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.