OGF OCCI-WG Deliverables

OCCI-WG

OGF OCCI-WG Deliverables

OCCI-WG

Publication date July 2009

Use Cases for Infrastructure as a Service based Clouds

Name of use case

Description: 3 paragraphs with brief description and figure

Functional Requirements

- TBD

Non-functional Requirements

- TBD

Requirements for an Cloud API

Functional Requirements

This section deals with the funtional requirements. The requirments have been split up in tables and prioritized.

Table 1. Functional requirements on VM description

ID	Description	Usecases	Priority

Table 2. Functional requirements on VM management

ID	Description	Usecases	Priority

Table 3. Functional requirements on Network management

ID	Description	Usecases	Priority

Table 4. Functional requirements on Storage management

ID	Description	Usecases	Priority

Table 5. Functional requirements on Image management

ID	Description	Usecases	Priority

Table 6. Identifications/References

ID	Description	Usecases	Priority

Non-functional Requirements

This section deals with all the non-funtional requirements.

Table 7. Security requirements

ID	Description	Usecases	Priority

Table 8. Quality of Service

ID	Description	Usecases	Priority

Table 9. Syntax

ID	Description	Usecases	Priority

Table 10. Backup/Disaster recovery

ID	Description	Usecases	Priority

OCCI Walkthrough

Overview

This document may lag behind the actual specification

The Open Cloud Computing Interface (OCCI) is an API for managing cloud infrastructure services (also known as Infrastructure as a Service or IaaS) which strictly adheres to REpresentational State Transfer (REST) principles and is closely tied to HyperText Tranfer Protocol (HTTP). For simplicity and scalability reasons it specifically avoids Remote Procedure Call (RPC) style interfaces and can essentially be implemented as a horizontally scalable document repository with which both nodes and clients interact.

This document describes a step-by-step walkthrough of performing various tasks as at the time of writing.

Getting Started

Connecting

Each implementation has a single OCCI end-point URL (we'll use http://example.com/) and everything you need to know is linked from this point - configuring clients is just a case of providing this parameter. In the simplest case the end-point may contain only a single resource or type of resource (e.g. a hypervisor burnt into the BIOS of a motherboard exposing compute resources, a network switch/router exposing network resources or a SAN exposing storage resources) and at the other end of the spectrum it may provide access to a global cloud infrastructure (e.g. the "Great Global Grid" or GGG). You will only ever see those resources to which you have access to (typically all of them for a private cloud or a small subset for a public cloud) and flexible categorisation and search provide fine-grained control which resources are returned, allowing OCCI to handle the largest of installations. You will always connect to this end-point over HTTP(S) and given the simplicity of the interface most user-agents are suitable, including libraries (e.g. urllib2, LWP), command line tools (e.g. curl, wget) and full blown browsers (e.g. Firefox).

Authenticating

When you connect you will normally be challenged to authenticate via HTTP (this is not always the case - in secure/offline environments it may not be necessary) and will need to do so via the specified mechanism. It is anticipated that most implementations will require HTTP Basic Authentication over SSL/TLS so at the very least you should support this (fortunately almost all user-agents already do), but more advanced mechanisms such as NTLM or Kerberos may be deployed. Certain types of accesses (such as a compute resource querying OCCI for introspection and configuration) may be possible anonymously (having already been authenticated by interface and/or IP address). Should you be redirected by the API to a node, storage device, etc. (for example, to retrieve a large binary representation) then you should either be able to transparently authenticate or a signed URL should be provided. That is, a single set of credentials is all that is required to access the entire system from any point.

Representations

As the resource itself (e.g. a physical machine, storage array or network switch) cannot be transferred over HTTP (at least not yet!) we instead make available one or more representations of that resource. For example, an API modeling a person might return a picture, fingerprints, identity document(s) or even a digitised DNA sequence, but not the person themselves. A circle might be represented by SVG drawing primatives or any three distinct points on the curve. For cloud infrastructure there are many useful representations, and while OCCI standardises a number of them for interoperability purposes,

an implementation is free to implement others in order to best serve the specific needs of their users and to differentiate from other offerings. Other examples include:

- Open Cloud Computing Interface (OCCI) descriptor format (application/occi+xml)
- Open Virtualisation Format (OVF) file (application/ovf+xml?)
- Open Virtualisation Archive (OVA) file (application/x-ova?)
- Screenshot of the console (image/png)
- Access to the console (application/x-vnc)

The client indicates which representation(s) it desires by way of the URL and/or HTTP Accept headers (e.g. HTTP Content Negotiation) and if the server is unable to satisfy the request then it should return HTTP 406 Not Acceptable.

Descriptors

In addition to the protocol itself, OCCI defines a simple key/value based descriptor format for cloud infrastructure resources:

compute	Provides computational services, ranging from dedicated physical machines (e.g. Dedibox) to virtual machines (e.g. Amazon EC2) to slices/zones/containers (e.g. Mosso Cloud Servers).
network	Provides connectivity between machines and the outside world. Usually virtual and may or may not be connected to a physical segment.
storage	Provides storage services, typically via magnetic mass storage devices (e.g. hard drives, RAID arrays, SANs).

Given the simplicity of the format it is trivial to translate between wire formats including plain text, JSON, XML and others. For example:

```
occi.compute.cores 2
compute.speed 3200
compute.memory 2048
```

Identifiers

Each resource is identified by its dereferenceable URL which is by definition unique, giving information about the origin and type of the resource as well as a local identifier (the combination of which forms a globally unique compound key). The primary drawback is that the more information that goes into the key (and therefore the more transparent it is), the more likely it is to change. For example, if you migrate a resource from one implementation to another then its identifier will change (though in this instance the source should provide a HTTP 301 Moved Permanently response along with the new location, assuming it is known, or HTTP 410 Gone otherwise).

In order to realise the benefit of transparent, dereferenceable identifiers while still being able to track resources through their entire lifecycle an immutable UUID attribute should be allocated which will remain with the resource throughout its life. This is particularly important where the same resource (e.g. a network) appears in multiple places.

New implementations should use type 4 (random) UUIDs anyway, as these can be safely allocated by any node without consulting a register/sequence, but where existing identifiers are available they should be used instead (e.g. http://amazon.com/compute/ami-ef48af86).

Operations

Create

To create a resource simply POST it to the appropropriate collection (e.g. /compute, /network or /storage) as an HTML form (supported by virtually all user agents) or in another supported format (e.g. OVF):

```
POST /compute HTTP/1.1
Host: example.com
Content-Length: 35
Content-Type: application/x-www-form-urlencoded

compute.cores=2&compute.memory=2048
```

Rather than generating the new resource from scratch you may also be given the option to GET a template and POST or PUT it back (for example, where "small", "medium" and "large" instances or pre-configured appliances are offered).

Retrieve

The simplest command is to retrieve a single resource by conducting a HTTP GET on its URL (which doubles as its identifier):

```
GET /compute/b10fa926-41a6-4125-ae94-bfad2670ca87 HTTP/1.1
Host: example.com
```

This will return *a HTTP 300 Multiple Choices response containing a list of available representations for the resource as well as a suggestion in the form of a HTTP Location: header of* the default rendering, which should be HTML (thereby allowing standard browsers to access the API directly). An arbitrary number of alternatives may also be returned by way of HTTP Link: headers.

If you just need to know what representations are available you should make a HEAD request instead of a GET - this will return the metadata in the headers without the default rendering.

Some requests (such as searches) will need to return a collection of resources. There are two options:

Pass-by-reference	A plain text or HTML list of links is provided but each needs to be retrieved separately, resulting in O(n+1) performance.
Pass-by-value	A wrapper format such as Atom is used to deliver [links to] the content as well as the metadata (e.g. links, associations, cahching information, etc.), resulting in O(1) performance.

Update

Updating resources is trivial - simply GET the resource, modify it as necessary and PUT it back where you found it.

Delete

Simply `DELETE` the resource:

```
DELETE /compute/b10fa926-41a6-4125-ae94-bfad2670ca87 HTTP/1.1
Host: example.com
```

Sub-resource Collections

(For want of a better name)

Each resource may expose collections for functions such as logging, auditing, change control, documentation and other operations (e.g. http://example.com/compute/123/log/456) in addition to any required by OCCI. As usual CRUD operations map to HTTP verbs (as above) and clients can either PUT entries directly if they know or will generate the identifiers, or POST them to the collection if this will be handled on the server side (using POST Once Exactly (POE) to ensure idempotency).

Requests

Requests are used to trigger state changes and other operations such as backups, snapshots, migrations and invasive reconfigurations (such as storage resource resizing). Those that do not complete immediately (returning HTTP 200 OK or similar) must be handled asynchronously (returning HTTP 201 Accepted or similar).

```
POST /compute/123/requests HTTP/1.1
Host: example.com
Content-Length: 35
Content-Type: application/x-www-form-urlencoded

state=shutdown&type=acpioff
```

The actual operation may not start immediately (for example, backups which are only handled daily at midnight) and may take some time to complete (for example a secure erase which requires multiple passes over the disk). Clients can poll for status periodically or use server push (or a non-HTTP technology such as XMPP) to monitor for events.

OCCI Core

Introduction

The Open Cloud Computing Interface is an open community consensus API, initially targeting cloud infrastructure services or "Infrastructure as a Service (IaaS)". A "Resource Oriented Architecture (ROA)", it is as close as possible to the underlying HyperText Transfer Protocol (HTTP), deviating only where absolutely necessary. Each resource (identified by a canonical URL) can have multiple representations which may or may not be hypertext (e.g. HTML). Metadata including associations between resources is exposed via HTTP headers (e.g. the Link: header), except in the case of collections where Atom is used as the meta-model.

Table 1. Common Attributes

Attribute	Type	Description
`id`	String *(Typically UUID Type 4)*	Random, immutable unique identifier (`atom:id`)
`title`	String	Human readable title (`atom:title`)
`summary`	String	Summary (`atom:summary`)

Basics

URL Namespace

An OCCI interface is defined by a single URL entry point (and optionally, suitable credentials for HTTP based authentication schemes). Implementors should also expose an AtomPub service document at the root to enable enumeration of resource types, supported formats and categories.

Nouns, Verbs and Attributes

Interfaces expose "nouns" which have "attributes" and on which "verbs" can be performed. The attributes are exposed as key-value pairs and appropriate verbs as links, following HATEOAS principles.

CRUD Operations

Create, Retrieve, Update and Delete (CRUD) operations map to the POST, GET, PUT and DELETE HTTP verbs respectively. HEAD and OPTIONS verbs may be used to retrieve metadata and valid operations without the entity body to improve performance. Additionally, all existing HTTP functionality is available for caching, proxying, gatewaying and other advanced functionality.

POST (Create)
: POSTing a representation (e.g. OVF) to a collection (e.g. /compute) will result in a new resource being created (e.g. /compute/123) and returned in the Location: header. POST is also used with HTML form data to trigger verbs (e.g. restart)

GET (Retrieve)
: GETting a resource (e.g. /compute/123) will return a representation of that resource in the most appropriate supported format specified by the client in the Accept header. Otherwise "406 Not Acceptable" will be returned.

PUT (Update)
: PUTting a representation (e.g. OVF) to a URL (e.g. /compute/123) will result in the resource being created or updated.

	The URL is known or selected by the client (in which case UUIDs should be used), in contrast to POSTs where the URL is selected by the server.
DELETE (Delete)	DELETE results in the deletion of the resource (and everything "under" it, as appropriate).

Collections

Operations that return multiple resources (e.g. categories, searches) are rendered as an Atom feed with an Atom entry per resource. Metadata that would normally appear in the HTTP headers appears in standard Atom elements with the entity-body itself being passed by reference or by value in the Atom content element.

Versioning

Clients and servers should expose the protocol version (e.g. OCCI/1.0) via the User-Agent and Server HTTP headers respectively. Should second or subsequent versions of the descriptor format be required the version will be added to the Internet media type (e.g. application/occi2+xml).

Extensions

Caching

Caching information improves performance by allowing clients to track freshness of cached objects.

Table 2. Caching Attributes

Attribute	Type	Description
`etag`	String	ETag (must match HTTP headers where present)
`updated`	Date	Time last updated (`atom:updated`)

Categories

Categories allow for simple, flexible organisation of information.

Table 3. Category Attributes

Attribute	Type	Description
`category[i].term`	Token	Category name (`atom:term`)
`category[i].scheme`	URI	Category vocabulary/schema (`atom:scheme`)
`category[i].label`	String	Human readable label (`atom:label`)

Links

Linking allows resources to refer to:

- Alternative representations
- Sub-collections
- Other nouns

- Related resources

Table 4. Linking Attributes

Attribute	Type	Description
`link[i].href`	URI	Link target (`atom:link[@href]`)
`link[i].rel`	URI	Link relation (`atom:link[@rel]`)
`link[i].title`	String	Human readable title (`atom:link[@title]`)

Status

Status reporting allows clients to monitor the status of a given task.

Table 5. Status Attributes

Attribute	Type	Description
`status.message`	String	Human readable status message
`status.percentage`	Float (0..100)	Percentage complete (0=not started, 100=finished)
`status.rate.average`	Float	Average rate of progress
`status.rate.current`	Float	Current rate of progress
`status.rate.units`	String	Units (e.g. `MB/s`)
`status.work.completed`	Float	Work completed
`status.work.remaining`	Float	Work remaining
`status.work.units`	String	Units (e.g. `MB`)
`status.time.start`	Date/Time	Start time
`status.time.finish`	Date/Time	Finish time (may be an estimate)
`status.time.remaining`	Time	Remaining time (may be an estimate)

Tasks

Asynchronous operations ("tasks") immediately return HTTP `202 Accepted` with a `Location:` header pointing to a simple task [sub]resource. This allows tasks to be monitored (`GET`), updated (`PUT`) and canceled (`DELETE`). Completed tasks *may* be deleted immediately, after a reasonable period of time (allowing clients to retrieve status) or retained indefinitely for audit purposes.

The collection of tasks for a given resource (including the entry-point itself for global tasks) is advertised under the `http://purl.org/occi#tasks` link relation and new tasks should be submitted via HTTP `POST` to the supplied `href`.

Table 6. Task Attributes

Attribute	Type	Description
`task.type`	Token	Task type (e.g. `backup`)
`task.sub-type`	Token	Task sub-type (e.g. `incremental`)
`task.schedule[i]`	String	Task schedule (e.g. "`every Friday at 21:00`")

Examples

The following is an example of an OCCI resource in `application/occi+txt` format:

Examples

Plain Text

The following is an example of an OCCI resource in `application/occi+txt` format:

```
id: 2acf3e85-33cb-493b-ab5c-7ef878032657
title: Resource #1
summary: Web resource for demonstration purposes
etag: "46dd20-23-464015228e7c0"
category[0].term: widget
category[0].scheme: http://example.com/products
category[0].label: Widgets
link[0].href: http://example.com/products/1234
link[0].rel: self
link[0].title: Link to myself
```

JSON (application/occi+json)

The following is an example of an OCCI resource in `application/occi+json` format:

```
{
    "id": "f63aaa26-30b7-4a30-91ca-1d03c1e52214",
    "title": "Resource #1",
    "summary": "Web resource for demonstration purposes",
    "etag": "d260e0b71609d8403d564a4b220814d4",
    "category": [{
        "term": "widget",
        "scheme": "http://example.com/products",
        "label": "Widgets"
    }],
    "link": [{
        "href": "http://example.com/products/1234",
        "rel": "self",
        "title": "Link to myself"
    }]
}
```

XML (application/occi+xml)

The following is an example of an OCCI resource in `application/occi+xml` format:

```
TBD
```

References

The following standards are referenced by this implementation.

- RFC 2616 Hypertext Transfer Protocol -- HTTP/1.1
- RFC 4287 The Atom Syndication Format

- RFC 5023 The Atom Publishing Protocol

Additionally RFC 2119 Requirement Levels are used throughout.

Registration

IANA Considerations

The following media types are to be registered:

- application/occi+txt
- application/occi+json
- application/occi+xml

OCCI Infrastructure

OCCI Infrastructure defines three nouns and various extensions relating to management of cloud infrastructure services (IaaS).

Table 1. Common Attributes

Attribute	Type	Description
`hostname`	String	Valid DNS hostname for the resource (may be FQDN)

Nouns

Cloud infrastructure can be modeled using three primary nouns: compute, network and storage.

Compute

A compute resource is capable of conducting computations (e.g. a virtual machine).

Table 2. Compute Attributes

Attribute	Type	Description
`compute.cpu.arch`	Enum (x86, x64)	CPU Architecture (e.g. x64)
`compute.cpu.cores`	Integer	Number of CPU cores (e.g. 1, 2)
`compute.cpu.speed`	Float (10^9 Hertz)	Clock speed in gigahertz (e.g. 2.4)
`compute.memory.size`	Float (10^6 bytes)	RAM in megabytes (e.g. 2048)
`compute.memory.speed`	Float (10^9 bps)	RAM speed in Gbit/s (e.g. 256)
`compute.memory.reliability`	Enum (standard, checksum)	Qualitative measure of RAM reliability (e.g. ECC)

Network

A network resource is capable of transferring data (e.g. a virtual network or VLAN).

Table 3. Network Attributes

Attribute	Type	Description
`network.vlan-id`	Integer (0..4095)	802.1q VLAN ID (e.g. 4095)
`network.vlan-tag`	Token	Tag based VLANs (e.g. external-dmz)
`network.ipv4[i].gateway`	IPv4 Address	IPv4 gateway address (e.g. 192.168.0.1)
`network.ipv4[i].netmask`	IPv4 Address	IPv4 netmask address (e.g. 255.255.255.0)
`network.ipv4[i].network`	IPv4 Address	IPv4 network address (e.g. 192.168.0.0)
`network.ipv4[i].cidr`	Integer (0..32)	Netmask in CIDR notation (e.g. 24)

Storage

A storage resource is capable of mass storage of data (e.g. a virtual hard drive).

Table 4. Storage Attributes

Attribute	Type	Description
`storage.reliability`	Enum (transient, persistent, reliable)	Qualitative device persistence (e.g. transient)
`storage.size`	Integer (10^9 bytes)	Drive size in gigabytes (e.g. 40)

Extensions

Various extensions provide for more advanced management functionality such as billing, monitoring and reporting.

State machine (state)

The state machine extension allows for the modeling of arbitrarily complex state machines and associated transitions (e.g. start, stop, restart).

OCCI Registries

Table 1. HTTP Status Codes

Code	Description	Example
200 OK	Request completed successfully	Response is returned
201 Created	Request completed successfully, resource was created	Pointer to new resource returned
202 Accepted	Request accepted, processing not completed	Workload starting but not yet active
301 Moved Permanently	Resource has been assigned a new permanent URI	Workload migrated to another installation
302 Found	Resource resides temporarily under a different URI	Alias pointing to UUID can be updated
304 Not Modified	Conditional GET on resource that is unchanged	Client already has the latest version of the resource
400 Bad Request	Request could not be understood by the server due to malformed syntax	Client sent a representation that was unable to be understood
401 Unauthorized	The request requires user authentication	Client must retry with authentication
402 Payment Required	The server has refused to fulfill the request	Credit limit exceeded
403 Forbidden	The server understood the request, but is refusing to fulfill it	Attempt to access resource without permission
404 Not Found	The server has not found the resource	Feed or entry unknown
405 Method Not Allowed	The method specified is not allowed for the resource	Attempt to delete an immutable resource
406 Not Acceptable	The resource is not capable of requested content characteristics	Unsupported output format requested
409 Conflict	Request is in conflict with the current state of the resource	Resource updated by a third-party in the interim
410 Gone	Resource is gone, no forwarding address	Resource was deleted
500 Internal Server Error	Server encountered an unexpected condition	An unknown failure has occurred (e.g. out of memory)
501 Not Implemented	Functionality required to fulfill request is not implemented	A missing extension was called
502 Bad Gateway	An invalid response was received from an upstream server	The gateway received a malformed response from a node
503 Service Unavailable	Server is temporarily unable to handle the request	Server may be overloaded or down for maintenance
504 Gateway Timeout	No response was received from an upstream server	The gateway did not receive a response within the timeout period

Legal Notices

Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

Full Copyright Notice

Copyright (C) Open Grid Forum (2009). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.