <success>false</success>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
```

```
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <attempts>
        <message>ECONNREFUSED, Connection refused</message>
        <from>ele</from>
      </attempts>
      <message>Notification failed after 10 attempts</message>
    </notification_result>
  </notification_results>
</alarm_notification_history_item>
```

Example 4.72. Get Alarm Notification History Response: JSON

```
{
    "timestamp": 1320885544875,
    "notification_plan_id": "npOne",
    "transaction_id": "sometransaction",
    "status": "matched return statement on line 6",
    "state": "WARNING",
    "previous_state": "OK",
    "notification_results": [
        {
            "notification_id": "ntOne",
            "notification_type": "webhook",
            "notification_details": {
                "url": "http://admin.example.com/notify/"
            },
            "in_progress": false,
```

```
            "message": "Success: Webhook successfully executed",
            "success": true,
            "attempts": [
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "Unexpected status code \"404\". Expected one
of: /2[0-9]{2}/",
                    "from": "ele"
                },
                {
                    "message": "Success: Webhook successfully executed",
                    "from": "ele"
                }
            ]
        },
        {
            "notification_id": "ntFive",
            "notification_type": "webhook",
            "notification_details": {
                "url": "https://down.example.com/notify/"
            },
            "in_progress": false,
            "success": false,
            "attempts": [
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
```

```
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                },
                {
                    "message": "ECONNREFUSED, Connection refused",
                    "from": "ele"
                }
            ],
            "message": "Notification failed after 10 attempts"
        }
    ]
}
```

4.10. Notifications

4.10.1. Summary

For instance, with a webhook type notification Rackspace Cloud Monitoring posts JSON formatted data to a user-specified URL on an alert condition (Check goes from OK -> CRITICAL and so on).

4.10.2. Attributes

Name	Description	Validation
details	A hash of notification specific details based on the notification type.	• Hash [String,String between 1 and 255 characters long:Non-empty string,String between 1 and 4096 characters long]
label	Friendly name for the notification.	• String between 1 and 255 characters long
type	The notification type to send.	• String • One of (webhook, email)

4.10.3. Create Notification

Verb	URI	Description
POST	/notifications	Check that a token is valid and that it belongs to a particular user and return the permissions relevant to a particular client.

Normal Response Code: (201) 'Location' header contains a link to the newly created notification.

Error Response Codes: 400, 401, 403, 500, 503

Example 4.73. Create Notification Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification>
  <label>my webhook #1</label>
  <type>webhook</type>
  <details>
    <url>https://systems.example.org/alert</url>
  </details>
</notification>
```

Example 4.74. Create Notification Request: JSON

```
{
    "label": "my webhook #1",
    "type": "webhook",
    "details": {
        "url": "https://systems.example.org/alert"
    }
}
```

4.10.4. Test Notification

Verb	URI	Description
POST	/test-notification	Test a notification.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 500, 503

This operation allows you to test a notification before creating it. The actual notification comes from the same server where the actual alert messages come from. This allow you to, among other things, verify that your firewall is configured properly.

Example 4.75. Test Notification Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<test_notification>
  <type>webhook</type>
  <details>
    <url>http://my.web-server.com:5981/</url>
  </details>
</test_notification>
```

Example 4.76. Test Notification Request: JSON

```
{
    "type": "webhook",
    "details": {
        "url": "http://my.web-server.com:5981/"
    }
}
```

Example 4.77. Test Notification Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_data>
  <status>success</status>
  <message>Success: Webhook successfully executed</message>
</notification_data>
```

Example 4.78. Test Notification Response: JSON

```
{
    "status": "success",
    "message": "Success: Webhook successfully executed"
}
```

4.10.5. Test Existing Notification

Verb	URI	Description
POST	/notifications/*notificationId*/test	Test an existing notification.

Normal Response Code: 200

Error Response Codes: 400, 401, 403, 500, 503

This operation allows you to test an existing notification. The actual notification comes from the same server where the actual alert messages come from. This allow you to, among other things, verify that your firewall is configured properly.

Example 4.79. Test Existing Notification Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_data>
  <status>success</status>
  <message>Success: Webhook successfully executed</message>
</notification_data>
```

Example 4.80. Test Existing Notification Response: JSON

```
{
    "status": "success",
    "message": "Success: Webhook successfully executed"
}
```

4.10.6. List Notifications

Verb	URI	Description
GET	/notifications	Lists the notifications for this particular account.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.81. List Notifications Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <notification id="ntAAAA">
      <label>my webhook #1</label>
      <type>webhook</type>
      <details>
        <url>https://systems.example.org/alert</url>
      </details>
    </notification>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.82. List Notifications Response: JSON

```
{
    "values": [
        {
            "id": "ntAAAA",
            "label": "my webhook #1",
            "type": "webhook",
            "details": {
                "url": "https://systems.example.org/alert"
            }
        }
```

```
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.10.7. Get Notifications

Verb	URI	Description
GET	/notifications/*notificationId*	Get information for a single notification.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.83. Get Notification Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification id="ntAAAA">
  <label>my webhook #1</label>
  <type>webhook</type>
  <details>
    <url>https://systems.example.org/alert</url>
  </details>
</notification>
```

Example 4.84. Get Notification Response: JSON

```
{
    "id": "ntAAAA",
    "label": "my webhook #1",
    "type": "webhook",
    "details": {
        "url": "https://systems.example.org/alert"
    }
}
```

4.10.8. Update Notifications

Verb	URI	Description
PUT	/notifications/*notificationId*	Updates a notification with the specified notificationId.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 400, 401, 403, 404, 500, 503

Example 4.85. Notification Update Request: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification>
  <type>webhook</type>
  <details>
    <url>https://systems.example.org/new_alert</url>
  </details>
</notification>
```

Example 4.86. Notification Update Request: JSON

```
{
    "type": "webhook",
    "details": {
        "url": "https://systems.example.org/new_alert"
    }
}
```

4.10.9. Delete Notifications

Verb	URI	Description
DELETE	/notifications/*notificationId*	Deletes a notification from your account.

Normal Response Code: (204) This code contains no content with an empty response body.

Error Response Codes: 401, 403, 404, 500, 503

4.11. Notification Types

The notification type represents the type of notification. When you create a notification in Rackspace Cloud Monitoring, you must specify the notification type. Rackspace Cloud Monitoring currently supports the following notification types:

Webhook Industry-standard web hooks, where JSON is posted to a configurable URL.

Email Email alerts where the message is delivered to a specified address.

4.11.1. Email Notifications

A notification sent by email.

4.11.1.1. Attributes

Name	Description	Validation
address	Email address to send notifications to	• Email address

4.11.2. Webhook Notifications

The webhook notification takes the following parameters:

4.11.2.1. Attributes

Name	Description	Validation
url	An HTTP or HTTPS URL to POST to	• URL

Example 4.87. Webhook Notification POST to a Specified URL with JSON Payload

```
{
    "event_id": "acOne:enOne:alOne:chOne:1326910500000:WARNING",
    "log_entry_id": "6da55310-4200-11e1-aaaf-cd4c8801b6b1",
    "details": {
        "target": null,
        "timestamp": 1326905540481,
        "metrics": {
            "tt_firstbyte": {
                "type": "I",
                "data": 2
            },
            "duration": {
                "type": "I",
                "data": 2
            },
            "bytes": {
                "type": "i",
                "data": 17
            },
            "tt_connect": {
                "type": "I",
                "data": 0
            },
            "code": {
                "type": "s",
                "data": "200"
            }
        },
        "state": "WARNING",
        "status": "warn.",
        "txn_id": "sometransaction",
        "collector_address_v4": "127.0.0.1",
        "collector_address_v6": null
    },
    "entity": {
        "id": "enOne",
        "label": "entity one",
        "ip_addresses": {
            "default": "127.0.0.1"
        },
        "metadata": null,
        "managed": false,
        "uri": null,
        "agent_id": null,
        "created_at": 1326905540481,
        "updated_at": 1326905540481
    },
    "check": {
        "id": "chOne",
        "label": "ch a",
```

```
        "type": "remote.http",
        "details": {
            "url": "http://www.foo.com",
            "body": "b",
            "method": "GET"
        },
        "monitoring_zones_poll": [
            "mzOne"
        ],
        "timeout": 60,
        "period": 150,
        "target_alias": "default",
        "target_hostname": "",
        "target_resolver": "",
        "disabled": false,
        "metadata": null,
        "created_at": 1326905540481,
        "updated_at": 1326905540481
    },
    "alarm": {
        "id": "alOne",
        "label": "Alarm 1",
        "check_type": "remote.http",
        "check_id": null,
        "criteria": "if (metric[\"t\"] >= 2.1) { return WARNING } return
 WARNING",
        "notification_plan_id": "npOne",
        "metadata": null,
        "created_at": 1326905540481,
        "updated_at": 1326905540481
    },
    "tenant_id": "91111"
}
```

4.11.2.2. Notification Payload

The following fields are contained within a single payload:

Field	Description
eventId	The ID for the event in the system.
logEntryId	The ID for the log entry.
entity	The entity record that triggered the alert.
check	The check record that triggered the alert.
alarm	The alarm record that triggered the alert.

4.11.2.3. Notification Header Contents

The following fields will be populated in the request header when your webhook is called.

Header Field	Description
x-rackspace-webhook-token	The webhook token defined in an account. This is used in your web application to verify that your webhook is called by an authorized Rackspace service.

4.11.3. Create Notification Type

Users cannot create notification types.

4.11.4. List Notification Types

Verb	URI	Description
GET	/notification_types	Lists available notification types.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.88. List Notification Types Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <notification_type id="webhook">
      <fields>
        <field>
          <name>url</name>
          <description>An HTTP or HTTPS URL to POST to</description>
          <optional>false</optional>
        </field>
      </fields>
    </notification_type>
    <notification_type id="email">
      <fields>
        <field>
          <name>address</name>
          <description>Email address to send notifications to</description>
          <optional>false</optional>
        </field>
      </fields>
    </notification_type>
  </values>
  <metadata>
    <count>2</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.89. List Notification Types Response: JSON

```
{
    "values": [
```

```
        {
            "id": "webhook",
            "fields": [
                {
                    "name": "url",
                    "description": "An HTTP or HTTPS URL to POST to",
                    "optional": false
                }
            ]
        },
        {
            "id": "email",
            "fields": [
                {
                    "name": "address",
                    "description": "Email address to send notifications to",
                    "optional": false
                }
            ]
        }
    ],
    "metadata": {
        "count": 2,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.11.5. Get Notification Type

Verb	URI	Description
GET	/notification_types/*notificationTypeId*	Get information for a single notification type.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.90. Get Notification Type Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<notification_type id="webhook">
  <fields>
    <field>
      <name>url</name>
      <description>An HTTP or HTTPS URL to POST to</description>
      <optional>false</optional>
    </field>
  </fields>
</notification_type>
```

Example 4.91. Get Notification Type Response: JSON

```
{
    "id": "webhook",
    "fields": [
        {
            "name": "url",
            "description": "An HTTP or HTTPS URL to POST to",
            "optional": false
        }
    ]
}
```

4.11.6. Update Notification Type

Users cannot update notification types.

4.11.7. Delete Notification Type

Users cannot delete notification types.

4.12. Changelogs

4.12.1. Summary

The monitoring service records changelogs for alarm statuses. Changelogs are accessible as a Time Series Collection. By default the API queries the last 7 days of changelog information.

4.12.2. List Alarm Changelogs

Alarm changelogs store the last 30 days of alarm state changes for all alarms on an account.

Verb	URI	Description
GET	/changelogs/alarms	Lists alarm changelogs for this account.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

Example 4.92. Alarm Changelog List Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <alarm_changelog>
      <timestamp>1320890228991</timestamp>
      <entity_id>enPhid7noo</entity_id>
      <alarm_id>alahf9vuNa</alarm_id>
      <check_id>chIe7vohba</check_id>
```

```
      <state>WARNING</state>
      <analyzed_by_monitoring_zone_id>DFW</analyzed_by_monitoring_zone_id>
    </alarm_changelog>
  </values>
  <metadata>
    <count>1</count>
    <limit>50</limit>
    <marker/>
    <next_marker/>
    <next_href/>
  </metadata>
</container>
```

Example 4.93. Alarm Changelog List Response: JSON

```
{
    "values": [
        {
            "timestamp": 1320890228991,
            "entity_id": "enPhid7noo",
            "alarm_id": "alahf9vuNa",
            "check_id": "chIe7vohba",
            "state": "WARNING",
            "analyzed_by_monitoring_zone_id": "DFW"
        }
    ],
    "metadata": {
        "count": 1,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

4.13. Views

Views contain a combination of data that usually includes multiple, different objects. The primary purpose of a view is to save API calls and make data retrieval more efficient. Instead of doing multiple API calls and then combining the result yourself, you can perform a single API call against the view endpoint.

4.13.1. Get Overview

Verb	URI	Description
GET	/views/overview	Return the overview view for this account.

There are no required parameters for this call. You may filter returned entities by ID or URI with the optional query parameters URI and ID. Up to 100 entities may be selected individually per request.

Normal Response Code: 200

Error Response Codes: 400,401, 403, 404, 500, 503

This view includes a list of entities on your account and each entity's child check and alarm objects. Along with the child check and alarm objects it also includes the latest computed state for each check and alarm pair. If there is no latest state available for a check and alarm pair, it means the alarm hasn't been evaluated yet and the current state for this pair is 'UNKNOWN'.

Example 4.94. Get Overview Response: JSON

```
{
    "values": [
        {
            "entity": {
                "id": "enBBBBIPV4",
                "label": "entity b v4",
                "ip_addresses": {
                    "default": "127.0.0.1"
                },
                "metadata": null
            },
            "checks": [
                {
                    "id": "chFour",
                    "label": "ch a",
                    "type": "remote.http",
                    "details": {
                        "url": "http://www.foo.com",
                        "method": "GET"
                    },
                    "monitoring_zones_poll": [
                        "mzA"
                    ],
                    "timeout": 60,
                    "period": 150,
                    "target_alias": "default",
                    "target_hostname": "",
                    "target_resolver": "",
                    "disabled": false
                }
            ],
            "alarms": [
                {
                    "id": "alThree",
                    "check_type": "remote.http",
                    "check_id": null,
                    "criteria": "if (metric[\"size\"] >= 200) { return
 CRITICAL } return OK",
                    "notification_plan_id": "npOne"
                }
            ],
            "latest_alarm_states": [
                {
                    "timestamp": 1321898988,
                    "entity_id": "enBBBBIPV4",
                    "alarm_id": "alThree",
                    "check_id": "chFour",
                    "status": "everything is ok",
                    "state": "OK",
                    "previous_state": "WARNING",
```

```
                "analyzed_by_monitoring_zone_id": null
            }
        ]
    },
    {
        "entity": {
            "id": "enCCCCIPV4",
            "label": "entity c v4",
            "ip_addresses": {
                "default": "127.0.0.1"
            },
            "metadata": null
        },
        "checks": [],
        "alarms": [],
        "latest_alarm_states": []
    },
    {
        "entity": {
            "id": "enAAAAIPV4",
            "label": "entity a",
            "ip_addresses": {
                "default": "127.0.0.1"
            },
            "metadata": null
        },
        "checks": [
            {
                "id": "chOne",
                "label": "ch a",
                "type": "remote.http",
                "details": {
                    "url": "http://www.foo.com",
                    "method": "GET"
                },
                "monitoring_zones_poll": [
                    "mzA"
                ],
                "timeout": 60,
                "period": 150,
                "target_alias": "default",
                "target_hostname": "",
                "target_resolver": "",
                "disabled": false
            },
            {
                "id": "chThree",
                "label": "ch a",
                "type": "remote.http",
                "details": {
                    "url": "http://www.foo.com",
                    "method": "GET"
                },
                "monitoring_zones_poll": [
                    "mzA"
                ],
                "timeout": 60,
                "period": 150,
                "target_alias": "default",
                "target_hostname": "",
```

```
                    "target_resolver": "",
                    "disabled": false
                },
                {
                    "id": "chTwo",
                    "label": "ch a",
                    "type": "remote.http",
                    "details": {
                        "url": "http://www.foo.com",
                        "method": "GET"
                    },
                    "monitoring_zones_poll": [
                        "mzA"
                    ],
                    "timeout": 60,
                    "period": 150,
                    "target_alias": "default",
                    "target_hostname": "",
                    "target_resolver": "",
                    "disabled": false
                }
            ],
            "alarms": [
                {
                    "id": "alOne",
                    "label": "Alarm 1",
                    "check_type": "remote.http",
                    "check_id": null,
                    "criteria": "if (metric[\"t\"] >= 2.1) { return CRITICAL }
return OK",
                    "notification_plan_id": "npOne"
                },
                {
                    "id": "alTwo",
                    "label": "Alarm 2",
                    "check_type": "remote.http",
                    "check_id": null,
                    "criteria": "if (metric[\"size\"] >= 200) { return
CRITICAL } return OK",
                    "notification_plan_id": "npOne"
                }
            ],
            "latest_alarm_states": [
                {
                    "timestamp": 1321898988,
                    "entity_id": "enAAAAIPV4",
                    "alarm_id": "alOne",
                    "check_id": "chOne",
                    "status": "matched return statement on line 7",
                    "state": "WARNING",
                    "previous_state": "OK",
                    "analyzed_by_monitoring_zone_id": null
                },
                {
                    "timestamp": 1321898988,
                    "entity_id": "enAAAAIPV4",
                    "alarm_id": "alOne",
                    "check_id": "chTwo",
                    "state": "CRITICAL",
                    "analyzed_by_monitoring_zone_id": null
```

```
                }
            ]
        }
    ],
    "metadata": {
        "count": 3,
        "limit": 50,
        "marker": null,
        "next_marker": null,
        "next_href": null
    }
}
```

Note

You may filter returned entities by URI or ID, but not both in the same request. When filtering by Entity ID, if any supplied Entity IDs are unknown or incorrect, the response will be a HTTP 404. When filtering by Entity URI, unknown URIs will be ignored, and entities will be returned for any correct URIs. A request filtering by Entity URI will only receive a HTTP 404 if no known URIs are supplied.

4.14. Alarm Examples

View provides examples alarms for the various checks in the system. They are presented as a template with parameters. Each of the parameters is documented with a type, name and description. There are quite a few different examples in the system.

4.14.1. List Alarm Examples

Verb	URI	Description
GET	/alarm_examples	Return a list of alarm examples.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

This list contains a complete picture of all the example alarms. There is extra information provided in the api such as the list of fields. Within the field is also a small description and type.

Example 4.95. List Alarm Examples Response: JSON

```
{
    "values": [
        {
            "id": "remote.http_body_match_1",
            "label": "Body match",
            "description": "Alarm which returns CRITICAL if the provided
 string is found in the body",
            "check_type": "remote.http",
```

```
            "criteria": "if (metric['body_match'] regex \"${string}\") {\n
return CRITICAL, \"${string} found, returning CRITICAL.\"\n}\n",
            "fields": [
                {
                    "name": "string",
                    "description": "String to check for in the body",
                    "type": "string"
                }
            ]
        },
        {
            "id": "remote.http_body_match_missing_string",
            "label": "Body match",
            "description": "Alarm which returns CRITICAL if the provided
string is not found in the body",
            "check_type": "remote.http",
            "criteria": "if (metric['body_match'] == \"\") {\n  return
CRITICAL, \"HTTP response did not contain the correct content.\"\n}\n\nreturn
OK, \"HTTP response contains the correct content\"\n",
            "fields": []
        },
        {
            "id": "remote.http_connection_time",
            "label": "Connection time",
            "description": "Alarm which returns WARNING or CRITICAL based on
the connection time",
            "check_type": "remote.http",
            "criteria": "if (metric['duration'] > ${critical_threshold})
{\n  return CRITICAL, \"HTTP request took more than ${critical_threshold}
milliseconds.\"\n}\n\nif (metric['duration'] > ${warning_threshold})
{\n  return WARNING, \"HTTP request took more than ${warning_threshold}
milliseconds.\"\n}\n\nreturn OK, \"HTTP connection time is normal\"\n",
            "fields": [
                {
                    "name": "warning_threshold",
                    "description": "Warning threshold (in milliseconds) for
the connection time",
                    "type": "integer"
                },
                {
                    "name": "critical_threshold",
                    "description": "Critical threshold (in milliseconds) for
the connection time",
                    "type": "integer"
                }
            ]
        },
        {
            "id": "remote.http_status_code",
            "label": "Status code",
            "description": "Alarm which returns WARNING if the server
responses with 4xx status code or CRITICAL if it responds with 5xx status
code",
            "check_type": "remote.http",
            "criteria": "if (metric['code'] regex \"4[0-9][0-9]\") {\n  return
CRITICAL, \"HTTP server responding with 4xx status\"\n}\n\nif (metric['code']
regex \"5[0-9][0-9]\") {\n  return CRITICAL, \"HTTP server responding with
5xx status\"\n}\n\nreturn OK, \"HTTP server is functioning normally\"\n",
            "fields": []
        },
```

```
       {
           "id": "remote.http_cert_expiration",
           "label": "SSL certificate expiration time",
           "description": "Alarm which returns WARNING or CRITICAL based on
 the certificate expiration date",
           "check_type": "remote.http",
           "criteria": "if (metric['cert_end_in'] < ${critical_threshold})
 {\n  return CRITICAL, \"Cert expiring in less than ${critical_threshold}
 seconds.\"\n}\n\nif (metric['cert_end_in'] < ${warning_threshold}) {\n
 return WARNING, \"Cert expiring in less than ${warning_threshold} seconds.\"\
n}\n\nreturn OK, \"HTTP certificate doesn't expire soon.\"\n",
           "fields": [
               {
                   "name": "warning_threshold",
                   "description": "Warning threshold (in seconds) for the
 certificate expiration",
                   "type": "integer"
               },
               {
                   "name": "critical_threshold",
                   "description": "Critical threshold (in seconds) for the
 certificate expiration",
                   "type": "integer"
               }
           ]
       },
       {
           "id": "remote.dns_address_match",
           "label": "DNS record address match",
           "description": "Alarm which returns CRITICAL if the DNS record is
 not resolved to the provided address",
           "check_type": "remote.dns",
           "criteria": "# Match if the 127... address was in the resolution
\n# if it wasn't than default to CRITICAL\n\nif (metric['answer'] regex \".*
${address}.*\") {\n  return OK, \"Resolved the correct address!\"\n}\nreturn
 CRITICAL\n",
           "fields": [
               {
                   "name": "address",
                   "description": "Address to which the DNS record must
 resolve to",
                   "type": "string"
               }
           ]
       },
       {
           "id": "remote.ssh_banner_match",
           "label": "SSH banner match",
           "description": "Alarm which returns CRITICAL if the service
 listening on SSH port doesn't return a valid banner",
           "check_type": "remote.ssh",
           "criteria": "/* Have the check match at the edge */\nif
 (metric['banner_matched'] != \"\") {\n  return OK\n}\n\n/* Or use the regex
 parser in the\n   language to build multiple matches\n   in a single alarm.
 */\nif (metric['banner'] regex \"OpenSSH.*\") {\n  return OK\n}\n\nreturn
 CRITICAL, \"Match not found.\"\n",
           "fields": []
       },
       {
           "id": "remote.ssh_fingerprint_match",
```

```
            "label": "SSH fingerprint match",
            "description": "Alarm which returns CRITICAL if the SSH
 fingerprint doesn't match the provided one",
            "check_type": "remote.ssh",
            "criteria": "if (metric['fingerprint'] != \"${fingerprint}\
") {\n  return CRITICAL, \"SSH fingerprint didn't match the expected
 one ${fingerprint}\"\n}\n\nreturn OK, \"Got expected SSH fingerprint
 (${fingerprint})\"\n",
            "fields": [
                {
                    "name": "fingerprint",
                    "description": "Expected SSH fingerprint",
                    "type": "string"
                }
            ]
        },
        {
            "id": "remote.ping_packet_loss",
            "label": "Ping packet loss",
            "description": "Alarm which returns WARNING if the packet loss is
 greater than 5% and CRITICAL if it's greater than 20%",
            "check_type": "remote.ping",
            "criteria": "if (metric['available'] < 80) {\n  return CRITICAL,
 \"Packet loss is greater than 20%\"\n}\n\nif (metric['available'] < 95) {\n
 return WARNING, \"Packet loss is greater than 5%\"\n}\n\nreturn OK, \"Packet
 loss is normal\"\n",
            "fields": []
        },
        {
            "id": "remote.tcp_connection_time",
            "label": "Connection time",
            "description": "Alarm which returns WARNING or CRITICAL based on
 the connection time",
            "check_type": "remote.tcp",
            "criteria": "if (metric['duration'] > ${critical_threshold}) {\
n  return CRITICAL, \"TCP Connection took more than ${critical_threshold}
 milliseconds.\"\n}\n\nif (metric['duration'] > ${warning_threshold}) {\
n  return WARNING, \"TCP Connection took more than ${warning_threshold}
 milliseconds.\"\n}\n\nreturn OK, \"TCP connection time is normal\"\n",
            "fields": [
                {
                    "name": "warning_threshold",
                    "description": "Warning threshold (in seconds) for the
 connection time",
                    "type": "integer"
                },
                {
                    "name": "critical_threshold",
                    "description": "Critical threshold (in seconds) for the
 connection time",
                    "type": "integer"
                }
            ]
        },
        {
            "id": "remote.dns_spf_record_include_match",
            "label": "SPF TXT record",
            "description": "Alarm which returns CRITICAL if the SPF record
 doesn't contain an include clause with the provided domain.",
            "check_type": "remote.dns",
```

```
            "criteria": "if (metric['answer'] regex \"v=spf1.* include:
${domain} .*[~|-]all\") {\n  return OK, \"SPF record with include clause for
 domain ${domain} exists\"\n}\n\nreturn CRITICAL, \"SPF record doesn't contain
 an include clause for domain ${domain}\"\n",
            "fields": [
                {
                    "name": "domain",
                    "description": "Domain to check for",
                    "type": "string"
                }
            ]
        },
        {
            "id": "remote.dns_dkim_public_key_match",
            "label": "DKIM TXT record",
            "description": "Alarm which returns CRITICAL if the DKIM record
 doesn't contain or match the provided public key.",
            "check_type": "remote.dns",
            "criteria": "if (metric['answer'] regex \".*p=${public_key}$\")
 {\n  return OK, \"DKIM record contains a provided public key\"\n}\n\nreturn
 CRITICAL, \"DKIM record doesn't contain a provided public key\"\n",
            "fields": [
                {
                    "name": "public_key",
                    "description": "Public key to check for. Note: Special
 characters must be escaped.",
                    "type": "string"
                }
            ]
        }
    ],
    "metadata": {
        "count": 12,
        "limit": null,
        "marker": null,
        "next_href": null
    }
}
```

Example 4.96. List Alarm Examples Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<container>
  <values>
    <alarm_template id="remote.http_body_match_1">
      <label>Body match</label>
      <description>Alarm which returns CRITICAL if the provided string is
 found in the body</description>
      <check_type>remote.http</check_type>
      <criteria>if (metric['body_match'] regex "${string}") {
  return CRITICAL, "${string} found, returning CRITICAL."
}
</criteria>
      <fields>
        <field>
          <name>string</name>
          <description>String to check for in the body</description>
          <type>string</type>
```

```
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.http_body_match_missing_string">
      <label>Body match</label>
      <description>Alarm which returns CRITICAL if the provided string is not
 found in the body</description>
      <check_type>remote.http</check_type>
      <criteria>if (metric['body_match'] == "") {
  return CRITICAL, "HTTP response did not contain the correct content."
}

return OK, "HTTP response contains the correct content"
</criteria>
      <fields/>
    </alarm_template>
    <alarm_template id="remote.http_connection_time">
      <label>Connection time</label>
      <description>Alarm which returns WARNING or CRITICAL based on the
 connection time</description>
      <check_type>remote.http</check_type>
      <criteria>if (metric['duration'] &gt; ${critical_threshold}) {
  return CRITICAL, "HTTP request took more than ${critical_threshold}
 milliseconds."
}

if (metric['duration'] &gt; ${warning_threshold}) {
  return WARNING, "HTTP request took more than ${warning_threshold}
 milliseconds."
}

return OK, "HTTP connection time is normal"
</criteria>
      <fields>
        <field>
          <name>warning_threshold</name>
          <description>Warning threshold (in milliseconds) for the connection
 time</description>
          <type>integer</type>
        </field>
        <field>
          <name>critical_threshold</name>
          <description>Critical threshold (in milliseconds) for the connection
 time</description>
          <type>integer</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.http_status_code">
      <label>Status code</label>
      <description>Alarm which returns WARNING if the server responses with
 4xx status code or CRITICAL if it responds with 5xx status code</description>
      <check_type>remote.http</check_type>
      <criteria>if (metric['code'] regex "4[0-9][0-9]") {
  return CRITICAL, "HTTP server responding with 4xx status"
}

if (metric['code'] regex "5[0-9][0-9]") {
  return CRITICAL, "HTTP server responding with 5xx status"
}
```

```
return OK, "HTTP server is functioning normally"
</criteria>
      <fields/>
    </alarm_template>
    <alarm_template id="remote.http_cert_expiration">
      <label>SSL certificate expiration time</label>
      <description>Alarm which returns WARNING or CRITICAL based on the
 certificate expiration date</description>
      <check_type>remote.http</check_type>
      <criteria>if (metric['cert_end_in'] &lt; ${critical_threshold}) {
  return CRITICAL, "Cert expiring in less than ${critical_threshold} seconds."
}

if (metric['cert_end_in'] &lt; ${warning_threshold}) {
  return WARNING, "Cert expiring in less than ${warning_threshold} seconds."
}

return OK, "HTTP certificate doesn't expire soon."
</criteria>
      <fields>
        <field>
          <name>warning_threshold</name>
          <description>Warning threshold (in seconds) for the certificate
 expiration</description>
          <type>integer</type>
        </field>
        <field>
          <name>critical_threshold</name>
          <description>Critical threshold (in seconds) for the certificate
 expiration</description>
          <type>integer</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.dns_address_match">
      <label>DNS record address match</label>
      <description>Alarm which returns CRITICAL if the DNS record is not
 resolved to the provided address</description>
      <check_type>remote.dns</check_type>
      <criteria># Match if the 127... address was in the resolution
# if it wasn't than default to CRITICAL

if (metric['answer'] regex ".*${address}.*") {
  return OK, "Resolved the correct address!"
}
return CRITICAL
</criteria>
      <fields>
        <field>
          <name>address</name>
          <description>Address to which the DNS record must resolve to</
description>
          <type>string</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.ssh_banner_match">
      <label>SSH banner match</label>
```

```
      <description>Alarm which returns CRITICAL if the service listening on
 SSH port doesn't return a valid banner</description>
      <check_type>remote.ssh</check_type>
      <criteria>/* Have the check match at the edge */
if (metric['banner_matched'] != "") {
  return OK
}

/* Or use the regex parser in the
   language to build multiple matches
   in a single alarm. */
if (metric['banner'] regex "OpenSSH.*") {
  return OK
}

return CRITICAL, "Match not found."
</criteria>
      <fields/>
    </alarm_template>
    <alarm_template id="remote.ssh_fingerprint_match">
      <label>SSH fingerprint match</label>
      <description>Alarm which returns CRITICAL if the SSH fingerprint doesn't
 match the provided one</description>
      <check_type>remote.ssh</check_type>
      <criteria>if (metric['fingerprint'] != "${fingerprint}") {
  return CRITICAL, "SSH fingerprint didn't match the expected one
 ${fingerprint}"
}

return OK, "Got expected SSH fingerprint (${fingerprint})"
</criteria>
      <fields>
        <field>
          <name>fingerprint</name>
          <description>Expected SSH fingerprint</description>
          <type>string</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.ping_packet_loss">
      <label>Ping packet loss</label>
      <description>Alarm which returns WARNING if the packet loss is greater
 than 5% and CRITICAL if it's greater than 20%</description>
      <check_type>remote.ping</check_type>
      <criteria>if (metric['available'] &lt; 80) {
  return CRITICAL, "Packet loss is greater than 20%"
}

if (metric['available'] &lt; 95) {
  return WARNING, "Packet loss is greater than 5%"
}

return OK, "Packet loss is normal"
</criteria>
      <fields/>
    </alarm_template>
    <alarm_template id="remote.tcp_connection_time">
      <label>Connection time</label>
      <description>Alarm which returns WARNING or CRITICAL based on the
 connection time</description>
```

```
      <check_type>remote.tcp</check_type>
      <criteria>if (metric['duration'] &gt; ${critical_threshold}) {
  return CRITICAL, "TCP Connection took more than ${critical_threshold}
 milliseconds."
}

if (metric['duration'] &gt; ${warning_threshold}) {
  return WARNING, "TCP Connection took more than ${warning_threshold}
 milliseconds."
}

return OK, "TCP connection time is normal"
</criteria>
      <fields>
        <field>
          <name>warning_threshold</name>
          <description>Warning threshold (in seconds) for the connection
 time</description>
          <type>integer</type>
        </field>
        <field>
          <name>critical_threshold</name>
          <description>Critical threshold (in seconds) for the connection
 time</description>
          <type>integer</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.dns_spf_record_include_match">
      <label>SPF TXT record</label>
      <description>Alarm which returns CRITICAL if the SPF record doesn't
 contain an include clause with the provided domain.</description>
      <check_type>remote.dns</check_type>
      <criteria>if (metric['answer'] regex "v=spf1.* include:${domain} .
*[~|-]all") {
  return OK, "SPF record with include clause for domain ${domain} exists"
}

return CRITICAL, "SPF record doesn't contain an include clause for domain
 ${domain}"
</criteria>
      <fields>
        <field>
          <name>domain</name>
          <description>Domain to check for</description>
          <type>string</type>
        </field>
      </fields>
    </alarm_template>
    <alarm_template id="remote.dns_dkim_public_key_match">
      <label>DKIM TXT record</label>
      <description>Alarm which returns CRITICAL if the DKIM record doesn't
 contain or match the provided public key.</description>
      <check_type>remote.dns</check_type>
      <criteria>if (metric['answer'] regex ".*p=${public_key}$") {
  return OK, "DKIM record contains a provided public key"
}

return CRITICAL, "DKIM record doesn't contain a provided public key"
</criteria>
```

```
      <fields>
        <field>
          <name>public_key</name>
          <description>Public key to check for. Note: Special characters must
 be escaped.</description>
          <type>string</type>
        </field>
      </fields>
    </alarm_template>
  </values>
  <metadata>
    <count>12</count>
    <limit/>
    <marker/>
    <next_href/>
  </metadata>
</container>
```

4.14.2. Get Alarm Example

Verb	URI	Description
GET	/alarm_examples/*alarmExampleId*	Get a specific alarm example.

There are no parameters for this call.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

This call returns a single alarm example.

Example 4.97. Get Alarm Examples Response: JSON

```
{
    "id": "remote.http_body_match_1",
    "label": "Body match",
    "description": "Alarm which returns CRITICAL if the provided string is
 found in the body",
    "check_type": "remote.http",
    "criteria": "if (metric['body_match'] regex \"${string}\") {\n  return
 CRITICAL, \"${string} found, returning CRITICAL.\"\n}\n",
    "fields": [
        {
            "name": "string",
            "description": "String to check for in the body",
            "type": "string"
        }
    ]
}
```

Example 4.98. Get Alarm Examples Response: XML

```
<?xml version="1.0" encoding="utf-8"?>
<alarm_template id="remote.http_body_match_1">
```

```
  <label>Body match</label>
  <description>Alarm which returns CRITICAL if the provided string is found in
 the body</description>
  <check_type>remote.http</check_type>
  <criteria>if (metric['body_match'] regex "${string}") {
  return CRITICAL, "${string} found, returning CRITICAL."
}
</criteria>
  <fields>
    <field>
      <name>string</name>
      <description>String to check for in the body</description>
      <type>string</type>
    </field>
  </fields>
</alarm_template>
```

4.14.3. Evaluate Alarm Example

Verb	URI	Description
POST	/alarm_examples/*alarmExampleId*	Evaluate a specific alarm example.

The parameters are specific to the alarm example.

Normal Response Code: 200

Error Response Codes: 401, 403, 500, 503

This call evaluates the template of a single alarm example. It takes arbitrary key/value pairs as specified by the fields section of the list call of the API. You then can evaluate the alarm example with a POST to the specific endpoint.

Example 4.99. Evaluate Alarm Examples Request: JSON

```
{
  "values": {
    "string": "12345"
  }
}
```

Example evaluating of a remote.http body match against '12345'. **POST** to https://monitoring.api.rackspacecloud.com/v1.0/*tenantId*/alarm_examples/remote.http_body_match_1

Example 4.100. Evaluate Alarm Example Response: JSON

```
{
  "bound_criteria": "if (metric['body_match'] regex \"12345\") {\n  return
 CRITICAL, \"12345 found, returning CRITICAL.\"\n}\n"
}
```

Appendix A. Alert Triggering and Alarms

This section describes alerting including an explanation of the alert flow, the alarm language, the policies that you can create using alarms and example best practices. In short, Rackspace Cloud Monitoring uses alarms to evaluate the metrics of a check and decide if a notification plan should be executed. It is the primary way to describe exactly what you want to be alerted on.

A.1. Alert Flow

Let's take an example user work flow of creating a monitor for a particular resource and follow it through the system to understand how the alerting system works:

1. Using the Create Check call, create a check with one or more monitoring zones (per the `monitoring_zone_poll` attribute).

2. Using the Create Alarm call, create a new alarm on the entity that matches this particular check.

   Note that alarms are created to match when a specific condition occurs. On this alarm let's assume you've specified the alarm policy as `QUORUM`. This parameter describes a deterministic way to represent mixed results in a "multi-datacenter" monitoring environment. To learn more about this concept, see Alert Policies. You can also read Best Practices on Alerting for more pattern applications of the alarm language.

3. When you apply the check, events provision the collectors. If the check is successfully applied (as indicated by the HTTP response code) the monitor starts executing its checks.

4. If the monitored resource fails, a state change event is generated (since all the collectors agree on the status per the `QUORUM` alert policy) and an alert is triggered. Based on the logic you created in the associated notification plan an error notification is sent (the call itself is a webhook).

A.2. Alarm Language

The alarm language is one of the most powerful parts of Rackspace Cloud Monitoring. It describes the mechanism to trigger an event. Upon triggering an event a notification plan is executed that describes how to send different notifications.

Note

If a check fails to execute, by default alarm associated with check returns a `CRITICAL` state.

This may change in future versions of the product, however this is currently the only behavior allowed. This represents a subclass of failures similar to *"Connection Timeout"*s or other errors where the result wasn't simply a failure result, but a result where the user was unable to run the check at all.

A.2.1. Check Availability

As mentioned above the default evaluation of a check depends upon whether the check is able to run successfully. We can illustrate this concept using the HTTP check as an example. If the alarm checks the status of a 404 response, but the check is actually getting a Connection Refused message, the result of that check is ERR0R. The availability of the check is determined by the ability to run the check.

Note

This is **very important** to understand this concept. All checks that go into a `CRITICAL` state will always force an *alarm* on an *entity*, `CRITICAL`.

A.2.2. Anatomy of a Query

An alarm query is broken down into the following main parts:

Comments

Comments are either line by line comments that begin with a # or c-style comments /* */.

```
# This is a comment
/* This is a comment */
// This is NOT a comment
```

String Literals

String literals are surrounded with either ' or ". String literals support the following escape sequences:

Sequence	Value
`\"`	Double quote
`\'`	Single quote
`\\`	Backslash
`\b`	Backspace
`\f`	Formfeed
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\uXXXX`	Unicode character where XXXX is the hex unicode character code

Some example string literals:

```
"Foo"           /* A double quoted string */
'Foo'           /* A single quoted string */
'"Foo\'s bar\"' /* Single quoted strings may contain
 unescaped double quotes */
                /* as well as escaped single or double
 quotes */
"'Bar's foo\'"  /* Double quoted strings may contain
 unescaped single quotes */
                /* as well as escaped single or double
 quotes */
'\u0027abc'     /* A string containing an escaped
 unicode character */
```

Numeric Literals

Numeric literals are written without quotation marks. Below are some examples:

```
2773.2                  /* Numeric literal */
200                     /* Numeric literal */
-200                    /* Numeric literal */
1.2e-7                  /* Numeric literal with
 exponential notation */
```

Declarations

This part of the alarm language is the setting declarations, which globally control the evaluation of the query. The syntax is shown below:

```
:set <name>=<value>
```

The current version of the product supports two settings. The first setting specifies the consistency level.

```
:set consistencyLevel=<value>
```

This is an important setting that is typically left as `QUORUM` (the default) unless there is a specific need to change it. For more information about alerting policies and consistency levels, see Alert Policies.

The second setting is the consecutive alert count. It determines how many consecutive evaluations of a state occur before issuing a state change. The default for this setting is **1**.

```
:set consecutiveCount=<value>
```

Conditionals

The second part of the query is the conditional statement. The conditional statements determine what criterion constitute sending an alert on behalf of the user. This is by far the most configurable part of the alarm language. There are two types of comparisons: numeric comparisons and text comparisons.

Numeric comparisons have a number of different operators, which are listed below:

```
== /* Equality when compared with a literal numeric */
!= /* Inequality */
>= /* Greater than or equal to */
<= /* Less than or equal to */
<  /* Less than */
>  /* Greater than */
```

If the left hand side of the conditional is a metric statement and the right hand side of the equality is another metric statement, then equality and inequality is evaluated based on lexicographical comparison.

Or if the left or right hand side is a literal than the following operators are available for use.

```
==     /* String comparison */
!=     /* String comparison */
regex  /* Regular expression match */
nregex /* Regular expression inverse match */
```

On top of the single conditional operators, you can also use boolean logic to evaluate multiple conditionals in a single alarm. When trying to determine if a resource is functioning correctly, this built-in flexibility provides you with a powerful tool that lets you examine multiple aspects of a single resource.

The operators available are:

```
&& /* And */
|| /* Or */
```

Return Statements

The third part of the query is the return statements. The return statements determine the notification or notifications to execute on the notification plan as well as the state of the alarm. There are two separate methods to represent a return query:

Returning the status:

```
return <OK|WARNING|CRITICAL>
```

Returning the status and message:

```
return <OK|WARNING|CRITICAL>, <String Status Message>
```

A.2.3. Limits and Defaults

Alarms have limits in their constructs. For instance, there are a limited set of conditionals you can use in a single alarm query.

The following list describes the limits and defaults for alarms:

- Minimum conditionals in a single query: **0**
- Maximum limit of conditionals in a single query:**10**
- Criteria: **Optional**

  Not that if criteria is not specified the availability of the check determines the alarm state.
- Default consistency level of the alert policy: **QUORUM**
- Default consecutive alert count: **1**

A.2.4. Status Messages

Checks and Alarms have status strings and there is a resolution policy for final message that get displayed to a user in an email or alarm change log or webhook. This message represents a human readable string for the status of the alarm.

The resolution policy is as follows:

- If no status is specified, then use the value from the most recent run *check*.
- If it's specified then use the specified string from the *alarm*.
- String interpolate the message if metric is present.

Status string interpolation will substitute metrics in a special format to the point in time metric. It looks like this:

```
return WARNING, "The check took #{duration}s to execute"
```

Note

String Interpolation will substitute a `#{metric-name}` for its corresponding point in time value, it is a very powerful feature.

A.3. Alert Policies (Consistency Level)

Alert policies define a system for interpreting mixed results from a check. Mixed results can occur during failure scenarios if you are monitoring a resource in multiple monitoring zones. For instance, if you're monitoring a website from three different monitoring zones and the website goes down, a `QUORUM` calculation consisting of two monitoring zones would need to agree before sending an alert.

There three different interpretations and alert policies for handling mixed results. Each interpretation has trade-offs that should be considered when determining which policy to use. The interpretation policy and their trade-offs are described below

A.3.1. One

A single monitoring zone failure. For example, an alert is triggered if one of three, or say one of five, monitoring zones report the failure.

The `ONE` policy optimizes speed of alerting at the expense of correctness. For instance, any network blip from the Rackspace Cloud Monitoring to the monitored resource would potentially generate an alert. This is mitigated in the `QUORUM` policy.

A.3.2. Quorum

A failure is detected in a majority of the monitoring zones. For example, two of three, or three of five monitoring zones report the failure. The calculation is TOTAL / 2 + 1

The `QUORUM` policy is the recommended solution. It offers the best speed to correctness trade-off. Only a majority of the infrastructure monitoring your resource has to agree that the resource is in fact down before sending an alert. This is the best approach to maintain speed and low time-to-alert.

A.3.3. All

All monitoring zone's agree the resource is down. For example, three of out of three monitoring zones report the failure.

The ALL policy is the most accurate, but is also prone to failure in significant failure scenarios. If a network partition between our internal datacenters happens the alert could be delayed due to the election process. In this case a machine has to be marked down, then the checks will be re-evaluated as a group. If they come to a consensus (with the downed collector) then an alert is generated.

A.4. Constructs and Functions

Function modifiers serve to alter the interpretation of a metric. There is currently a single modifier. The format of a modifier is as follows:

```
ex: <funcname>(metric['response_time'])
```

A.4.1. Previous

Function name: **previous**

This is used to look back at the same metric in the previous time period from the same datacenter. This is useful to make sure a value is always incrementing. Or another use is detecting string changes and sending an alert on that.

```
if (previous(metric['fingerprint']) != metric['fingerprint']) {
    return new AlarmStatus(CRITICAL, 'Fingerprint has changed to:
 #{fingerprint}');
}
```

A.4.2. Rate

Function name: **rate**

This is best used for counters. For instance if you are tracking a gauge such as bytes_in on an network interface, this will give you the *rate* as defined by this formula where V=value, and T=time.

$(V_1 - V_0) / (T_1 - T_0)$

A.5. Best Practices on Alerting

This section covers common solution patterns for creating useful alerts. It focuses on alarms and how you can use the alarm language to best achieve these patterns.

A.5.1. Best Practices for HTTP/S Check

A.5.1.1. Check for HTTP 404

Critical on 404 or Connection Refused

This example assumes a provisioned Remote HTTP with standard settings. It checks that the return code (which is a metric of type string) is the string equivalent of a 404. HTTP

response codes are numeric, but since they hold no numeric value, we interpret them as strings.

```
if (metric['code'] == "404") {
  return CRITICAL, "Page not found!"
}
```

A.5.1.2. Critical on a specific body match

Check for the existence of a body match and error out if present

This example assumes a provisioned Remote HTTP with an metric called `body_match` added to the response. You can use this string metric to check the existence of the text, and error out if found.

Using the HTTPS prefix automatically defaults the port to the standard 443 instead of port 80. This particular example looks for the word "forbidden" in the body match, and if found returns `CRITICAL` with the error message: `"Forbidden found, returning CRITICAL."`

```
if (metric['body_match'] regex ".*forbidden.*") {
  return CRITICAL, "Forbidden found, returning CRITICAL."
}
```

A.5.1.3. Critical on certificate expiration

Check the `cert_end_in` metric; critical if less than a week away

This example assumes a provisioned Remote HTTP against an HTTPS server and adds a set of metrics that are specific to SSL in the hash of metrics.

This example checks the certificate expiration in seconds, abbreviated as the `cert_end_in`:

```
/* 1 week = 604 800 seconds */
if (metric['cert_end_in'] < 604800) {
  return WARNING, "Cert expiring in less than 1 week."
}
/* 2 days = 172 800 seconds */
if (metric['cert_end_in'] < 172800) {
  return CRITICAL, "Cert expiring in less than 2 days."
}
```

A.5.2. Best Practices for Port/Banner Checks

A.5.2.1. Failure on banner match

This example assumes a provisioned Remote TCP check. It also specifies a `banner_match` *`'OpenSSH.*'`*, which matches content based on the text sent from the server upon connection. For a complete description, see Remote TCP. However if a banner matches, then a metric is added to the result, called `banner_match`. One common solution is to check for the existence of that metric and return `CRITICAL` otherwise.

```
/* Have the check match at the edge */
if (metric['banner_matched'] != "") {
  return OK
}
/* Or use the regex parser in the
   language to build multiple matches
   in a single alarm. */
if (metric['banner'] regex "OpenSSH.*") {
  return OK
}

return CRITICAL, "Match not found."
```

A.5.3. Best Practices for DNS Checks

A.5.3.1. Check for an IP in a DNS query, fail otherwise.

This example assumes a provisioned Remote DNS check against a working nameserver. In this example, the alarm matches against the `answer` metric. As with all alarms, if the check is marked available=false (which in this case means the nameserver fails to respond) than the alarm is `CRITICAL`.

```
# Match if the 127... address was in the resolution
# if it wasn't than default to CRITICAL

if (metric["answer"] regex ".*127.8.2.1.*") {
  return OK, "Resolved the correct address!"
}
return CRITICAL
```

A.5.4. Best Practices for SSH Checks

The following example uses the Rackspace Cloud Monitoring command line interface (CLI). For information on downloading and installing the CLI, see https://github.com/racker/rackspace-monitoring-cli.

One of the most widely used remote checks is the SSH check. This check not only verifies that something is listening on the expected port, but establishes an SSH session and returns the host key fingerprint as a metric, further verifying that the SSH server is operating as expected.

The following example assumes the existence of an entity with the IP address eth0 and ID enk8YUv0Cd, and a notification plan with ID nplU9hLUgc. This check connects to an SSH server using port 22 by default:

```
raxmon-checks-create \
--entity-id=enk8YUv0Cd \
--label=ssh \
--type=remote.ssh \
--target-alias=eth0 \
--monitoring-zones=mzord,mzdfw,mzlon
```

Alarm for this check:

If the monitoring service is unable to connect to the SSH server for the check, any alarms using the check will automatically fail. However, we can additionally verify that the server

returns the expected host key fingerprint, which could reveal an unexpected change on the server or a man in the middle attack.

```
raxmon-alarms-create \
--entity-id=enk8YUv0Cd \
--notification-plan-id=nplU9hLUgc \
--check-id=chTFHxHn0p \
--criteria="if (metric['fingerprint'] != '13dd6c5df600f9a15c67ea5db491ac9a')
 { return CRITICAL, 'Incorrect SSH Host Fingerprint' }"
```

Appendix B. Client Libraries and Tools

This section describes client libraries and tools that you can use to interact with the API.

B.1. Client Libraries

B.1.1. Python - *"rackspace-monitoring"* Library

The official client library for interacting with the API is written in Python and based on the open-source http://libcloud.apache.org Apache Libcloud framework.

Access the library from the following website:

https://github.com/racker/rackspace-monitoring

Use the following command to install the library:

```
$ pip install rackspace-monitoring
```

B.1.2. Ruby - *"rackspace-monitoring"* Library

The official client library for interacting with the API is written in Ruby and based on the open-source http://fog.io/ Fog library.

Access the library from the following website:

https://github.com/racker/rackspace-monitoring-rb

Use the following command to install the library:

```
$ gem install rackspace-monitoring
```

B.2. Tools (CLI, etc...)

B.2.1. Command Line Interface - *"Raxmon"* Tool

Rackspace Cloud Monitoring provides a command line utility for performing API actions.

Access the library from the following website:

https://github.com/racker/rackspace-monitoring-cli

Use the following command to install the command line utility:

```
$ pip install --upgrade rackspace-monitoring-cli
```

B.2.2. Integration - *"Cloud Monitoring Cookbook"*

Rackspace Cloud Monitoring provides a cookbook for integrating with Opscode Chef.

Access the code and read some of the ways to use the integration here:

https://github.com/racker/cookbook-cloudmonitoring

And here:

http://www.rackspace.com/knowledge_center/article/cloud-monitoring-automation-integrate-monitoring-with-chef

Follow the instructions to install the Chef integration.

Glossary

A

Alarm
: An alarm contains a set of rules that determine when a notification is triggered.

C

Check
: Checks explicitly specify how you want to monitor an entity.

Collector
: A collector collects data from the monitoring zone. The collector is mapped directly to an individual machine or a virtual machine.

E

Entity
: An entity is a resource that you want to monitor. Some examples are a server, a website, or a service.

M

Monitoring Zone
: A monitoring zone is the "launch point" of a check. You can launch checks from multiple monitoring zones.

N

Notification
: A notification is an informational message sent to one or more addresses when an alarm is triggered.

Notification Plan
: A notification plan is a set of notification rules to execute when an alarm is triggered.

R

RESTful
: A type of web service API that uses Representational State Transfer. REST is the architectural style for hypermedia systems used for the World Wide Web.