Apache Deltacloud

David Lutterkort {lutter@redhat.com}
Marios Andreou {marios@redhat.com}
Michal Fojtik {mfojtik@redhat.com}

August 12, 2010

Contents

List of Figures

1 Introduction

Apache Deltacloud is a REST-based, cloud abstraction API. Deltacloud makes it possible to manage resources in different IaaS clouds using a single REST-based API. A series of back-end drivers 'speaks' each cloud provider's native API and the Deltacloud Core Framework provides the basis for implementing drivers as needed for other/new IaaS cloud providers. Apache Deltacloud currently supports: Amazon EC3 and S3, Rackspace Cloud Servers and Cloud Files, Gogrid Cloud Servers, Terremark Vcloud Express, Rimuhosting VPS, Red Hat Enterprise Virtualisation (rhev-m) and Opennebula.
The Apache Deltacloud project aims at addressing two issues:

1. Avoiding lockin by providing an API abstraction that can be implemented as a wrapper around a large number of clouds, freeing users of cloud from dealing with the particulars of each cloud's API.

2. Providing a basis for open-source evolution of cloud API's.

1.1 Concepts

The following terms are used in the Apache Deltacloud API and are introduced here to aid the reader. Each represents an entity in the 'back-end' provider cloud such as a running virtual server or a server image. It should be noted that not all clouds support all of the following entities. Only the appropriate entity collections are exposed for a given back-end driver.

Realms: A distinct organizational unit within the back-end cloud such as a datacenter. A realm may but does not necessarily represent the geographical location of the compute resources being accessed.

Instances: A realized virtual server, running in a given back-end cloud. These are instantiated from server Images.

Images: These are templates (virtual machine images) from which Instances are created. Each Image defines the root partition and initial storage for the Instance operating system.

Instance_States: These represent the Instance lifecycle; at any time an Instance will be in one of *start, pending, running, stopped, shutting_down,*

finished.

Keys: These represent credentials used to access a running Instance.

Storage_Volume: This is a virtual storage device that can be attached to an Instance and mounted by the OS therein.

Storage_Snapshot: These are copies, snapshots of a Storage_Volume at a specified point in time.

Blob_Storage: Generic '*key* ==> *value*' based data store (such as Rackspace CloudFiles or Amazon S3). This part of the API is currently under development.

1.2 Client requests

In keeping with `REST`, clients make requests through `HTTP`, with the usual meanings assigned to the standard `HTTP` verbs `GET`, `POST`, `PUT`, and `DELETE`. Beyond the generally accepted `REST` design principles, Apache Deltacloud follows the guidelines discussed in [1].

The URL space of the API is structured into collections of resources (entities, objects). The top level entities used in the Deltacloud API are: `Realms, Images, Instance_States, Instances, Keys, Storage_Volume, Storage_Snapshots, Blob_Storage`.

1.2.1 Authentication

The Deltacloud API server is stateless, and does not keep any information about the current client. In particular, it does not store the credentials for the backend cloud it is talking to. Instead, it uses HTTP basic authentication, and clients have to send the username/password for the backend cloud on every request.

The specifics of what needs to be sent varies from cloud to cloud; some cloud providers employ a username and password for API access, whilst others use special-purpose API keys.

[1] `http://fedoraproject.org/wiki/Cloud_APIs_REST_Style_Guide`

1.3 Server responses

The server can respond to client requests in a variety of formats. The appropriate response format is determined by HTTP content negotiation. The primary format is `XML`, which is the basis for this document. Output is also available as `JSON` and, mostly for testing, as `HTML`.

In general, list operations, such as `GET /api/realms` will only provide a brief list of the objects of this resource type; full details can be retrieved by making a request `GET /api/realms/:id` to the URL of the individual realm.

1.4 API conventions

Any XML element that represents an object, such as an instance has an `href` and a `id` attribute. The `href` provides the URL at which object-specific actions can be performed (e.g., a `GET` to the URL will give details of the object). The `id` provides an identifier of the object and this is unique within its collection (i.e., unique `id` for each Instance, Image, Realm etc).

Generally, objects also have a human-readable name; the name is provided in a `<name/>` child element of the object's container tag.

1.5 API stability and evolution

Future changes to the API will be made in a manner that allows old clients to work against newer versions of the the API server.

1.6 Online documentation

Automatically generated documentation can be accessed on every server running the Deltacloud Core API service through the URL `http://localhost:3001/api/docs/`. The documentation is both available in HTML and XML, though the XML format is not part of this specification, and may change in an incompatible way.

2 The API entry point

Any part of the official API can be reached through the main entry point, by default `http://localhost:3001/api`. The entry point list the resources the server knows about. Currently, these are:

- Instances
- Instance states
- Images
- Realms
- Hardware profiles
- Keys
- Blob storage (in development)
- Storage volumes and snapshots (in development)

```
<api driver="rackspace" version="0.1">
      ...
  <link href="http://localhost:3001/api/instance states"
  rel="instance states">
  <link href="http://localhost:3001/api/instances"
  rel="instances">
  <link href="http://localhost:3001/api/images" rel="images">
  <link href="http://localhost:3001/api/realms" rel="realms">
  <link href="http://localhost:3001/api/hardware profiles"
  rel="hardware profiles"
      ...
</api>
```

Figure 1: The API entry point returns an XML `<link>` element for each resource collection

Specific implementations for the Apache Deltacloud API may not support all resource types defined by this API. For example, a Deltacloud instance pointing at a storage-only service will not expose compute resources like instances and hardware profiles.

```
<api driver="mock" version="0.1">
      ...
    <link href="http://localhost:3001/api/instances"
      rel="instances">
      <feature name="user\_name"/>
    </link>
      ...
 </api>
```

Figure 2: Features are advertised at the API entry point

2.1 Features

The Apache Deltacloud API defines the standard behavior and semantics for each of the resource types as a baseline for any API implementation; it is often desirable to enhance standard API behavior with specific features. The API also defines all the features that can be supported by an API implementation - each of them has a fixed, predefined meaning. As an example, the feature `user-name` indicates that a user-specified name can be assigned to an instance when it is created. Features are advertised in the top-level entry point as illustrated by Figure 2.

3 Compute resources and other toplevel entities

The compute resources are *instances, instance states, images, realms*, and *hardware profiles*. Not strictly a compute resource, we deal with *keys* here, too, since they generally are used to access instances (e.g., via `ssh`).

3.1 Realms

A *realm* represents a distinct unit within the same cloud, such as a data center. The exact definition of a *realm* is left to the cloud provider. Generally speaking, going from one realm to another within the same cloud may change many aspects of the cloud, such as SLA's, pricing terms, etc.

3.1.1 `GET /api/realms`

List all realms. Can be filtered by adding a request parameter `architecture` to the realms that support a specific architecture such as `i386`.

3.1.2 GET /api/realms/:id

Provide the details of a `realm`. Currently, these are a `name` and a `state`. The `name` is an arbitrary label with no specific meaning in the API. The `state` can be either `AVAILABLE` or `UNAVAILABLE`, as shown in Figure 3.

```
<realm href="http://localhost:3001/api/realms/us"
   id="us">
   <name>United States</name>
   <state>AVAILABLE</state>
</realm>
```

Figure 3: XML output showing attributes of a `realm`

3.2 Hardware profiles

A *hardware profile* describes the sizing of a virtual machine in a cloud and prescribes details such as how many virtual CPU's, how much memory or how much local storage an instance might have.

Since clouds differ sharply in how virtual machine sizing is represented and influenced, *hardware profile*s provide a generic mechanism to express sizing constraints. For each dimension (amount of memory etc.), the hardware profile can express one of the following:

1. Size is *fixed* in this dimension, e.g. instances all have 2GB of memory, *or*
2. Size can be varied freely within some *range*, e.g. instances can have from 1GB to 4GB of memory, *or*
3. Size can be chosen from a predefined set of values, an *enum*eration, e.g., instances can have 512 MB, 1 GB or 4GB of memory.

When creating a new *instance*, a client must specify the *hardware profile* on which the *instance* is based. Optionally a client can also specify values for the variable dimensions of the given hardware profile (otherwise the defaults specified within each hardware profile are used).

3.2.1 GET /api/hardware_profiles

Produce a list if all *hardware profiles* availaible with this cloud.

3.2.2 GET /api/hardware_profiles/:id

The attributes of a *hardware profile* consist of a human-readable `name` and a list of `<property/>` elements. Each property defines possible values along a sizing dimension. In the example below, the `large` hardware profile defines instances with exactly 2 virtual CPUs, memory from between 2GB and 4GB and local storage that can either be 850MB or 1GB. The default value for each dimension is indicated by the `value` attribute on the property element.

In addition to the sizing constraints, the hardware profile also lists which parameters can be used in instance operations to change the value of a property. In the example shown in Figure 4, only values for the instance create operation can be changed.

```
<hardware_profile href="..." id="large">
  <name>Large profile</name>
  <property kind="fixed" name="cpu" unit="count" value="2"/>

  <property kind="range" name="memory" unit="MB"
   value="10240">
     <param href="/api/instances" method="post" name="hwp_memory"
      operation="create"/>
        <range first="2048" last="4096"/>
  </property>

  <property kind="enum" name="storage" unit="MB"
   value="850">
     <param href="/api/instances" method="post" name="hwp_storage"
      operation="create"/>
       <enum>
          <entry value="850"/>
          <entry value="1024"/>
       </enum>
  </property>

  <property kind="fixed" name="architecture" unit="label"
   value="x86_64"/>

</hardware_profile>
```

Figure 4: Hardware profiles expressing `fixed`, `range` and `enum` values for attributes

3.3 Images

Images are used to launch *instances*. An *image* has human-readable `name` and `description` attributes as well as an `architecture`. Each *image* rep-

resents a virtual machine image in the back-end cloud, containing the root partition and initial storage for an *instance* operating system.

3.3.1 `GET /api/images`

Return a list of all *images* available in the back-end cloud.

3.3.2 `GET /api/images/:id`

Describe on *image* in detail. The `architecture` element indicates what CPU architecture this image expects. It can be either `x86_64` for Intel-based 64 bit processors, and `i386` for Intel-compatible 32 bit processors. The XML description of the image is as shown in Figure 5

```
<image href="/api/images/img1" id="img1">
   <name>Fedora 10</name>
   <owner_id>fedoraproject</owner_id>
   <description>Fedora 10</description>
   <architecture>x86_64</architecture>
</image>
```

Figure 5: XML description of an *image*

3.4 Instance states

Each cloud defines a slightly different lifecycle model for *instances*. In some clouds, *instances* start running immediately after creation, in others, they enter a pending state and they need to be explicitly started to become running.

These differences between clouds are modelled by expressing the lifecycle of an *instance* as a finite state machine and capturing this in a *instance_states* entity. The start state of the automaton is `start` and its final state is `finished`. The API defines the following states for an instance, as in Table 1.

The actions (state transitions) possible for an *instance* are as shown in Table 2. The precise actions that can be performed on a specific *instance* are expressed as part of the details for that *instance* (in `action` attributes, as shown in Figure 7). An example of the finite state machine for an *instance* is shown in Figure 6.

State	Meaning
start	Instances are in this state before they are created
pending	Creation of the instance has been requested and is in progress
running	The instance is running
shutting_down	A shutdown has been requested for the instance and is in progress
stopped	The instance is stopped
finished	All resources for the instance has been freed

Table 1: Instance states and their meanings

Action	Meaning
start	Start the instance
stop	Stop/shutdown the instance
reboot	Reboot the instance
destroy	Stop the instance and completely destroy it

Table 2: Transitions between instance states

3.4.1 `GET /api/instance_states`

The *instance_states* entity defines the transitions possible between the various states of an *instance*, and these are back-end cloud specific. In effect instance_states defines the finite state machine for instances from the given cloud. An example is shown in Figure 6 (note that the diagram does not show the full *instance_state* description).

3.5 Instances

An *instance* represents the focus of all cloud compute activity: a running virtual machine. An *instance* is created from an `image`, with a specified `hardware_profile` and in a given `realm`. Besides these attributes, each *instance* also has a human readable `name`, an `owner_id`, a `state`, `public_addresses` and `private_addresses` (IP address). As shown in Figure 7, each *instance* also has an `actions` attribute (which will depend on current state) as well as a `key`.

3.5.1 `GET /api/instances`

Produce a listing of all current `Instances` in the given cloud (belonging to the specified account).

```
<states>

   <state name="start">
       <transition auto="true" to="pending"/>
   </state>

   <state name="pending">
       <transition auto="true" to="running"/>
       <transition action="stop" to="stopping"/>
       <transition auto="true" to="stopped"/>
   </state>

   <state name="running">
       <transition action="reboot" to="running"/>
       <transition action="stop" to="stopping"/>
   </state>

   <state name="stopping">
   </state>

   ...

</states>
```

Figure 6: Part of the finite state machine of an instance

3.5.2 `GET /api/instances/:id`

Get the details for a specific Instance. The XML returned by the server is as shown in Figure 7.

3.5.3 `POST /api/instances/:id/:action`

The valid `actions` for an *instance* are as specified by the *instance_states* entity. At a given time and depending on the current *instance* `state`, the set of permissible actions is as reported in the response to `GET /api/instances/:id` (an example is shown in Figure 7).

3.5.4 `POST /api/instances/:create`

Create a new *instance*. At a minimum clients must specify the *image* from which the virtual machine *instance* is to be created. Optionally a client may also specify a *hardware_profile* and *realm* (with default values used otherwise). The details of the new *instance* are returned in response to this operation.

```
<instance href="http://localhost:3001/api/instances/i-6109820b"
 id="i-6109820b">
    <name>ami-73f2171a</name>
    <owner_id>297467797945</owner_id>
    <image href="http://localhost:3001/api/images/ami-73f2171a"
      id="ami-73f2171a"/>
    <realm href="http://localhost:3001/api/realms/us-east-1a"
      id="us-east-1a"/>
    <state>PENDING</state>
    <hardware_profile href="http://localhost:3001/api/hardware_profiles/
      m1.small" id="m1.small">
    </hardware_profile>
    <actions>
        <link href="http://localhost:3001/api/instances/i-6109820b/stop"
         method="post" rel="stop"/>
    </actions>
    <launch_time>2010-08-11T13:13:10.000Z</launch_time>
    <public_addresses>
        <address/>
    </public_addresses>
    <private_addresses>
        <address/>
    </private_addresses>
    <authentication type="key">
        <login>
            <keyname>eftah</keyname>
        </login>
    </authentication>
</instance>
```

Figure 7: XML description of a specified instance

3.6 Keys

A *key* captures the credentials required to access an Instance. These could be reported by the back-end cloud during instance creation, in which case they are captured from response in a *key* of type `:password` (with `:username` and `:password` attributes). For other cloud providers, credentials are specified by the client itself. In this case credentials are represented by a *key* of type `:key` (with `:fingerprint` and `:private_key` attributes). Figure 8 shows a *key* of type `:key`.

3.6.1 `GET /api/keys`

This gives a listing of all available keys.

3.6.2 `GET /api/keys/:id`

Get the XML description for a specified key, as shown in Figure 8

```
<key href="http://localhost:3001/api/keys/eftah" id="eftah"
 type="key">

   <actions>
       <link href="http://localhost:3001/api/keys/eftah"
        method="delete" rel="destroy"/>
   </actions>

   <fingerprint>
       a3:d1:51:8d:8b:cd:5e:23:e6:1e:84:d7:31:78:d1:3e:ac:e3:a5:95
   </fingerprint>
</key>
```

Figure 8: XML description of a key of type :key

3.6.3 `POST /api/keys/create`

Some back end cloud providers allow a client to create new credentials for accessing Instances. The parameters (key attributes) required by this function will depend on the back-end and are specified in the relevant driver. At present only the EC2 driver implements a `key_create` method and this requires the `name` parameter to be specified to create the EC2 keypair. This feature will be further implemented as support is provided by back-end clouds (e.g., Terremark Vcloud Express has recently added support for key management).

4 Storage resources

Storage resources are divided into two groups: *storage volumes* can be attached to a running *instance* (mountable by the instance OS), and *blob storage* which represents a simpler '$key <-- > value$' based data store (such as Rackspace CloudFiles or Amazon S3). *Storage snapshots* represent a *storage volume*, a backup of which is created at a particular point in time (a snapshot).

4.1 Storage volumes (in development)

4.1.1 GET /api/storage_volumes

List all *storage_volumes*.

4.1.2 GET /api/storage_volumes/:id

Get the details for a specic *storage_volume*, as shown in Figure 9.

```
<storage_volume href="http://localhost:3001/api/
 storage_volumes/vol-5403443d" id="vol-5403443d">
   <created>2010-08-11T20:40:11.000Z</created>
   <capacity>1</capacity>
   <device/>
   <state>AVAILABLE</state>
   <instance/>
</storage_volume>
```

Figure 9: XML description of a *storage volume*

4.2 Storage snapshots (in development)

4.2.1 GET /api/storage_snapshots/

List all available *storage snapshots*.

4.2.2 GET /api/storage_snapshots/:id

Get all details for a specified *storage snapshot*, as shown in Figure 10.

```
<storage_snapshot href="http://localhost:3001/api/
 storage_snapshots/snap-201dac4b" id="snap-201dac4b">
   <created>2010-08-11T20:46:10.000Z</created>
   <state>COMPLETED</state>
   <storage_volume href="http://localhost:3001/api/
    storage_volumes/vol-5403443d" id="vol-5403443d"/>
</storage_snapshot>
```

Figure 10: XML description of a *storage snapshot*

4.3 Blob storage (in development)

5 Further information, errata and contributions

- General Deltacloud site:
  `http://deltacloud.org/`,
- Deltacloud API incubation status page at Apache Incubator:
  `http://incubator.apache.org/projects/deltacloud.html`,
- IRC channel `'#deltacloud'` (freenode),
- `'deltacloud-dev'` at Apache incubator: deltacloud-dev@incubator.apache.org
  `http://mail-archives.apache.org/mod_mbox/incubator-deltacloud-dev/`,
- `'deltacloud-users'`: deltacloud-users@lists.fedorahosted.org
  `https://fedorahosted.org/mailman/listinfo/deltacloud-users`,
- `'deltacloud-devel'`: deltacloud-devel@lists.fedorahosted.org
  `https://fedorahosted.org/mailman/listinfo/deltacloud-devel`.