Zend_Service_Rackspace - Enrico Zimuel

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Service_Rackspace Component Proposal

Proposed Component Name	Zend_Service_Rackspace
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Service_Rackspace
Proposers	Enrico Zimuel
Zend Liaison	TBD
Revision	1.0 - 2 May 2011: Initial Draft. 1.1 - 11 May 2011: Completed the prototype of the Rackspace Files service Checkout the source code here (wiki revision: 19)

Table of Contents

1. Overview
Cloud Files
Cloud Servers
Load Balancers
API version
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
Files
Servers
9. Class Skeletons

1. Overview

Zend\Service\Rackspace is a class to manage the cloud service of Rackspace: Files, Servers, Load Balancers.

Cloud Files

The Cloud Files is a service to store any files in a cloud environment. A user can store an unlimited quantity of files and each file can be as large as 5 gigabytes. The files can be private or public. The private files can be accessed using the API of Rackspace. The public files are accessed using a CDN (Content Delivery Network).
Rackspace exposes a REST API to manage the Cloud Files.
Using the API features of Cloud Files you can:

- Upload files programmatically for tight integration with your application
- Enable Cloud Files CDN integration on any container for public distribution
- Create Containers programmatically
- Retrieve lists of containers and files

The **Zend\Service\Rackspace\Files** component will implement all these features.

Cloud Servers

Rackspace Cloud Servers is a compute service that provides server capacity in the cloud. Cloud Servers come in different flavors of memory, disk space, and CPU.
Using the Rackspace Server API you can:

- Create new servers
- List and get information on each server
- Delete a server
- Manage the public/private IP addresses of a server
- Resize the server capacity
- Reboot a server
- Create new images for a server
- Manage the backup of a server
- Create a group of server to share the IP addresses for High Availability architecture

The **Zend\Service\Rackspace\Servers** component will implement all these features.

Load Balancers

A load balancer is a logical device which belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration. The Rackspace Cloud Load Balancers API is implemented using a ReSTful web service interface. Like other products in the Rackspace Cloud suite, the load balancing service shares a common token-based authentication system that allows seamless access between products and services.
Using the Rackspace Load Balancers API you can:

- Manage the load balancer
- Create virtual IPs
- Manage access list
- Check the usage report
- Monitor the system
- Manage Session Persistance

The **Zend\Service\Rackspace\LoadBalancers** component will implement all these features.

API version

The current version of the Rackspace Cloud API is **1.0**.

2. References

- Rackspace Cloud Files - Developer Guide
- Rackspace Cloud Servers - Developer Guide
- Rackspace Cloud Load Balancer - Developer Guide
- PHP API for the Cloud Files

3. Component Requirements, Constraints, and Acceptance Criteria

- This component will require a valid Rackspace account.
- This component will implement full Rackspace Cloud REST API.
- This component will use the JSON format to exchange data with the Rackspace API
- This component will throw an Exception in case of a response error.

4. Dependencies on Other Framework Components

- Zend\Http\Client
- Zend\Validator\Ip

5. Theory of Operation

We propose to write an abstract class (**Zend\Service\Rackspace\Rackspace**) to share methods and properties common to all the **Rackspace** services: Files, Servers and Load Balancers.
The authentication request of the Rackspace API is shared between all the cloud services and can be implemented in the abstract class. Each subclass of Zend\Service\Rackspace will implement a specific service.

6. Milestones / Tasks

- Milestone 1: Complete the proposal for Files [DONE]
- Milestone 2: Prototype of the Zend\Service\Rackspace\Files [DONE]
- Milestone 3: Provide examples and unit tests
- Milestone 4: Write the documentation
- Milestone 5: Propose an adapter for the Zend\Cloud\StorageService
- Milestone 6: Complete the proposal for Servers
- Milestone 7: Prototype of the Zend\Service\Rackspace\Servers
- Milestone 8: Provide examples and unit tests
- Milestone 9: Write the documentation
- Milestone 10: Propose an adapter for the Zend\Cloud\Infrastructure
- Milestone 11: Complete the proposal for Load Balancers
- Milestone 12: Prototype of the Zend\Service\Rackspace\LoadBalancers
- Milestone 13: Provide examples and unit tests
- Milestone 14: Write the documentation

7. Class Index

- Zend\Service\Rackspace\Rackspace
- Zend\Service\Rackspace\Exception
- Zend\Service\Rackspace\Files
- Zend\Service\Rackspace\Files\Object
- Zend\Service\Rackspace\Files\ObjectList
- Zend\Service\Rackspace\Files\Container
- Zend\Service\Rackspace\Files\ContainerList
- Zend\Service\Rackspace\Servers
- Zend\Service\Rackspace\Servers\Server
- Zend\Service\Rackspace\Servers\ServerList
- Zend\Service\Rackspace\Servers\Image
- Zend\Service\Rackspace\Servers\ImageList
- Zend\Service\Rackspace\Servers\SharedIpGroup
- Zend\Service\Rackspace\Servers\SharedIpGroupList

8. Use Cases

Files

UC1-01

Get all the containers and the files for each container.

```
$user= 'xxx';
$key= 'yyy';

$cloudFiles= new \Zend\Service\Rackspace\Files($user,$key);

if (!$cloudFiles->authenticate()) {
 die('Error during the authentication: '.$cloudFiles->getErrorMsg());
}

$containers= $cloudFiles->getContainers();

foreach ($containers as $container) {
 echo "Container: ".$container->getName()."\n\n";
 $listFiles= $container->getObjects();
 foreach ($listFiles as $file) {
  echo "File name: ".$file->getName()."\n";
  echo "Size (byte): ".$file->getSize()."\n";
 }
}
```

UC1-02

Add an object (file) in a container. The metadata array is optional.

```
$user= 'xxx';
$key= 'yyy';

$cloudFiles= new \Zend\Service\Rackspace\Files($user,$key);

$file= file_get_contents('php.jpg');

$metadata= array (
    'foo' => 'bar',
    'foo2' => 'bar2'
);

$container='test';

if ($cloudFiles->storeObject($container,'php.jpg',$file,$metadata)) {
    echo 'The file was uploaded successfully';
} else {
    die ("ERROR: ".$cloudFiles->getErrorMsg());
}
```

UC1-03

Create a container, store an object, and delete it. The metadata array is optional.

```
$user= 'xxx';
$key= 'yyy';

$cloudFiles= new \Zend\Service\Rackspace\Files($user,$key);

$metadata= array (
    'foo' => 'bar',
    'test' => 'value'
);

$container=$cloudFiles->createContainer('test',$metadata);

if ($cloudFiles->isSuccessful()) {
    echo 'The container was created successfully';
} else {
    die ("ERROR: ".$cloudFiles->getErrorMsg());
}


$file= file_get_contents('php.jpg');

if ($container->addObject('php.jpg',$file)) {
  echo 'The file was uploaded successfully';
} else {
  die ("ERROR: ".$cloudFiles->getErrorMsg());
}

if ($container->deleteObject('php.jpg')) {
  echo 'The file was deleted successfully';
} else {
  die ("ERROR: ".$cloudFiles->getErrorMsg());
}
```

UC1-04

Check if a container is CDN enabled. If not, try to enable it.

```
$user= 'xxx';
$key= 'yyy';

$cloudFiles= new \Zend\Service\Rackspace\Files($user,$key);

$container= $cloudFiles->getContainer('test');

if ($container->isCdnEnabled()) {
    echo "The container is CDN enabled\n";
} else {
    echo "The container is not CDN enabled\n";
    if ($container->enableCdn()) {
      echo "The container has been enabled as CDN\n";
    } else {
      die("ERROR: Failed to enable the CDN. ".$cloudFiles->getErrorMsg());
    }
}
echo "This is the CDN URI: ".$container->getCdnUri()."\n";
echo "This is the CDN URI SSL: ".$container->getCdnUriSsl()."\n";
```

UC1-05

Copy an object (file) from a container to another. During the copy you can change the name of the object (file) and add metadata to it (optional).

```
$user= 'xxx';
$key= 'yyy';

$cloudFiles= new \Zend\Service\Rackspace\Files($user,$key);

// Add metadata in the destination file (optional)
$metadata = array ('foo' => 'bar');

$container_source= 'test';
$file_source= 'php.jpg';
$container_dest= 'img';
$file_dest= 'logo_php.jpg';

if ($cloudFiles->copyObject($container_source,$file_source,$container_dest,$file_dest,$metadata)) {
    echo "The file has been copied successfully\n";
} else {
    die("ERROR: ".$cloudFiles->getErrorMsg());
}
```

Servers

UC2-01

Create a new server instance. Metadata ($metadata) and files ($files) are optional parameters.

```
$user= 'xxx';
$key= 'yyy';

$rackspace= new Zend\Service\Rackspace\Servers($user,$key);

$data = array (
    'name'     => 'test',
    'imageId'  => '49',
    'flavorId' => '1'
);

$metadata = array(
    'foo' => 'bar'
);

$files = array (
    '/root/test' => '/home/enrico/test'
);

$server= $rackspace->createServer($data,$metadata,$files);

if ($server===false) {
    die("ERROR: ".$rackspace->getErrorMsg());
}

printf ("Name: %s\n",$server->getName());
printf ("Id: %s\n",$server->getId());
$ips= $server->getPublicIp();
printf ("IP public: %s\n",$ips[0]);
echo "Metadata:\n";
print_r($server->getMetadata());
printf ("Admin password: %s\n",$server->getAdminPass());
```

UC2-02

Get a server, using the ID of the server.

```
$user= 'xxx';
$key= 'yyy';

$rackspace= new Zend\Service\Rackspace\Servers($user,$key);

$server= $rackspace->getServer('id');

if ($server===false) {
    die("ERROR: ".$rackspace->getErrorMsg());
}

printf("Name: %s\n",$server->getName());
printf("Id: %s\n",$server->getId());
printf("Flavor Id: %s\n",$server->getFlavorId());
```

UC2-03

Get all the servers. The parameter of listServers() is a boolean flag to get a list of server with detailed information (true) or not (false).

```
$user= 'xxx';
$key= 'yyy';

$rackspace= new Zend\Service\Rackspace\Servers($user,$key);

$servers= $rackspace->listServers(true);

if ($servers===false) {
    die("ERROR: ".$rackspace->getErrorMsg());
}

foreach ($servers as $srv) {
    printf("Name: %s\n",$srv->getName());
    printf("Id: %s\n",$srv->getId());
    printf("Flavor Id: %s\n",$srv->getFlavorId());
    printf("Image Id: %s\n",$srv->getImageId());
    printf("Status: %s\n\n",$srv->getStatus());
}
```

UC2-04

Change the name of a server. The parameter of listServers() is a boolean flag to get a list of server with detailed information (true) or not (false).

```
$user= 'xxx';
$key= 'yyy';

$rackspace= new Zend\Service\Rackspace\Servers($user,$key);

if ($rackspace->changeServerName('id','new name')) {
    echo "The name of the server has been changed successfully\n";
} else {
    die("ERROR: ".$rackspace->getErrorMsg());
}
```

UC2-05

Change the admin password of a server. For security reason the admin password is returned only when you create a server.

```
$user= 'xxx';
$key= 'yyy';

$rackspace= new Zend\Service\Rackspace\Servers($user,$key);

if ($rackspace->changeServerPassword('id','new password')) {
    echo "The admin password of the server has been changed successfully\n";
} else {
    die("ERROR: ".$rackspace->getErrorMsg());
}
```

9. Class Skeletons

```


namespace Zend\Service\Rackspace;

use Zend\Service\Rackspace\Exception,
 Zend\Http\Client as HttpClient;

abstract class Rackspace
{
    const VERSION                = 'v1.0';
    const US_AUTH_URL            = 'https://auth.api.rackspacecloud.com';
    const UK_AUTH_URL            = 'https://lon.auth.api.rackspacecloud.com';
    const API_FORMAT             = 'json';
    const USER_AGENT             = 'Zend\Service\Rackspace';
    const STORAGE_URL            = "X-Storage-Url";
    const AUTHtoken              = "X-Auth-Token";
    const AUTHuser_HEADER        = "X-Auth-User";
    const AUTHkey_HEADER         = "X-Auth-Key";
    const AUTHuser_HEADER_LEGACY = "X-Storage-User";
    const AUTHkey_HEADER_LEGACY  = "X-Storage-Pass";
    const AUTHtoken_LEGACY       = "X-Storage-Token";
    const CDNM_URL               = "X-CDN-Management-Url";
    const MANAGEMENT_URL         = "X-Server-Management-Url";
    /**
     * Rackspace Key
     *
     * @var string
     */
    protected $key;
    /**
     * Rackspace account name
     *
     * @var string
     */
    protected $user;
    /**
     * Token of authentication
     *
     * @var string
     */
    protected $token;
    /**
     * Authentication URL
     *
     * @var string
     */
    protected $authUrl;
    /**
     * @var Zend\Http\Client
     */
```

```
    protected $httpClient;
    /**
     * Error Msg
     *
     * @var string
     */
    protected $errorMsg;
    /**
     * HTTP error code
     *
     * @var string
     */
    protected $errorCode;
    /**
     * Storage URL
     *
     * @var string
     */
    protected $storageUrl;
    /**
     * CDN URL
     *
     * @var string
     */
    protected $cdnUrl;
    /**
     * Server management URL
     *
     * @var string
     */
    protected $managementUrl;
    /**
     * __construct()
     *
     * You must pass the account and the Rackspace authentication key.
     * Optional: the authentication url (default is US)
     *
     * @param string $user
     * @param string $key
     * @param string $authUrl
     */
    public function __construct($user, $key, $authUrl=self::US_AUTH_URL)
    /**
     * Get User account
     *
     * @return string
     */
    public function getUser()
    /**
     * Get user key
     *
     * @return string
     */
    public function getKey()
    /**
     * Get authentication URL
     *
     * @return string
     */
    public function getAuthUrl()

    /**
     * Get the storage URL
     *
     * @return string|boolean
     */
    public function getStorageUrl()
    /**
     * Get the CDN URL
     *
```

```
     * @return string|boolean
     */
    public function getCdnUrl()
    /**
     * Get the management server URL
     *
     * @return string|boolean
     */
    public function getManagementUrl()

    /**
     * Set the user account
     *
     * @param string $user
     * @return void
     */
    public function setUser($user)

    /**
     * Set the authentication key
     *
     * @param string $key
     * @return void
     */
    public function setKey($key)

    /**
     * Set the Authentication URL
     *
     * @param string $url
     * @return void
     */
    public function setAuthUrl($url)

    /**
     * Get the authentication token
     *
     * @return string
     */
    public function getToken()

    /**
     * Get the error msg of the last REST call
     *
     * @return string
     */
    public function getErrorMsg()
    /**
     * Get the error code of the last REST call
     *
     * @return strig
     */
    public function getErrorCode()
    /**
     * get the HttpClient instance
     *
     * @return Zend\Http\Client
     */
    public function getHttpClient()

    /**
     * Return true is the last call was successful
     *
     * @return boolean
     */
    public function isSuccessful()

    /**
     * HTTP call
     *
```

```
     * @param string $url
     * @param string $method
     * @param array $headers
     * @param array $get
     * @param string $body
     * @return Zend\Http\Response
     */
    protected function httpCall($url,$method,$headers=array(),$data=array(),$body=null)

    /**
     * Authentication
     *
     * @return boolean
     */
```

```
    public function authenticate()

  }
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>