GWD-R
OCCI-WG

Ralf Nyrén

September 16, 2011
Updated: September 18, 2011

Open Cloud Computing Interface - JSON Rendering

Status of this Document

This document provides information to the community regarding the specification of the Open Cloud Computing Interface. Distribution is unlimited.

Abstract

This document, part of a document series, produced by the OCCI working group within the Open Grid Forum (OGF), provides a high-level definition of a Protocol and API. The document is based upon previously gathered requirements and focuses on the scope of important capabilities required to support modern service offerings.

Comments

Contents

1 Introduction

2 Notational Conventions

todo

3 Mandatory HTTP headers

HTTP headers which MUST be present in every JSON-rendered request and response.

3.1 Mandatory HTTP request headers

Accept: application/occi+json

Content-Type: application/occi+json

3.2 Mandatory HTTP response headers

Content-Type: application/occi+json

4 JSON representation

RN: This section would describe all the details of the JSON objects used to represent resource instances and collections. IOW lots of ABNF, MUST/MUST-NOTs, etc... but for now an example will have to do.

The JSON representation of a Compute instance. Notice that the "full" category representation is used here, i.e. what you would normally only see in the discovery interface. It has been included here to show how things are rendered.

```
{
  "categories": [
    {
      "term": "compute",
      "scheme": "http://schemas.ogf.org/occi/infrastructure#",
      "class": "kind",
      "title": "Compute Resource",
      "related": "http://schemas.ogf.org/occi/core#resource",
      "attributes": {
        "occi.compute.architecture": {
          "mutable": true,
          "required": false,
          "type": "string"
        },
        "occi.compute.cores": {
          "mutable": true,
          "required": false,
          "type": "integer"
        },
        "occi.compute.hostname": {
          "mutable": true,
          "required": false,
          "type": "string"
```

```
        },
        "occi.compute.speed": {
          "mutable": true,
          "required": false,
          "type": "float"
        },
        "occi.compute.memory": {
          "mutable": true,
          "required": false,
          "type": "float"
        },
        "occi.compute.state": {
          "mutable": false,
          "required": false,
          "type": "string"
        }
      },
      "actions": [
        "http://schemas.ogf.org/occi/infrastructure/compute/action#start",
        "http://schemas.ogf.org/occi/infrastructure/compute/action#stop",
        "http://schemas.ogf.org/occi/infrastructure/compute/action#restart",
        "http://schemas.ogf.org/occi/infrastructure/compute/action#suspend"
      ],
      "location": "/api/compute/"
    }
  ],
  "actions": [
    {
      "title": "Start Compute Resource",
      "uri": "/api/compute/92cfd112-cfe5-406b-a019-f0a2334d5ddc?action=start",
      "type": "http://schemas.ogf.org/occi/infrastructure/compute/action#start"
    }
  ],
  "links": [
    {
      "title": "My disk",
      "target_uri": "http://www.nyren.net/api/storage/6d076470-d64b-4fbd-8bef-39eff94a89ac",
      "target_type": [
        "http://schemas.ogf.org/occi/infrastructure#storage"
      ],
      "link_uri": "http://www.nyren.net/api/link/storage/0f8f91dc-7e8e-4b96-8a3b-8e864186fe8c",
      "link_type": [
        "http://schemas.ogf.org/occi/infrastructure#storagelink"
      ],
      "attributes": {
        "occi.storagelink.deviceid": "ide:0:1"
      }
    }
  ],
  "attributes": {
    "occi.core.id": "urn:uuid:92cfd112-cfe5-406b-a019-f0a2334d5ddc",
    "occi.compute.architecture": "x86_64",
    "occi.compute.speed": 2.6699999999999999,
    "occi.compute.memory": 1.0,
    "occi.compute.state": "inactive"
  },
```

```
  "location": "http://www.nyren.net/api/compute/92cfd112-cfe5-406b-a019-f0a2334d5ddc"
}
```

5 HTTP methods applied to resource instance URLs

This section describes the HTTP methods used to retrieve and manipulate individual resource instances. Each HTTP method described is assumed to operate on an URL referring to a single element in a collection, an URL such as the following:

```
http://example.com/compute/012d2b48-c334-47f2-9368-557e75249042
```

An OCCI client MUST supply the mandatory headers described in section 3 with every HTTP request.

5.1 GET resource instance

The HTTP GET method retrieves the JSON representation of an OCCI resource instance.

5.1.1 Client GET request

The body of the HTTP GET request MUST be empty.

```
GET /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi+json
User-Agent: occi-client/x.x OCCI/1.1
```

5.1.2 Server GET response

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Category: compute;
          scheme="http://schemas.ogf.org/occi/infrastructure#";
          class="kind";
          title="Compute Resource"
Content-Type: application/occi+json; charset=utf-8

{
  "categories": [ ... ],
  "actions": [ ... ],
  "links": [ ... ],
  "attributes": { ... },
  "location": "http://example.com/compute/012d2b48-c334-47f2-9368-557e75249042"
}
```

5.2 PUT resource instance

The HTTP PUT method creates or replaces the resource instance at the specified URL. Since the resource identifier is supplied in the request URL an OCCI server MAY refuse to create a new instance.

5.2.1 Client PUT request

The full JSON representation of the resource instance MUST be supplied in the HTTP body of the request.

RN: Including Links in the request breaks the rule of PUT being idempotent. Simply prohibit Links in PUT requests?

```
PUT /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi+json; charset=utf-8

{
  "categories": [ ... ],
  "attributes": { ... },
}
```

5.2.2 Server PUT response

Upon success an OCCI server MUST return HTTP status code 200 and a complete JSON representation of the created/replaced resource instance. The response MUST be identical[1] of the to that of a subsequent GET request same URL.

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Category: compute;
          scheme="http://schemas.ogf.org/occi/infrastructure#";
          class="kind";
          title="Compute Resource"
Content-Type: application/occi+json; charset=utf-8

{
  "categories": [ ... ],
  "actions": [ ... ],
  "links": [ ... ],
  "attributes": { ... },
  "location": "http://example.com/compute/012d2b48-c334-47f2-9368-557e75249042"
}
```

5.3 POST resource instance with "action" query parameter

An OCCI Action is invoked using the HTTP POST method together with query parameter named "action".

5.3.1 Client POST action request

```
POST /compute/012d2b48-c334-47f2-9368-557e75249042?action=stop HTTP/1.1
Host: example.com
Accept: application/occi+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi+json; charset=utf-8

{
```

[1] Provided the resource instance was not changed in the meantime.

```
  "categories": [{
    "term": "stop",
    "scheme": "http://schemas.ogf.org/occi/infrastructure/compute/action#",
    "class": "kind"
  }],
  "attributes": {
    "method": "graceful"
  },
}
```

5.3.2 Server POST action response

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

5.4 POST resource instance without any query parameters

RN: This would imply a partial update of the resource instance. While it is easy to supply only the attributes to be updated the question is if there are any valid use cases for partial updates using JSON?

5.5 DELETE resource instance

The HTTP DELETE method destroys a resource instance and any OCCI Links associated with the resource instance.

5.5.1 Client DELETE request

```
DELETE /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi+json
```

5.5.2 Server DELETE response

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

6 HTTP methods applied to collections URLs

This section describes the HTTP methods used to manipulate collections. Each HTTP method described is assumed to operate on an URL referring to a collection of elements, an URL such as the following:

```
http://example.com/storage/
```

A collection consist of a set of resource instances and there are three different types of collections which may be exposed by an OCCI server. The request and response format is identical for all three types collections although the semantics differ slightly for the PUT and POST methods.

Kind locations The location associated with an OCCI Kind instance represents the collection of all resource instances of that particular Kind.

Mixin locations The location of an OCCI Mixin instance represents the collection of all resource instances associated with that Mixin.

Arbitrary path Any path in the URL namespace which is neither a Kind nor a Mixin location. A typical example is the root URL e.g. `http://example.com/`. Such a path combines all collections in the sub-tree starting at the path. Therefore the root URL is a collection of all resource instances available.

6.1 GET collection

The HTTP GET method retrieves a list of all resource instances in the collection. Filtering and pagination information is encoded in the query string of the URL.

6.1.1 Client GET request

The query string of the request URL MUST have the following format:

```
query-string        = ""
                    | "?" query-parameter *( "&" query-parameter )
  query-parameter   = attribute-filter
                    | category-filter
                    | pagination-start
                    | pagination-count
  attribute-filter  = "q=" attribute-search *( "+" attribute-search )
  attribute-search  = 1*( string-urlencoded |
                          attribute-name "%3D" string-urlencoded )
  category-filter   = "category=" string-urlencoded
  pagination-start  = "start=" 1*( DIGIT )
  pagination-count  = "count=" 1*( DIGIT )
  attribute-name    = attr-component *( "." attr-component )
  attr-component    = LOALPHA *( LOALPHA | DIGIT | "-" | "_" )
  string-urlencoded = *( ALPHA | DIGIT | "-" | "_" | "." | "~" | "%" )
```

Filtering A search filter can be applied to categories and attributes of resource instances in a collection. An OCCI server SHOULD support filtering. The query parameters MUST be URL encoded.

Attribute filters are specified using the *q* query parameter. A filter such as `q=ubuntu+inactive` would match all resource instances whose combined set of attribute values includes both the word "ubuntu" and "inactive". It is also possible to match on specific attributes by preceding the search term with the attribute name and an equal sign, for example `occi.core.title%3Dubuntu+occi.compute.state%3Dinactive`.

The category filter is specified using the *category* query parameter and represent a single Kind, Mixin or Action category to be matched. The following query would include only resource instances of the Compute type: `category=http%3A%2F%2Fschemas.ogf.org%2Focci%2Finfrastructure%23compute`

Pagination An OCCI client MAY request that the server only return a subset of a collection. This is accomplished using the *start* and *count* query parameters. An OCCI server MUST support pagination.

The *start* parameter specifies the offset into the collection. A value of zero, `start=0` indicates the beginning of the collection. The *count* parameter sets the maximum number of elements to include in the response. For example `?start=20&count=10` would indicate the third page with a limit of 10 elements per page.

Example request

```
GET /storage/?q=ubuntu+server&start=0&count=20 HTTP/1.1
Host: example.com
Accept: application/occi+json
User-Agent: occi-client/x.x OCCI/1.1
```

6.1.2 Server GET response

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Content-Type: application/occi+json; charset=utf-8

{
  "start": 0,
  "count": 3,
  "collection": [
    {
      "categories": [ ... ],
      "actions": [ ... ],
      "links": [ ... ],
      "attributes": { ... },
      "location": "http://example.com/storage/e3578467-b6ba-448b-8032-5203278d54db"
    },
    { ... },
    { ... }
  ]
}
```

6.2 POST collection

The HTTP POST method is used to create/update one or more resource instances in a single atomic request. An OCCI server MUST identify existing resource instances using the `occi.core.id` attribute.

6.2.1 Client POST request

```
POST /storage/ HTTP/1.1
Host: example.com
Accept: application/occi+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi+json; charset=utf-8

{
  "collection": [
    {
      "categories": [ ... ],
      "links": [ ... ],
      "attributes": { ... },
    },
    { ... },
    { ... }
  ]
}
```

6.2.2 Server POST response

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

RN: Should we support HTTP 200 returning the whole collection? Or maybe just the resource instances created/updated?

6.3 POST collection with "action" query parameter

todo

6.4 PUT collection

Replace the entire collection with a new one. RN: Should we support this?

6.5 DELETE collection

Delete the entire collection. RN: Should we support this?

7 More examples

The OCCI demo instance of occi-py[2] running at `http://www.nyren.net/api/` has an early version of the draft JSON rendering available. Feel free to play around with it. However, please note the following limitations:

- The Content-Type is `application/json` and not `application/occi+json` which would be more appropriate.
- It does not support request data in JSON.
- Filtering and pagination is not yet supported.

A few example queries using curl:

```
curl -i -H 'accept: application/json' http://www.nyren.net/api/-/
curl -i -H 'accept: application/json' http://www.nyren.net/api/link/
curl -i -X POST -H 'accept: application/json' http://www.nyren.net/api/compute/
```

[2] `http://github.com/nyren/occi-py`