OGF OCCI-WG Deliverables

Sam Johnston

Thijs Metsch

Andrew Edmonds

Alexis Richardson

OGF OCCI-WG Deliverables

Sam Johnston
Thijs Metsch
Andrew Edmonds
Alexis Richardson

Publication date July 2009

OCCI Use Cases

The following section describes the Use Cases which were gathered during the requirements analyses for the OCCI working group. They are used to set up the requirements and later on to verify the OCCI specification.

SLA-aware cloud infrastructure using SLA@SOI

There is a need for a standard interface for dynamic infrastructure provisioning. While doing so it must be guaranteed and verified that the infrastructure provisioning uses 'machine-readable' SLAs. (SLA_SOI)

Functional Requirements

- VM Description: request format important - In this area is where there is least coherency amongst providers.
- VM Description: a means to add non-functional constraints on functional attributes.
- VM Management: all parameters in the request should be "monitor-able" and verifiable. Full control of resources (VMs) allocated required; at a minimum: start, stop, suspend, resume.
- VM Monitoring: Monitoring non-functional constraints declared in provisioning request
- Network Management: resources assignable by network tag - defaults of public and private further sub-categorisation could be allowed e.g. tag of web could be assigned to the public network group.
- Storage Management: simple mount points, reuse storage SaaS offerings

Non-functional Requirements

- Security: Transport and user level (ACLs? oAuth?) security
- Quality of Service: Can be many - Part of service offering from the infrastructure provider e.g. Security, QoS, geo-location, isolation levels - NFPs are the basic building blocks of differentiating IaaS providers.
- Scheduling Information: When a particular resource is to be run. Also in which order should a collection of resources be ran in the case that one resource is dependent on another.

Service Manager to control the Life cycle of Services

This Use Case is based in the 'Service Manager' (SM) layer of the RESERVOIR project architecture. 'Service Providers' (SP) willing to deploy their service on the Cloud use this layer to control the service life cycle. The SM operates over the Cloud infrastructure automatically as the service demands. In a way, the SM maps the service configuration and needs to calls to the Cloud infrastructure, so many of the requirements imposed by the SM are due to the flexibility that the SM aims to provide to SPs. (RV)

Functional Requirements

- Network Management: There should be methods for the Allocation of private networks, where VMs can be attached to. A special network (e.g. 'Public Network') should be available. When some network interface is attached to it, the infrastructure must assign a public IP address.

- Image Management: There should be methods to register, upload, update and download disk images.
- VM Description: It should be possible to describe all the VM hardware components and their attributes, along with any restriction regarding the VM location:
  - Memory: Size
  - CPU: Architecture, amount of CPU's and speed.
  - Disk: Size, Interface (SCSI, IDE, SATA...), RAID (yes/no, and RAID level), Disk image to mount, Automatic backup (yes/no, backups frequency...).
  - Network: Interfaces, for each interface its bandwidth, and Network they are attached to.
  - Geographical restrictions: Location(s) where the VM can/cannot be deployed (for example for legal purposes).
  - Migration allowed (yes/no): If migration is supported by the infrastructure, this flag sets if it is allowed for the VM.
- VM Management: There should be methods to allow the SM to change the VM state (for example, from ACTIVE to SUSPENDED), if such transition is allowed by the infrastructure (i.e. is defined in the OCCI's State Machine). The description of a VM can be changed when the machine is running (ACTIVE, SUSPENDED...). But it will not be taken into account until the machine is stopped and started again, unless it is a change regarding geographical or migration restrictions. Each disk backup will have an id, as the images defined by the SM. Methods to download any backup should be provided. As each backup is, after all, a disk image, it should be possible to mount it on any VM. For example, it should be possible to stop a VM, change its configuration so its disk mounts this backup image, and restart the VM.
- Monitoring: The status (We use the term 'status' when talking about monitoring, and try not to use the term 'state' to avoid confusion with the states of the OCCI State Machine.) representation of any element is given as a list of keys and their values. For example, the status of a memory component could be given by the amount of memory used and the cache memory. Then, the keys could be: 'used' and 'cache' with the values '142MB' and '430MB'. Both the request and the reply use the corresponding element identifier. Two types of monitoring should be supported:
  - Pull based: The SM can request the status of any element it has registered: VMs, networks... Also, the SM can request the status of components, for example, the status of certain disk of a certain VM.
  - Publish/subscribe based: The SM can subscribe to be notified about events on the VMs and/or Networks. Some of the events to be notified are:
    - Errors on some component of a VM.
    - Changes on the state of a VM (e.g. from ACTIVE to SUSPENDED).
    - Periodic notifications about some element state. The frequency of this notifications can be configured in the subscription message.
- Error messages: If a VM could not be created, or a image could not be uploaded, etc... the platform should return an error message carrying a detailed description of the reason.
- Identifications: Networks, VMs and images should have unique IDs, (UUIDs, URIs, or the like). It is to be determined whether components of VMs (disks, memory...) should have an unique ID too. IDs are assigned by the Cloud infrastructure when the corresponding element is created.

Non-functional Requirements

- Both for hardware configuration and monitoring values there should be a clear, standard way to set which magnitude the value represents. For example, when setting the memory size to '2', it must be clear that we refer to GBs and not to MBs. An option would be setting the value to '2GB', another would be allowing to set both the value and the magnitude: value '2' and magnitude 'GB'.
- Protocols: The transport, message format, and state representation should use open and standard protocols, each one which strong software support (i.e. libraries and frameworks available for several programming languages).

Interoperability across Cloud Infrastructures using OpenNebula

OpenNebula is a Virtual Infrastructure Engine, being enhanced in the RESERVOIR project, which allows the management of Virtual Machines on a pool of physical resources It offers three main functionalities: backend of a public cloud, manage a virtual infrastructure in the data-center or cluster (private cloud), achieve cloud interoperation (hybrid cloud), the latter being relevant in this Use Case.

The aim of this Use Case is to state the requirements that an API for cloud providers should take into account in order to expose an interface that will enable the management of groups of Virtual Machines across them. These requirements are gathered from the experience using OpenNebula to manage Virtual Machines from different cloud providers. Currently, there are two set of plugins for OpenNebula to access Amazon EC2 and ElasticHosts cloud providers that leverage the use of both cloud providers in a transparent fashion for the end user. (ONE)

Functional Requirements

- VM Description: Virtual Machines should be described consistently across cloud providers using a slim set of indispensable attributes, such as:
  - Memory: Amount of RAM needed by the Virtual Machine
  - CPU: Number of CPUs needed by the Virtual Machine (this needs to be normalized)
  - Disk: Disks that will conform the basic filesystem and possibly others for the Virtual Machine
  - Network: How many network interface this Virtual Machine should have, and where should be attached
- VM Management: API should offer functionality to enforce operations upon Virtual Machines, such as:
  - DEPLOY: Launches the Virtual Machine
  - SHUTDOWN: Shutdown the Virtual Machine
  - CANCEL: Cancels the Virtual Machine in case of failure, or destroys it if it is running
  - CHECKPOINT: Creates a snapshot of the Virtual Machine
  - SAVE: Creates a snapshot of the Virtual Machine AND suspends it
  - RESTORE: Resumes a Virtual Machine from a previous snapshot
  - POLL: Retrieves information about Virtual Machine state and consumption attributes (percentage of Memory, CPU used, bytes transferred, and so on)

- Additionally, Virtual Machines should be in one of the following states:
  - PENDING: VM is waiting for a physical resource slot.
  - BOOTING: VM is being booted
  - RUNNING: VM is active, it should be able to start offering a service
  - SUSPENDED: VM is suspended, waiting for a resume.
  - SHUTDOWN: VM is being shutdown.
  - CANCEL: VM has been canceled by the user or by a scheduler.
  - FAILED: VM crashed or hasn't started properly.
- Network Management: API should expose functionality to
  - Create Private Virtual Networks
  - Attach Public IP to Virtual Machine
- Image Management: The ability to upload disk images is fundamental to virtual machine management to avoid the need to reinstall software for each cloud provider. The upload process should return an identifier to be used in the Virtual Machine Description.

Non-functional Requirements

- Security: Security should be handled using X509 certificates for authentication. Also, authorization can be based on said certificates and ACL lists.
- Quality of Service: When used in conjunction with Haizea, OpenNebula provides advanced reservation functionality. Cloud providers API should provide similar capabilities to ensure proper QoS.

AJAX web front-end directly calling API

This Use Case describes the ability to create web front-ends for Clouds. A cloud provider implements their customer web front-end as an entirely client-side AJAX application calling the OCCI API directly.

Functional Requirements

- Completeness: API must be contain complete set of calls to completely specify and control cloud (but this is likely only ~15-20 verbs on ~3-4 nouns!)
- Responsiveness: Calls must return swiftly. In particular, we should provide a simple and quick call to poll the _list_ of servers, drives, etc. that exist without listing all of their properties, since this is computationally much cheaper for the cloud to return, and will need to be regularly polled to catch any servers, etc. that are created outside of the interface.

Non-functional Requirements

- Syntax: A simple JSON syntax for the API will make the AJAX interface much simpler to implement

Single technical integration to support multiple service providers

Today, each cloud provider (ElasticHosts, GoGrid, Amazon, etc.) integrates independently with every other player in the cloud ecosystem (CohesiveFT, RightScale, etc), producing O(n^2) separate technical integrations. In the future, if all cloud providers and cloud ecosystem partners use a single standard API, then we have O(n) technical integrations, and all potential partnerships can immediately interoperate.

Non-functional Requirements

- Uptake: Standardized IaaS API needs strong uptake in by both cloud providers and cloud ecosystem.

Wrapping EC2 in OCCI

At the time of this writing, Amazon EC2 is popular cloud API for IaaS. Cloud providers implementing EC2 as well as other proprietary and open cloud APIs may not implement OCCI. To help ensure that the OCCI API would be capable of interfacing to EC2 though gateways, minimizing the impact to provider operations.

Functional Requirements

- Semantics: Must include the ability to fully describe core EC2 objects and operations

Non-functional Requirements

- A gateway to support the integration of OCCI and EC2

Automated Business Continuity and Disaster Recovery

Maintain a up-to-date remote shadows of physical and/or virtual machines, such that in the event of a disaster it is possible to start and switch to the remote machines.

Functional Requirements

- VM Description: Metadata mapping to legacy systems
- VM Management: Automated management in the event of a disaster (e.g. startup, IP changes).
- Network Management: Runtime alteration of IPs
- Image Management: Advanced, rsync style updates to synchronise machines with physical equivalents (e.g. rsync block devices to remote raw disk files).

Non-functional Requirements

- Quality of Service: Reservation of capacity sufficient for fail over

Simple scripting of cloud from Unix shell

An end user wishes to script a simple task (such as starting a server at midnight every night and shutting it down an hour later, automating fail over, reporting, etc.). They are using a typical Unix/Linux setup, so would like to write a simple cron job which carries this out.

Non-functional Requirements

- Syntax: This should be as simple as possible to place minimal barriers to entry on the user. The user should not need any development tools or libraries. They should be able to write 1-2 lines of shell script, posting a simple <5 lines of command data using curl, wget, etc.

Typical web hosting cluster

An end-user runs a typical web hosting cluster on a cloud, with: n database servers, m front-end web server (bursting to x under load) and a load balancer (either a specialized virtual machine or provided by the cloud like GoGrid).

Functional Requirements

- Completeness: The API should be able to fully express this cluster, which will require at least: (n +m+x) virtual machines, storage for each virtual machine, two networks (a private one connecting the machines, and the public Internet also connected to the load balancer), a fixed static IP for the website on the public Internet, possible specification of the load balancer itself.

Manage cloud resources from a centralized dashboard

An end user wishes to view and control all of his cloud-based resources in a lightweight (perhaps AJAX-based) console, perhaps the same web front-end referred to in this Use Case: AJAX web front-end directly calling API

Functional Requirements

- Completeness: Every resource provided by the cloud is discoverable by the API, and every action that can be performed on all these resources is also available via the API, together with actuators to actually perform those actions, and all the attributes of the resources are available via the API.

- Responsiveness: Calls must return swiftly. In particular, we should provide a simple and quick call to poll the _list_ of servers, drives, etc. that exist without listing all of their properties, since this is computationally much cheaper for the cloud to return, and will need to be regularly polled to catch any servers, etc. that are created outside of the interface. (text copied from AJAX web front-end directly calling API)

- Categorizability: (there's gotta be a better word...) The client must be able to identify what type each resource is in order to display like-typed resources together and in order to provide separate UI views that might be specialized for certain resource types. For example, the client must be able to differentiate between a compute resource that does not represent an actual CPU (perhaps this is a compute template) and between a compute resource that actually represents a running CPU. The interface for actually-running CPUs might display the current IP address of the instance and allow you to SSH into the instance, while a different tab in the interface might display all the compute templates and allow you to instantiate instances from them.

- Taggability: Every resource discoverable by the API must be able to be tagged by the user. This supports the oft-occurring situation where resources, though they are identified by the implementation-specific identifier, are easily identified using terminology defined by the user for his specific context. For example, one might tag resource "/compute/instanceABCDEFG" with the label "database server", and the resource "/storage/disk12345678" with the label "superSecretCorporateData".

- Searchability: The ability to request lists of resources must allow an optional filter that can specify a category or tag upon which to filter the results. This allows one to further limit their view to, for example, resources tagged "productionEnvironment", or resources of the category "storage".

Non-functional Requirements

- Usability: This should be a user interface with context-menus and context-aware links that allow the user to easily see what actions can be performed for each resource.

Compute Cloud

A cloud provider implements a RESTful API for provisioning, executing, and monitoring of tasks.

Functional Requirements

- Secure: API must be secured to ensure that only authorized identities are permitted to use the API.
- Resource: An endpoint must be created for external monitoring, status, and auditing of the task. This endpoint would be responsive to RESTful calls supporting AJAX and other clients.
- Scripted: The target system needs to understand and process directives which would be provided with the task. These directives would include the ability to pull binaries or data onto the system, run executables, and status the system resources.

Non-functional Requirements

- Single Compute Method: The resultant service should be the same service that can be used for many other purposes. It could be used for monitoring of system health, system life-cycle management, system patching, and configuration changes. If this was the only service on the system initially, it could then be used to build up the other services in a plug-in manner.

Multiple Allocation

Allocate a whole cluster with one call.

Functional Requirements

- Definition of groups: There should be a way to define groups of computers. In the example of a cluster, there would be two groups: The Headnode and a couple of Workernodes.
- Information: For configuration of the members of the defined groups, there should be way (maybe a URL) to find out about all groups and their basic configurations. In the example, the Headnode would want to know IPs or Hostnames of all Workernodes. The workernodes will need to know this, as well _and_ they need to know, that the headnode is in a different group.

Cloud Consumer Discovery of Cloud Provider's VM Input and Output Format Support

A cloud consumer would like to discover the VM input and output formats accepted and delivered by the cloud provider.

Functional Requirements

- The provider supplies an API which is availed over unsecured network connections.
- The provider supplies an API which is availed over secured network connections.
- The provider supplied API is availed for all consumer authentication and authorization levels.
- The provider supplied API identifies the supported VM input formats API uniquely and commonly across all providers.
- The provider supplied API identifies the supported VM output formats API uniquely and commonly across all providers.
- The provider supplied API identifies the supported VM formats uniquely and commonly across all providers.
- The provider API identifies mutliple supported VM input formats as a list uniquely and commonly across all providers
- The provider API identifier is unique and and consistent across all API representations.
- The provider API VM input and output format identifiers are unique and and consistent across all providers.
- The reported VM input and output formats are not required to be symetrical and equal and in consistent order.

Cloud Consumer Discovery of Cloud Provider's Dataset Input and Output Format Support

A cloud consumer would like to discover the Dataset input and output formats accepted and delivered by the cloud provider.

Functional Requirements

- The provider supplies an API which is availed over unsecured network connections.
- The provider supplies an API which is availed over secured network connections.
- The provider supplied API is availed for all consumer authentication and authorization levels.
- The provider supplied API identifies the supported Dataset input formats API uniquely and commonly across all providers.
- The provider supplied API identifies the supported Dataset output formats API uniquely and commonly across all providers.
- The provider supplied API identifies the supported Dataset formats uniquely and commonly across all providers.
- The provider API identifies multiple supported Dataset formats as a list uniquely and commonly across all providers
- The provider API identifier is unique and and consistent across all API representations.

- The provider API Dataset input and output format identifiers are unique and and consistent across all providers.
- The reported Dataset input and output formats are not required to be symetrical and equal and in consistent order.

OCCI Requirements

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
A.1.1	Attributes to define memory, CPU, disk and network requirements should be available.	2.2, 2.3, 2.6	High
A.1.2.	Attributes to define placement constraints, such as geographical location must be supported	2.2	Medium
A.1.3.	A attributes should demonstrate if migration is supported by the infrastructure	2.2	Medium
A.1.4.	The API should be able to fully express a cluster (e.g. 5 VMs, storage for each VM, two networks (a private one connecting the machines, and the public internet also connected to the load balancer), a fixed static IP for the website on the public internet)	2.9	High
A.1.5.	A means to add constraints (non-functional, functional) on attributes which are declared in a provisioning request	2.1	High
A.1.6.	Support the scheduling of resource execution. Allow provisioned resources to be execute sometime in the future from the original request	2.1	Medium
A.1.7.	Common operating systems should be supported	-	High
A.1.8.	Resources should be grouped according to provider policies	-	High
A.1.9.	Then requesting new resource(s) the request must be fully complete/describing	-	High

Table 2. Functional requirements on VM management

ID	Description	Usecases	Priority
A.2.1.	Methods to start, stop, suspend and resume VMs must be available	2.1, 2.2, 2.3, 2.5, 2.11, 2.10	High
A.2.2.	Automated management in the event of a disaster should be supported	2.1, 2.7	Low
A.2.3.	Provide IDs for each backup disk and images	2.2	High
A.2.4.	Provide methods to donwload any backup	2.2	Medium
A.2.5.	API should offer functionality to enforce the following operations: deploy, shutdown, cancel, checkpoint, save, restore, poll (could be merged with monitoring)	2.3	High
A.2.6.	The state model should include: pending, booting, running, suspended, shutdown, cancel, failed	2.3	Medium
A.2.7.	Listing collections should be possible without listing all properties for each entry	2.4	Medium

ID	Description	Usecases	Priority
A.2.8.	Allow resource representations to be updated and have those changes trigger events/changes upon VMs	-	Low
A.2.9.	Support the usage of terminal, web, desktop and automated management interfaces	2.10	Low
A.2.10.	Support the migration of resources from a physical resource to the cloud, from a cloud to another cloud and from a virtual resource to the cloud (This is a topic regarding Interoperability)	-	Medium
A.2.11.	Support a subset of all functions of today IaaS based Clouds (e.g. Amaton EC2)	2.6	Medium
A.2.12.	A common interface should be used which can be supported by many Cloud service providers (regarding Infrastructure and Data interfaces).	2.13, 2.14	Medium

Table 3. Functional requirements on Network management

ID	Description	Usecases	Priority
A.3.1.	Support the creation of VPNs	2.3	Low
A.3.2.	Support multiple network connection (Public and Private)	2.1, 2.2, 2.3	High
A.3.3.	It must be possible to attach and change IPs at runtime	2.3, 2.7	Medium
A.3.4.	Support a tagging mechanism for a group of network connections	2.1, 2.2, 2.3	Low
A.3.5.	Support network setups which allow an 'Intercloud' setup (This relates to Integration)	-	Medium

Table 4. Functional requirements on Storage management

ID	Description	Usecases	Priority
A.4.1.	Allow the usage of URIs as mount points - allows reuse of Storage service offerings	2.1	High
A.4.2.	Allow the attachment of additional storage resources at runtime	-	Medium

Table 5. Functional requirements on Image management

ID	Description	Usecases	Priority
A.5.1.	Methods which are capable to register, upload, update and download disk images must be available.	2.2	Medium
A.5.2.	Updates based on rsync commands to synchronize machines with physical equivalents should be supported	2.7	Medium
A.5.3.	When an upload completes successfully, an identifier should be returned	2.2	Low

Table 6. Identifications/References

ID	Description	Usecases	Priority
A.6.1.	Unique IDs for VM images and their components must be available	2.2, 2.13, 2.14	High

ID	Description	Usecases	Priority
A.6.2.	It must be possbile to tag resources and their components	2.10, 2.12	Medium
A.6.3.	It must be possible to search for resources based on e.g. tags.	2.10, 2.12	Medium

Table 7. Monitoring

ID	Description	Usecases	Priority
A.7.1.	Support pull-based monitoring that request the status of the elements such as network , VM ...	2.1, 2.2, 2.3	Medium
A.7.2.	Support for a publish/subscribe pattern that request events which occur in the VM or networks (such as Errors on some component, changes in the VM state and other periodic notifications)	2.2	Medium
A.7.3.	Attributes that define simple quick call to poll the list of servers, drives, etc should monitorable	2.4	Low
A.7.4.	Attributes about resource consumption of the VM from the hypervisor (CPU, memory...) should be monitorable	2.1, 2.2	Medium
A.7.5.	Management reports should be generated from in some of the following formats XML, PDF	-	Low

Non-functional Requirements

This section deals with all the non-funtional requirements.

Table 8. Security requirements

ID	Description	Usecases	Priority
B.1.1.	Support the usage of X509 Certificates	2.3, 2.13, 2.14	High
B.1.2.	Support the usage of ACLs	B.1, 2.1	High
B.1.3.	Attributes to define Security levels should be available in the descriptions	2.1	High
B.1.4.	Transport and user level security should be given	2.1, 2.13, 2.14	High
B.1.5.	Allow geographical region to be specified	B.4	High

Table 9. Quality of Service

ID	Description	Usecases	Priority
B.2.1.	Support capacities requirements for recovery / failover cases	2.7	Low
B.2.2.	Support of attributes in the VM description to define QoS level (this also includes the reponse times)	2.1	High
B.2.3.	Support of attributes in the VM describing the Isolation level	2.1	Medium
B.2.4.	Support of attributes for an advanced reservation functionality	2.3	Low
B.2.5.	Allow VM response times to be specified	B.4	High

Table 10. Syntax

ID	Description	Usecases	Priority
B.3.1.	No development tools or libraries should be needed by the end-user	2.8	Medium
B.3.2.	Support simple JSON syntax to suppot Ajax interface	2.4, 2.10	Medium
B.3.3.	Clear definition of units (MB, GB etc) should be used in the requests (Like those defined by IEC 60027-2 A.2)	A.2, 2.4	Medium

Table 11. Backup/Disaster recovery

ID	Description	Usecases	Priority
B.4.1.	Support a backup functionality of cloud resources	-	Low
B.4.2.	The interface should reconsider failover, disaster recovery and business continuity plans	-	Medium

OCCI Walkthrough

Overview

THIS WALKTHROUGH DOCUMENT DOES NOT REFLECT THE CURRENT STATE OF THE SPECIFICATION. IT WILL BE UPDATED ON PUBLICATION.

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 Frequently Asked Questions

General

Who created the Open Cloud Computing Interface (OCCI)?	The Open Grid Forum (OGF)'s Open Cloud Computing Interface Working Group (OCCI-WG) created the Open Cloud Computing Interface (OCCI).
Who are the Open Cloud Computing Interface Working Group (OCCI-WG) officials?	Andrew Edmonds (Intel, SLA@SOI), Thijs Metsch (Sun Microsystems) and Alexis Richardson (Rabbit Technologies Ltd) are the chairs, and Sam Johnston (Australian Online Solutions) is the secretary.
Who else was involved?	Around 200 individuals representing over 100 companies were involved in the development of the Open Cloud Computing Interface (OCCI).

Use Cases

How were the use cases collected?	Use cases were solicited from the working group mailing list as well as other external sources.

Technical

Why didn't you invent your own XML representation?	See Tim Bray's Don’t Invent XML Languages post.

OCCI Core Specification

Introduction

The Open Cloud Computing Interface (OCCI) is an open protocol for all cloud computing services. A RESTful interface, it deviates from the underlying HyperText Transfer Protocol (HTTP) only where absolutely necessary and can be described as a "Resource Oriented Architecture (ROA)".RWS Unlike other envelope-based protocols which operate in-band, all existing HTTP features are available for caching, proxying, gatewaying and other advanced functionality such as partial GETs.

Each resource is identified by URL(s) and has one or more representations which may include a hypertext (e.g. XHTML5) rendering for direct end-user accessibility. As such OCCI can present both a machine interface (using native resource renderings) and a user interface (using HTML markup with forms and other web technologies such as Javascript/Ajax). HTTP content negotiation is used to select between alternative representations and metadata including associations between resources is exposed via HTTP headers (e.g. the `Link:` and `Category:` headers).

In this way OCCI is not responsible for the representations themselves, rather it enables users to organise and group resources together to build arbitrarily complex systems of inter-related resources. It relies on existing standards for rendering and does not make any recommendations of one standard format over any other.

Tip

This is the case for the World Wide Web today where many image, video and other supporting formats co-exist. Browsers support a number of the common formats and users choose the most appropriate for the task.

Example

```
> GET /us-east/webapp/vm01 HTTP/1.1
> User-Agent: occi-client/1.0 (linux) libcurl/7.19.4 OCCI/1.0
> Host: cloud.example.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 10 Oct 2009 12:56:51 GMT
< Content-Type: application/ovf
< Link: </us-east/webapp/vm01;start>;
<       rel="http://purl.org/occi/action/start";
<       title="Start"
< Link: </us-east/webapp/build.pdf>;
<       rel="related";
<       title="Documentation";
<       type="application/pdf"
< Category: compute;
<       label="Compute Resource";
<       scheme="http://purl.org/occi/kind/"
< Server: occi-server/1.0 (linux) OCCI/1.0
< Connection: close
<
< <?xml version="1.0" encoding="UTF-8"?>
< <Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<           xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
<           xmlns="http://schemas.dmtf.org/ovf/envelope/1"
<           xml:lang="en-US">
...
```

Basics

Entry point

The interface is defined by a single URL entry point which will either be a *collection*, contain *link*(s) to *collection*(s) or both.

Kinds, Actions & Attributes

An interface exposes "kinds" which have "attributes" and on which "actions" can be performed. The attributes are exposed as key-value pairs and applicable actions as links, following the REST hypertext constraint (whereby state transitions are defined *in-band* rather than via rules).

HTTP Verbs

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. WebDAV definitions are used for MKCOL, MOVE and COPY.

POST (Create)	“The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.”RFC2616 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 - Metadata and Entity)	“The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.”RFC2616 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 (Create or Update)	“The PUT method requests that the enclosed entity be stored under the supplied Request-URI.”RFC2616 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)	“The DELETE method requests that the origin server delete the resource identified by the Request-URI.”RFC2616 DELETE results in the deletion of the resource (and everything "under" it, as appropriate).

Additionally the following HTTP methods are used:

COPY (Duplicate)	“The COPY method creates a duplicate of the source resource identified by the Request-URI, in the destination resource identified by the URI in the Destination header.”RFC4918
HEAD (Retrieve - Metadata Only)	“The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.”RFC2616
MKCOL (Make Collection)	“MKCOL creates a new collection resource at the location specified by the Request-URI.”RFC4918
MOVE (Relocate)	“The MOVE operation on a non-collection resource is the logical equivalent of a copy (COPY), followed by consistency maintenance processing, followed by a delete of the source, where all three actions are performed in a single operation.”RFC4918
OPTIONS	“The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI.”RFC2616

Connection

Authentication

Servers *may* require that requests be authenticated using standard HTTP-based authentication mechanisms (including OAuth).OAuth They indicate this requirement by returning `HTTP 401` with a `WWW-Authenticate` header and a suitable challenge (e.g. `Basic`, `Digest`, `OAuth`). The client then includes appropriate `Authorization` headers in its responses.RFC2617

Servers *may* set and clients *may* accept *cookies* in order to maintain authentication state between requests. Such sessions *should not* be used for other purposes (such as server-side state) in line with RESTful principles.RFC2109

Versioning

Servers and clients *should* indicate the latest version of OCCI they support (e.g. `1.0`) by way of the `Server:` and `User-Agent:` headers respectively, using the token “`OCCI`” (e.g. “`OCCI/1.0`”). If none is provided the latest available version *shall* be used.

Model

The model defines the objects themselves without regard to how they interrelate.

Kinds

Each category of resources distinguished by some common characteristic or quality is called a *kind* (e.g. `compute`, `network`, `storage`, `queue`, `application`, `contact`).

Kinds defined by this standard live in the `http://purl.org/occi/kind/` namespace but anyone can define a new kind by allocating a URI they control.

Warning

Defining your own kinds can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

Each resource *must* specify a kind by way of a *category* within the *scheme* “`http://purl.org/occi/kind/`”.

Tip

The word *type* is not used in this context in order to avoid confusion with Internet media types.

Attributes

An *attribute* is a specification that defines a property of an object. It is expressed in the form of key-value pairs. Attributes are divided into namespaces which are separated by the dot character (“.”).

Tip

This scalable approach was derived from the Mozilla Firefox `about:config` page.

Attributes defined by this standard reside under the `occi` namespace (e.g. "`occi.abc`") but anyone can define a new attribute by allocating a unique namespace based on their reversed Internet domain (e.g. “`com.cisco.cdp`”).

Warning

Defining your own attributes can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

Registry Entries

Table 1. Core Attributes

Attribute	Description	Type	Example
`occi.id`	Immutable identifier for the resource	URI	`urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770`
`occi.title`	Display name for the resource	String	`Compute Resource #123`
`occi.summary`	Description of the resource	String	`A virtual compute resource`
`occi.version`	Specification version	Float	`1.0`

Actions

An *action* is some process that can be carried out on one or more *resource*s.

Each available *action* for a given *resource* is indicated via a *link* with the `action` class.

```
Link: </us-east/webapp/vm01;start>;
      rel="http://purl.org/occi/action/start";
      title="Start"
```

Actions defined by this standard reside under the `http://purl.org/occi/action/` namespace but anyone can define a new action by allocating a URI they control.

Warning

Defining your own actions can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

An *action* is triggered via an HTTP POST and depending on the action requested (e.g. `resize`), parameters *may* be provided using HTML forms (e.g. `application/x-www-form-encoded`). In the case of HTML-based renderings the actions can therefore be actual HTML forms.

Tip

Some resources can be interacted with but not rendered due to the nature of the resource or prevailing security policies (for example, an operator may be able to backup a machine without knowing anything about it).

Asynchronous Actions

Synchronous actions *may* return `200 OK` on successful completion or `201 Created` with a `Location:` header indicating a new resource for audit purposes.

Tip

Assume that clients are paranoid and want audit trails for all but the most trivial of actions.

In the event that the *action* does not complete immediately it *should* return `HTTP 202 Accepted` and a `Location:` header indicating a new resource where status and other pertinent information can be obtained.

Tip

Don't keep clients waiting - if you're not sure to return immediately then give them a resource they can monitor.

Advanced Actions

The specific parameters required and allowable values for them depend on the action and for advanced actions *may* require sending of custom *content type*s rather than `application/x-www-form-encoded`.

Meta-model

The meta-model defines how objects interrelate.

Categories

Category information allows for flexible organisation of resources into one or more vocabularies (each of which is referred to as a *scheme*).

The meta-model was derived from Atom, consisting of three attributes:

term	The term itself (e.g. “`compute`”)
scheme (optional)	The vocabulary (e.g. “`http://purl.org/occi/kind/`”)
label (optional)	A human-friendly display name for the term (e.g. “`Compute Resource`”)

Category schemes and/or terms defined by this standard reside throughout the `http://purl.org/occi/` namespace but anyone can define a new scheme by allocating a URI they control.

Tip

Categories provide a flexible way to manage resources by taxonomy (categories) and/or folksonomy (tags), where both can be shared between [groups of] users or globally. For example, users can create schemes for resource locations (e.g. `US-East`, `US-West`, `Europe`), operating systems (e.g. `Windows`, `Linux`) and patch levels (e.g.

Examples

```
Category: compute;
      label="Compute Resource";
      scheme="http://purl.org/occi/kind/"
```

Querying

TODO: Pull query interface from GData: http://code.google.com/apis/gdata/docs/2.0/reference.html#Queries

Registry Entries

Table 2. Core Category Schemes

Scheme	**Description**
`http://purl.org/occi/kind/`	OCCI Kinds

Collections

Where an operation could return multiple resources (e.g. categories, searches) this is referred to as a *collection*. Collections are returned as a list of URLs in `text/uri-list` format.RFC2483

Tip

Collections are passed by reference for simplicity rather than performance reasons, requiring O(n+1) requests. Including metadata (requiring a wrapper format like Atom or SOAP) and/or the data itself would provide O(1) performance, though passing by value should only be considered where the representations are known to be small as such encodings add significant overhead.

Examples

```
# OCCI Example Collection
/examples/custom-extension
/examples/lamp-multi-vm
/examples/lamp
/examples/myservice
```

Advertising

Any given URL can be a collection and/or advertise *link*s to other *collection*s using the `collection` class:

```
Link: <http://example.com/123/audit>;
      rel="http://purl.org/occi/collection/audit";
      title="Audit Entries"
```

Tip

The root (“/”) *should* expose collections *in-band* and/or *out-of-band* in order for clients to discover resources.

Paging

Collections *may* be divided into *page*s, with each linking to the “first”, “last”, “next” and “previous” *link relation*s.

```
Link: <http://example.com/xyz;start=0>; rel="first"
Link: <http://example.com/xyz;start=400>; rel="previous"
Link: <http://example.com/xyz;start=500>; rel="self"
Link: <http://example.com/xyz;start=600>; rel="next"
Link: <http://example.com/xyz;start=900>; rel="last"
```

Linking

Web linking standards for HTTP [LINK] and HTML [HTML5] are used to indicate associations between resources. All formats *must* support *in-band* linking including:

- Link relations (e.g. `rel="alternate"`)
- Pointers to resources (e.g. `href="http://example.com/"`)
- Internet media types (e.g. `type="text/html"`)
- Extensibility (e.g. `attribute="value"`)

```
Link: </us-east/webapp/build.pdf>;
      rel="related";
      title="Documentation";
      type="application/pdf"
```

*Link relation*s defined by this standard reside under the `http://purl.org/occi/rel` namespace but anyone can define a new *link relation* by allocating a URI they control.

Registry Entries

Table 3. Core Link Relations

Relation	**Description**
`collection` (`http://purl.org/occi/rel#collection`)	A related collection whereby: • The root of the collection is indicated by the `href` attribute. • The *kind* of the collection is indicated by the `kind` extended attribute.
`first`	“An IRI that refers to the furthest preceding resource in a series of resources.” [LINK]
`help`	“The referenced document provides further help information for the page as a whole.” [HTML5]
`icon`	“The specified resource is an icon representing the page or site, and should be used by the user agent when representing the page in the user interface.” [HTML5]
`last`	“An IRI that refers to the furthest following resource in a series of resources.” [LINK]
`next`	“A URI that refers to the immediately following document in a series of documents.” [LINK]

Relation	**Description**
`previous`	“A URI that refers to the immediately preceding document in a series of documents.” [LINK]
`search`	“The referenced document provides an interface specifically for searching the document and its related resources.” [HTML5, OpenSearch]
`self`	“Identifies a resource equivalent to the containing element” [RFC4287]

Extensibility

The interface is fully extensible, both via a public peer review process (in order to update the specification itself, usually via registries) and via independent allocation of unique namespaces (in order to cater for vendor-specific enhancements).

Foreign markup

Implementations *must* accept and forward but otherwise ignore markup they do not understand.

Security Considerations

Encryption is not required by the specification in order to cater for sites that do not or can not use it (e.g. due to export restrictions, performance reasons, etc.), however SSL/TLS *should* be used over public networks including the Internet.

Glossary

in-band	“Sending of metadata and control information in the same band, on the same channel, as used for data”, for example, by embedding it in HTML. [`http://en.wikipedia.org/wiki/In-band`]
kind	“A category of things distinguished by some common characteristic or quality”, for example events, messages, media. [`http://wordnetweb.princeton.edu/perl/webwn?s=kind`]
out-of-band	“Communications which occur outside of a previously established communications method or channel”, for example, in HTTP headers. [`http://en.wikipedia.org/wiki/Out-of-band_signaling`]
type	Internet media (MIME) type.

Bibliography

Normative References

[RFC2109] *RFC 2109 - HTTP State Management Mechanism*. `http://tools.ietf.org/html/rfc2109`. Internet Engineering Task Force (IETF) 1997-02.

[RFC2483] *RFC 2483 - URI Resolution Services Necessary for URN Resolution*. `http://tools.ietf.org/html/rfc2483#section-5` [`http://tools.ietf.org/html/rfc2109`]. Internet Engineering Task Force (IETF) 1999-01.

[RFC2616] *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1*. `http://tools.ietf.org/html/rfc2616`. Internet Engineering Task Force (IETF) 1999-06.

[RFC2617] *RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication*. `http://tools.ietf.org/html/rfc2617 [http://tools.ietf.org/html/rfc2616]`. Internet Engineering Task Force (IETF) 1999-06.

[RFC3339] *RFC 3339 - Date and Time on the Internet: Timestamps*. `http://tools.ietf.org/html/rfc3339`. Internet Engineering Task Force (IETF) 2002-07.

[RFC4918] *RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. `http://tools.ietf.org/html/rfc4918 [http://tools.ietf.org/html/rfc2616]`. Internet Engineering Task Force (IETF) 2007-06.

[OpenSearch] *OpenSearch 1.1*. `http://www.opensearch.org/Specifications/OpenSearch/1.1 [http://tools.ietf.org/html/rfc2616]`. A9.com, Inc. (an Amazon company) Clinton DeWitt. Joel Tesler. Michael Fagan. Joe Gregorio. Aaron Sauve. James Snell. 2009.

Informative References

[RFC4287] *RFC 4287 - The Atom Syndication Format*. `http://tools.ietf.org/html/rfc4287`. Robert Syre. Mark Nottingham. Internet Engineering Task Force (IETF) 2005-12.

[HTML5] *HTML 5*. `http://www.w3.org/TR/html5/ [http://tools.ietf.org/html/rfc4287]`. Ian Hickson. David Hyatt. World Wide Web Consortium (W3C) 2009-08-25.

[OAuth] *OAuth*. `http://oauth.net/core/1.0 [http://tools.ietf.org/html/rfc2616]`. OAuth Core Workgroup `<spec@oauth.net>`. 2007-12-04.

[RWS] *RESTful Web Services*. `http://oreilly.com/catalog/9780596529260/`. 9780596529260. O'Reilly Media Leonard Richardson. Sam Ruby. 2007-05.

[LINK] *Web Linking*. `http://tools.ietf.org/html/draft-nottingham-http-link-header`. Internet Engineering Task Force (IETF) Mark Nottingham. 2009-07-12.

[CATEGORY] *Web Categories*. `http://tools.ietf.org/html/draft-johnston-http-category-header`. Internet Engineering Task Force (IETF) Sam Johnston. 2009-07-1.

OCCI Machine Interface (HTTP)

2009-10-07

Specification

The HTTP binding for OCCI provides a machine interface, delivering resources in their native formats:

- The HTTP binding is defined by RFC2616 (HTTP).
- Web Linking [LINK] and Web Categories [CATEGORY] specifications are used for the meta-model.
- Server-side cookies ("Attributes") are used for name-value pairs.
- Collections are transferred as the `text/uri-list` content type.RFC2483

In all cases the same process as for Cookies (defined in RFC2965) is used to set/get headers, where `[Set-]Attribute:`, `[Set-]Category:` and `[Set-]Link:` are used in place of `Cookie:` and `Set-Cookie:`. `Set-*` headers may also be included on PUT or POST requests (including empty POSTs in order to update the metadata independently of the representation).

Existing values can be discarded by sending a `Set-*` header with the `discard` attribute and updated atomically by providing a new value in the same HTTP transaction.

Example

POST Request

```
POST /compute/123 HTTP/1.1
Host: example.com
Content-Length: 0
Set-Attribute: id="urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770"
Set-Category: compute;
  scheme="http://purl.org/occi/kind/";
  label="Compute Resource"
Set-Link: <http://example.com/products/1234>;
  rel="alternate";
  title="Alternate representation"
```

GET Response

```
Attribute: id="urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770"
Attribute: title="Compute Resource #123"
Attribute: summary="A virtual compute resource"
Attribute: updated="2009-12-31T12:59:59Z"
Attribute: compute.cores=2
Attribute: compute.speed=3000
Attribute: compute.memory=2048
ETag: "dad86c61eea237932f"
Category: compute;
  scheme="http://purl.org/occi/kind/";
  label="Compute Resource"
Link: <http://example.com/products/1234>;
  rel="alternate";
  title="Alternate representation"

<?xml version="1.0" encoding="UTF-8"?>
<ovf:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ovf="http://schemas.dmtf.org/ovf/1/envelope"
<!-- snip -->
```

Bibliography

Normative References

[RFC2483] *RFC 2483 - URI Resolution Services Necessary for URN Resolution.* `http://tools.ietf.org/html/rfc2483#section-5` [`http://tools.ietf.org/html/rfc2109`]. Internet Engineering Task Force (IETF) 1999-01.

[RFC2616] *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.* `http://tools.ietf.org/html/rfc2616`. Internet Engineering Task Force (IETF) 1999-06.

[RFC2965] *RFC 2965 - HTTP State Management Mechanism.* `http://tools.ietf.org/html/rfc2965` [`http://tools.ietf.org/html/rfc2822`]. Internet Engineering Task Force (IETF) 2000-10.

Informative References

[CATEGORY] *Web Categories.* `http://tools.ietf.org/html/draft-johnston-http-category-header`. Internet Engineering Task Force (IETF) Sam Johnston. 2009-07-1.

[LINK] *Web Linking.* `http://tools.ietf.org/html/draft-nottingham-http-link-header`. Internet Engineering Task Force (IETF) Mark Nottingham. 2009-07-12.

[HTML5-article] *Designing a great HTTP API - why heavyweight XML is not the answer.* `http://www.elastichosts.com/blog/2009/01/01/designing-a-great-http-api/` [`http://www.smashingmagazine.com/2009/07/29/misunderstanding-markup-xhtml-2-comic-strip/`]. . 2009-01-01.

OCCI Infrastructure

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

Table 1. Common Attributes

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

Kinds

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

Table 2. Kinds

Kind	URI	Description
`compute`	`http://purl.org/occi/kind/compute`	Information processing resources
`network`	`http://purl.org/occi/kind/network`	Interconnection resources
`storage`	`http://purl.org/occi/kind/storage`	Recorded information resources

Compute

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

Attributes

The following attributes apply to this kind:

Table 3. Compute Attributes

Attribute	Type	Description
`occi.compute.architecture`	Enum (x86, x64)	CPU Architecture (e.g. x64)
`occi.compute.cores`	Integer	Number of CPU cores (e.g. 1, 2)
`occi.compute.speed`	Float (10^9 Hertz)	Clock speed in gigahertz (e.g. 2.4)
`occi.compute.memory`	Float (10^6 bytes)	RAM in megabytes (e.g. 8192)
`occi.compute.memory.speed`	Float (10^9 bytes/second)	RAM speed in Gbit/s (e.g. `17` for PC-8500 DDR3 per Wikipedia)
`occi.compute.memory.reliability`	Enum (standard, checksum)	Qualitative measure of RAM reliability (e.g. ECC)
`occi.compute.status`	Enum (active, inactive, standby)	Status of the compute resource

Representations

The following representation formats are identified (but not necessarily recommended) as possible candidates for this kind:

Table 4. Compute Representations

Name	Type	Specification
Citrix Xen Virtual Appliance (XVA) [deprecated]	application/octet-stream	VHD
Open Virtualisation Appliance (OVA)	application/ova+xml	OVF
Open Virtualisation Format (OVF)	application/ovf+xml	OVF

Network

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

Attributes

The following attributes apply to this kind:

Table 5. Network Attributes

Attribute	Type	Description
`occi.network.vlan`	Integer (0..4095)	802.1q VLAN ID (e.g. `4095`)
`occi.network.label`	Token	Tag based VLANs (e.g. `external-dmz`)
`occi.network.address`	IPv4 or IPv6 Address (in CIDR notation)	Internet Protocol (IP) network address (e.g. `192.168.0.0/24`, `fc00::/7`)
`occi.network.gateway`	IPv4 or IPv6 Address (in CIDR notation)	Internet Protocol (IP) network address (e.g. `192.168.0.0/24`, `fc00::1/64`)
`occi.network.allocation`	Enum (`auto`, `dhcp`, `manual`)	Address allocation mechanism: • `auto` is handled automatically by infrastructure and/or guest agent • `dhcp` uses network-based allocation protocol(s) • `manual` requires preconfiguration or manual allocation

Representations

The following representation formats are identified (but not necessarily recommended) as possible candidates for this kind:

Table 6. Network Representations

Name	Type	Specification
N/A	N/A	N/A

Storage

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

Attributes

The following attributes apply to this kind:

Table 7. Storage Attributes

Attribute	Type	Description
`occi.storage.reliability`	Enum (transient, persistent, reliable)	Qualitative device persistence (e.g. transient)
`occi.storage.size`	Integer (10^9 bytes)	Drive size in gigabytes (e.g. `40`, `0.00144`)
`occi.storage.speed`	Integer (10^6 bytes/second)	Drive speed in MB/s (e.g. `600` for SAS/SATA-600 Wikipedia)
`occi.storage.status`	Enum (online, offline, standby, degraded)	Currenty status of the storage resource

Representations

The following representation formats are identified (but not necessarily recommended) as possible candidates for this kind:

Table 8. Storage Representations

Name	Type	Specification
ISO image format	application/x-iso9660-image	ISO
Microsoft Virtual Hard Disk	application/octet-stream	VHD
QEMU qcow2 image format	application/x-qemu-qcow2	QCOW2
Raw disk format (IMA, IMG, RAW, etc.)	application/octet-stream	RAW
Virtualbox Virtual Disk Image [undocumented]	application/octet-stream	VDI
VMware Virtual Disk Format	application/octet-stream	VMDK

Extensions

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

Bibliography

Normative References

Informative References

[VHD] *CTX121652 Overview of the Open Virtualisation Format.* `http://support.citrix.com/article/CTX121652` `[http://en.wikipedia.org/wiki/List_of_device_bandwidths]`..

[RAW] *Wikipedia - Disk Image.* `http://en.wikipedia.org/wiki/Disk_image` [`http://en.wikipedia.org/wiki/List_of_device_bandwidths`]. .

[OVF] *DSP0243 Open Virtualisation Format (OVF).* `http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf` [`http://en.wikipedia.org/wiki/List_of_device_bandwidths`]. .

[ISO] *Wikipedia - ISO 9660.* `http://en.wikipedia.org/wiki/ISO_9660` [`http://en.wikipedia.org/wiki/List_of_device_bandwidths`]. .

[QCOW2] *QCOW2 Image Format.* `http://www.gnome.org/~markmc/qcow-image-format.html` [`http://en.wikipedia.org/wiki/List_of_device_bandwidths`]. .

[VMDK] *VMware Virtual Disk Format.* `http://www.vmware.com/app/vmdk/?src=vmdk`. .

[VDI] *Virtualbox Source Code - Virtual Disk Image (VDI).* `http://www.virtualbox.org/svn/vbox/trunk/src/VBox/Devices/Storage/VDICore.h`. .

[VHD] *Microsoft Virtual Hard Disk (VHD) Image Format Specification.* `http://technet.microsoft.com/en-us/virtualserver/bb676673.aspx` [`http://en.wikipedia.org/wiki/List_of_device_bandwidths`]. .

[Wikipedia] *Wikipedia: List of device bandwidths.* `http://en.wikipedia.org/wiki/List_of_device_bandwidths`. .

OCCI Search Extension

2009-09-15

Overview

OpenSearch is used to advertise the search interface:

“Search clients can use OpenSearch description documents to learn about the public interface of a search engine. These description documents contain parameterized URL templates that indicate how the search client should make search requests. Search engines can use the OpenSearch response elements to add search metadata to results in a variety of content formats.”???

Bibliography

Normative References

Informative References

OCCI Status Extension

2009-09-15

Status

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

Table 1. 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)

Bibliography

Normative References

Informative References

OCCI Tasks Extension

2009-09-15

Overview

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 1. 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`")

Bibliography

Normative References

Informative References

OCCI Registries

Extensibility of OCCI relies heavily on registries, which allow the working group to initially specify the minimum required functionatliy based on existing implementations while also catering for future evolution.

Implementors who wish to extend the specification can do so by way of a process TBD in which their request is assessed within a fixed time frame of 7-30 days. If the request is found to be useful for interoperability then it will be accepted and entered into the registry.

If rejected the implementor can still use custom extensions which will be preserved but otherwise ignored by other implementations which do not understand them.

Tip

This document seeds the registries which are to be hosted in an editable medium such as a database, spreadsheet or wiki.

Core

Common Attributes

Table 1. Common Attributes

Attribute	Description	Example
`occi.id`	Immutable identifier for the resource	`urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770`
`occi.title`	Display name for the resource	`Compute Resource #123`
`occi.summary`	Description of the resource	`A virtual compute resource`
`occi.version`	Specification version	`1.0`

Link Relations

Table 2. Link Relations

Relation	Description
`category` (`http://purl.org/occi/rel#category`)	A category mapping whereby: • The `scheme` is required and indicated by the `href` attribute. • The `label` is optional and indicated by the `title` attribute. • The `term` is required and indicated by the `term` extended attribute.
`collection` (`http://purl.org/occi/rel#collection`)	A related collection whereby: • The root of the collection is indicated by the `href` attribute.

Relation	**Description**
	• The *kind* of the collection is indicated by the `kind` extended attribute.
`first`	“An IRI that refers to the furthest preceding resource in a series of resources.” [???]
`help`	“The referenced document provides further help information for the page as a whole.” [???]
`icon`	“The specified resource is an icon representing the page or site, and should be used by the user agent when representing the page in the user interface.” [???]
`last`	“An IRI that refers to the furthest following resource in a series of resources.” [???]
`next`	“A URI that refers to the immediately following document in a series of documents.” [???]
`previous`	“A URI that refers to the immediately preceding document in a series of documents.” [???]
`search`	“The referenced document provides an interface specifically for searching the document and its related resources.” [???, ???]
`self`	“Identifies a resource equivalent to the containing element” [???]

OCCI Legal Notices

Intellectual Property Statement

Disclaimer

Full Copyright Notice