

# Zend\_Cloud\_Infrastructure - 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\_Cloud\_Infrastructure Component Proposal

<b>Proposed Component Name</b>	Zend_Cloud_Infrastructure
<b>Developer Notes</b>	<a href="http://framework.zend.com/wiki/display/ZFDEV/Zend_Cloud_Infrastructure">http://framework.zend.com/wiki/display/ZFDEV/Zend_Cloud_Infrastructure</a>
<b>Proposers</b>	Enrico Zimuel
<b>Zend Liaison</b>	TBD
<b>Revision</b>	1.0 - 4 April 2011: Initial Draft. 1.1 - 17 June 2011: Alpha version of Zend\Cloud\Infrastructure for ZF2 ( <a href="#">available on github</a> ) (wiki revision: 18)

## Table of Contents

1. Overview
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
9. Class Skeletons

## 1. Overview

Zend\Cloud\_Infrastructure is a component of the cloud services to manage a cloud computing infrastructure.

### Cloud Components in Zend Framework

In Zend Framework we have a specific class, Zend\Cloud, to manage different cloud services coming from different vendors. This class is an abstract interface of the most common used cloud services in the market.

These cloud services are:

- **Document databases** (Zend\Cloud\DocumentService)
- **Queue** (Zend\Cloud\Queue)
- **Storage** (Zend\Cloud\Storage)

The adapters supported by these services are reported in the table below:

	<b>Document</b>	<b>Queue</b>	<b>Storage</b>
<b>Amazon Web Service</b>	(SimpleDB)	(SQS)	(S3)
<b>Windows Azure</b>			
<b>Nirvanix</b>			

The Zend\Cloud class uses some specific Zend Framework components to implement the vendor adapters. These service classes are:

- Zend\Service\Amazon
- Zend\Service\Amazon\SimpleDB
- Zend\Service\Amazon\Sqs
- Zend\Service\Amazon\S3
- Zend\Service\WindowsAzure
- Zend\Service\WindowsAzure\Storage\Table
- Zend\Service\WindowsAzure\Storage\Queue
- Zend\Service\WindowsAzure\StorageBlob
- Zend\Service\Nirvanix

## Proposal

We would like to extend the Zend\Cloud class to manage the deploy phase of a PHP application in a cloud environment. In particular we would like to insert some features to manage the computational resources (instances) in a cloud environment. The idea is to provide an abstract interface to manage instances of different vendors.

There are other open source projects that provide these abstraction layer for cloud services for different computer languages. For instance:

- **libcloud**, <http://incubator.apache.org/libcloud/>  
a standard client library for many popular cloud providers, for python and java
- **unifiedcloud**, <http://code.google.com/p/unifiedcloud/>  
Unified Cloud Computing is an attempt to create an open and standardized cloud interface for the unification of various cloud api's. A singular programmatic point of contact that can encompass the entire infrastructure stack as well as emerging cloud centric technologies all through a unified interface. One of the key drivers of the unified cloud interface is to create an api about other api's. A singular programmatic point of contact that can encompass the entire infrastructure stack as well as emerging cloud centric technologies all through a unified interface.
- **deltacloud**, <http://incubator.apache.org/deltacloud/>  
an API that abstracts the differences between clouds.

We would like to provide a similar solution for PHP.

In order to manage the deploy of a web application in different cloud environments we have to support two different types of cloud services:

- by *Application*
- by *Infrastructure*

*Application* means cloud services focused on the applications instead of the infrastructure.

*Infrastructure* means cloud services focused on the infrastructure instead of the application.

For instance, Windows Azure is a Microsoft cloud service focused on applications, instead Amazon Elastic Compute Cloud (EC2) is a cloud service focused on the infrastructure.

Below we reported a more generic classification of the functionalities of some of the most important cloud vendors:

	Application	Infrastructure	Services
<b>Amazon Web Service</b>			
<b>Windows Azure</b>		?	
<b>GoGrid</b>			
<b>Nirvanix</b>			
<b>IBM</b>	?		
<b>Rackspace</b>			

The cloud vendors offer specific API to manage the cloud services. These API are based on web services (REST/SOAP) that can be easily managed in different computer languages. Some of these vendors offer specific API for PHP:

- Amazon Web Service SDK for PHP: <http://aws.amazon.com/php/>
- PHPAzure for Windows Azure: <http://phpazure.codeplex.com/>
- GoGrid API for PHP: [http://wiki.gogrid.com/wiki/index.php/API:PHP\\_API\\_Developer\\_Home](http://wiki.gogrid.com/wiki/index.php/API:PHP_API_Developer_Home)

## 2. References

- [CloudAPI theory](#)
- [Unified Cloud Interface Project](#)
- [libcloud](#)
- [Amazon Elastic Compute Cloud \(EC2\)](#)
- [GoGrid PHP API](#)
- [deltacloud](#)

## 3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** include a factory method.
- This component **will** use the `Zend\Service*` class to manage the Adapter of each cloud computing vendors (so far we have only available the class `Zend\Service\Amazon\Ec2`).
- This component **will not** save any data using `Zend\Cache` or the filesystem.

## 4. Dependencies on Other Framework Components

- `Zend\Service\Amazon\Ec2`
- `Zend\Service*` (cloud computing services)

## 5. Theory of Operation

The component is managed by a factory to initialize specific cloud computing adapters.

## 6. Milestones / Tasks

- Milestone 1: Proposal [DONE]
- Milestone 2: Working prototype supporting use cases #1, #2 .. #8 [DONE]
- Milestone 3: Working prototype supporting use cases #9 [DONE]
- Milestone 4: Working prototype supporting use cases #10 (you need the SSH extension enabled "ext/ssh2") [DONE]
- Milestone 5: Unit tests [DONE]
- Milestone 6: Initial documentation exists.

## 7. Class Index

- `Zend\Cloud\Infrastructure`
- `Zend\Cloud\Infrastructure\Adapter`
- `Zend\Cloud\Infrastructure\Exception`
- `Zend\Cloud\Infrastructure\Instance`
- `Zend\Cloud\Infrastructure\InstanceList`
- `Zend\Cloud\Infrastructure\Image`
- `Zend\Cloud\Infrastructure\ImageList`

## 8. Use Cases

### UC-01

```
use Zend\Cloud\Infrastructure\Factory,
    Zend\Cloud\Infrastructure\Instance,
    Zend\Cloud\Infrastructure\Adapter\Ec2 as Ec2Adapter;

$amazonKey= 'xxx';
$amazonSecret= 'yyy';
$amazonZone= 'us-east-1';

$infrastructure = Factory::getAdapter(array(
    Factory::INFRASTRUCTURE_ADAPTER_KEY =>
'Zend\Cloud\Infrastructure\Adapter\Ec2',
    Ec2Adapter::AWS_ACCESS_KEY => $amazonKey,
    Ec2Adapter::AWS_SECRET_KEY => $amazonSecret,
    Ec2Adapter::AWS_REGION => $amazonZone
));

$options= array (
    Instance::INSTANCE_IMAGEID => 'ami-8c1fece5',
    Ec2Adapter::AWS_SECURITY_GROUP => array ('default')
);

$instance= $infrastructure->createInstance('test',$options);

if ($instance->waitStatus(Instance::STATUS_RUNNING)) {
    printf("The instance %s is running now!", $instance->getId());
}
```

### UC-02

```
$nodeId= 'i-32242-sd';

$status= $infrastructure->statusInstance($nodeId);

echo "The node $nodeId is in the $status status";
```

### UC-03

```
$nodeId= 'i-32242-sd';

if ($infrastructure->rebootInstance($nodeId)) {
    echo "The node $nodeId has been rebooted successfully";
}
```

#### UC-04

```
$nodeId= 'i-32242-sd';

$zones= $infrastructure->zonesInstance();

echo "The available zones in the cloud are:\n";
print_r ($zones);
```

#### UC-05

```
$nodeId= 'i-32242-sd';

if ($infrastructure->stopInstance($nodeId)) {
    echo "The node $nodeId has been stopped successfully.";
}
```

#### UC-06

```
$nodeId= 'i-32242-sd';

if ($infrastructure->startInstance($nodeId)) {
    echo "The node $nodeId has been started successfully.";
}
```

#### UC-07

```
$nodeId= 'i-32242-sd';

if ($infrastructure->destroyInstance($nodeId)) {
    echo "The node $nodeId has been destroyed successfully.";
}
```

**UC-08**

```
$nodeId= 'i-32242-sd';

$images= $infrastructure->imagesInstance();

echo "The available images in the cloud are:\n";
print_r ($images);
```

**UC-09**

```
use Zend\Cloud\Infrastructure\Instance;

$nodeId= 'i-32242-sd';

$cpuUsage= $infrastructure->monitorInstance($nodeId,Instance::MONITOR_CPU);

echo "The CPU usage of the instance $nodeId is as follow:\n";
print_r ($cpuUsage);
```

**UC-10**

```
$nodeId= 'i-32242-sd';

$param= array (
    Instance::SSH_USERNAME => 'username',
    Instance::SSH_PASSWORD => 'password'
);

$cmd= 'ls -la /var/www';

$output= $infrastructure->deployInstance($nodeId,$param,$cmd);

echo "The files in the DocumentRoot of the $nodeId instance are:\n";
print_r ($output);
```

## 9. Class Skeletons

```
namespace Zend\Cloud\Infrastructure;
```

```

interface Adapter
{
    const HTTP_ADAPTER    = 'http_adapter';
    const TIMEOUT_STATUS_CHANGE = 30;
    /**
     * Return a list of the available instances
     *
     * @return array
     */
    public function listInstances();

    /**
     * Return the status of an instance
     *
     * @param string $id
     * @return string
     */
    public function statusInstance($id);

    /**
     * Wait for status $status with a timeout of $timeout seconds
     *
     * @param string $id
     * @param string $status
     * @param integer $timeout
     * @return boolean
     */
    public function
waitStatusInstance($id,$status,$timeout=self::TIMEOUT_STATUS_CHANGE);

    /**
     * Return the public DNS name of the instance
     *
     * @param string $id
     * @return string|boolean
     */
    public function publicDnsInstance($id);

    /**
     * Reboot an instance
     *
     * @param string $id
     * @return boolean
     */
    public function rebootInstance($id);

    /**
     * Create a new instance
     *
     * @param string $name
     * @param array $options
     * @return boolean
     */
    public function createInstance($name,$options);

    /**
     * Stop the execution of an instance
     *

```

```
* @param string $id
* @return boolean
*/
public function stopInstance($id);

/**
 * Start the execution of an instance
 *
 * @param string $id
 * @return boolean
 */
public function startInstance($id);

/**
 * Destroy an instance
 *
 * @param string $id
 * @return boolean
 */
public function destroyInstance($id);

/**
 * Return all the available instances images
 *
 * @return array
 */
public function imagesInstance();

/**
 * Return all the available zones
 */
public function zonesInstance();

/**
 * Return the system informations about the $metric of an instance
 *
 * @param string $id
 * @param string $metric
 * @param array $options
 * @return array
 */
public function monitorInstance($id,$metric,$options=null);

/**
 * Run arbitrary shell script on an instance
 *
 * @param string $id
 * @param array $param
 * @param string|array $cmd
 * @return string|array
 */
public function deployInstance($id,$param,$cmd);

/**
 * Get the adapter instance
 *
 * @return object
 */
public function getAdapter();
```

```
/**
 * Get the adapter result
 *
 * @return array
 */
public function getAdapterResult();

/**
 * Get the last HTTP response
 *
 * @return Zend\Http\Response
 */
public function getLastHttpResponse();

/**
 * Get the last HTTP request
 *
 * @return string
 */
public function getLastHttpRequest();
```

}

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