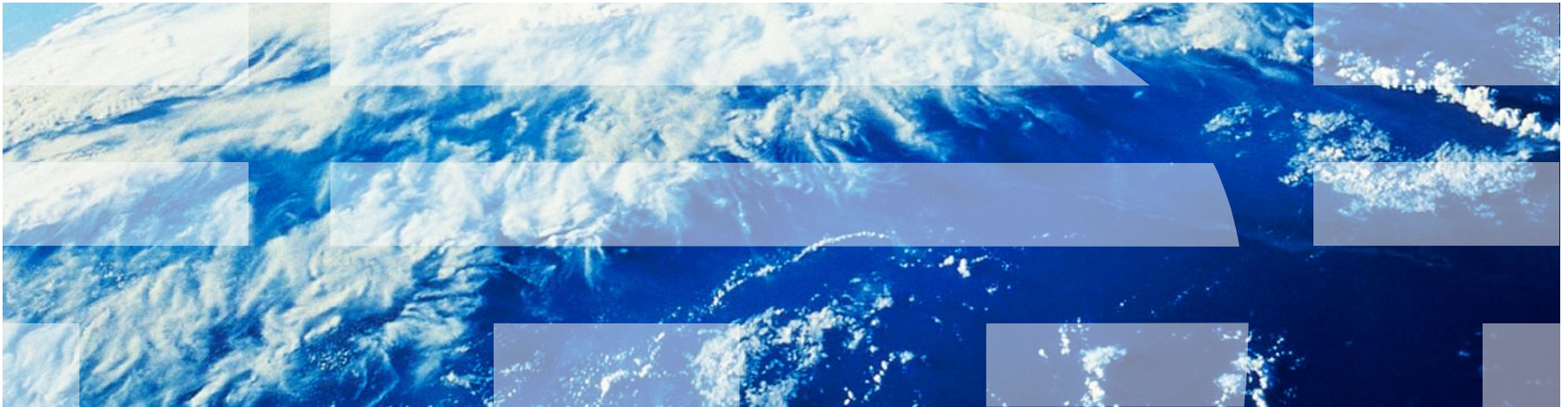


# The Simple Cloud Storage API



---

# Agenda

- Portability and interoperability
- Getting started
- Basic operations
- Metadata
- Going beyond the basics
- Building your own adapter
- Resources / Next steps

# The Simple Cloud API



- A joint effort of Zend, GoGrid, IBM, Microsoft, Nirvanix and Rackspace
  - But you can add your own libraries to support other cloud providers.
- The goal: Make it possible to write portable, interoperable code that works with multiple cloud vendors.
- There's an article on the Simple Cloud API in the developerWorks Open Source zone: [bit.ly/1bSkTx](http://bit.ly/1bSkTx)

# The Simple Cloud API

- Covers three areas:
  - File storage (S3, Nirvanix, Azure Blob Storage, Rackspace Cloud Files)
  - Document storage (SimpleDB, Azure Table Storage)
  - Simple queues (SQS, Azure Table Storage)
- Uses the Factory and Adapter design patterns
  - A configuration file tells the Factory object which adapter to create.

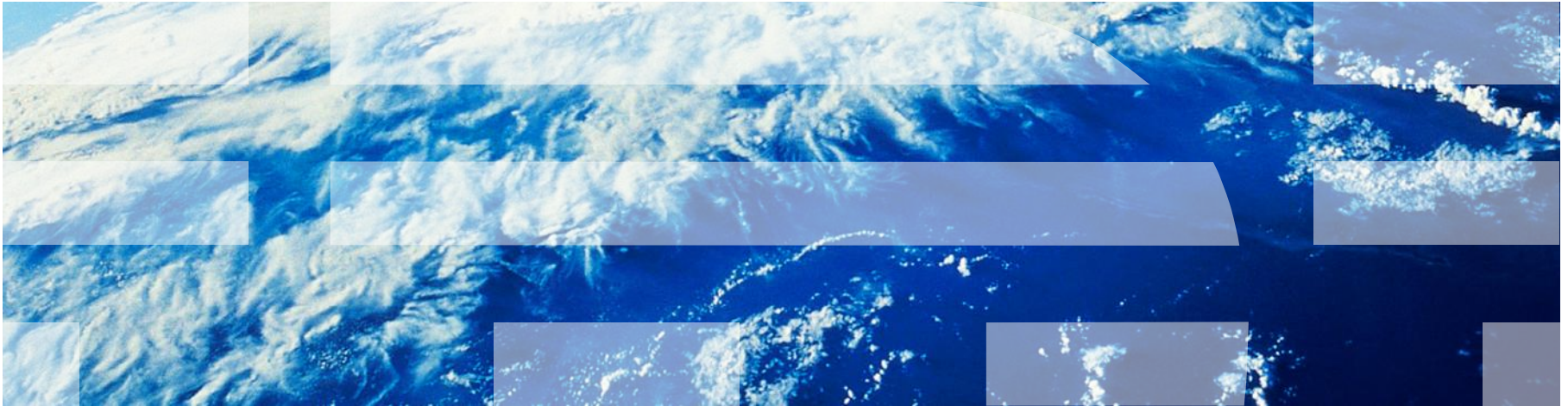
# Vendor lock-in

- If there's a new technology, any talented programmer will want to use it.
  - Maybe the shiny new thing is appropriate for what we're doing.
  - Maybe not.
  - We're probably going to use it anyway.
- The challenge is to walk the line between using the newest, coolest thing and avoiding vendor lock-in.

# Portability and Interoperability

- In writing flexible code for the cloud, there are two key concepts:
  - **Portability** is the ability to run components or systems written for one cloud provider in another cloud provider's environment.
  - **Interoperability** is the ability to write one piece of code that works with multiple cloud providers, regardless of the differences between them.
- You'll often have to write code that works with multiple providers at the same time.

# Getting started



# Getting started

- The Simple Cloud API uses dependency injection to do its magic.
- We'll create an adapter using a properties file.
  - That properties file contains all the information Zend needs to create the adapter.
  - Once Zend creates the adapter, it takes the values that adapter needs to do its job and injects them into the class.

# Getting started

- The first step in using the Simple Cloud API is creating a properties file.
  - This uses the `Zend_Config_Ini` class.
  - Each Simple Cloud adapter has different values, but all of them contain the `storage_adapter` parameter. That tells Zend the name of the class to instantiate.

# Getting started

- When you're writing a PHP application that uses Simple Cloud, you need to include three classes:

```
require_once
```

```
'Zend/Cloud/StorageService/Factory.php';
```

```
require_once
```

```
'Zend/Http/Client/Adapter/Socket.php';
```

```
require_once 'Zend/Config/Ini.php';
```

# A properties file

- `s3.ini` looks like this:

```
storage_adapter =  
    "Zend_Cloud_StorageService_Adapter_S3"  
bucket_name = "open_cloud_demo"  
aws_accesskey = "75PHLGJ5AXVAZQ"  
aws_secretkey =  
    "1LERHdtVMtaw/qEn9W7O3xU9HOSSdD5"  
http_adapter =  
    "Zend_Http_Client_Adapter_Socket"
```

## Another properties file

- Here's `nirvanix.ini`:

```
storage_adapter =  
"Zend_Cloud_StorageService_Adapter_Nirvanix"  
nirvanix.appName = "ZendCon"  
auth_accesskey = "533a2...79ef10"  
auth_username = "johndoe"  
auth_password = "PD3x7Js/"  
remote_directory = "/johndoe"
```

## And one more

- Finally, `openstack.ini`:

```
storage_adapter =  
"Zend_Cloud_StorageService_Adapter_Rackspace"  
api_key = "d9b67018f861f12b23379330a"  
auth_username = "johndoe"  
auth_password = "Uk3XJkf0w"  
remote_directory = "Pictures"  
local_url = "http://localhost..."
```

- (More on Open Stack later.)

# Getting started: Properties files

- With the properties file built, we create the storage adapter:

```
$credentials =  
    new Zend_Config_Ini($configFile);  
$sc = Zend_Cloud_StorageService_Factory::  
    getAdapter($credentials);
```

- After these lines of code, `$sc` is a storage adapter.
  - What class it actually is, we don't know. (Or care.)

# Vendor-specific APIs

- Listing all the items in a Nirvanix directory:

```
$auth = array('username'    => 'your-username',  
              'password'    => 'your-password',  
              'appKey'      => 'your-appkey');  
  
$nirvanix = new Zend_Service_Nirvanix($auth);  
$imfs = $nirvanix->getService('IMFS');  
$args = array('folderPath' => '/dougtdwell',  
              'pageNumber' => 1,  
              'pageSize'   => 5);  
  
$stuff = $imfs->ListFolder($args);
```

- All of these lines of code are specific to Nirvanix.

# Vendor-specific APIs

- Listing all the items in an S3 bucket:

```
$s3 = new Zend_Service_Amazon_S3  
      ($accessKey, $secretKey);
```

```
$stuff =
```

```
    $s3->getObjectsByBucket($bucketName);
```

- All of these lines of code are specific to S3.

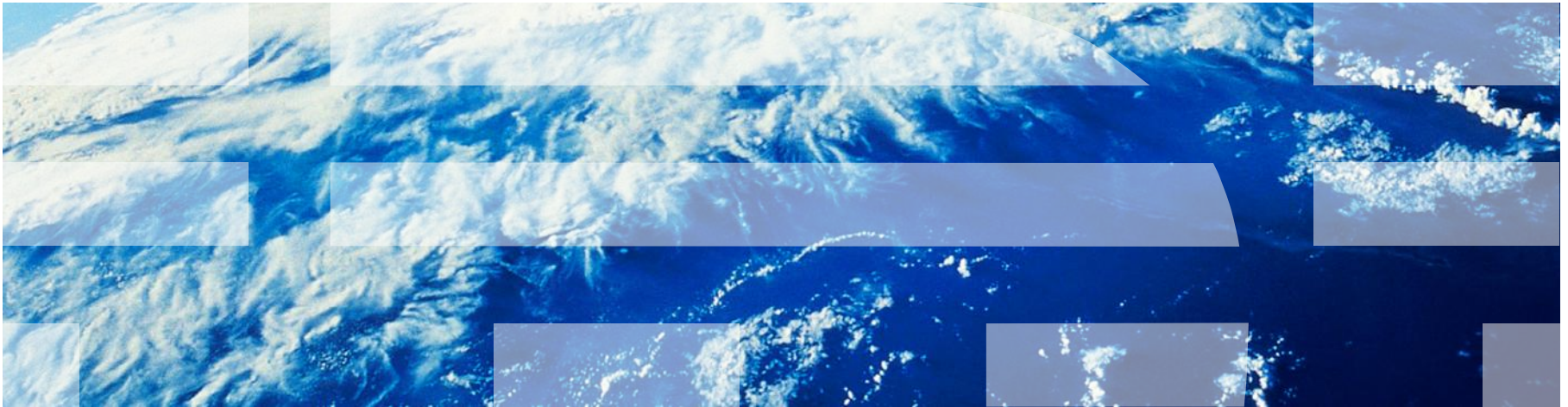
# The Simple Cloud Storage API

- Listing all the items in a Nirvanix directory or S3 bucket or OpenStack folder:

```
$credentials =  
    new Zend_Config_Ini($configFile);  
$stuff = Zend_Cloud_StorageService_Factory  
    ::getAdapter($credentials)->listItems();
```

- These lines of code work with Nirvanix and S3 (and OpenStack, etc.).
  - We can write one application that works with all of these providers.

# Basic operations



# Methods

- The storage API supports several common operations:
  - `storeItem()`, `fetchItem()` and `deleteItem()`
  - `copyItem()`, `moveItem()` and `renameItem()`
  - `listItems()`
  - `storeMetadata()`, `fetchMetadata()` and `deleteMetadata()`
- Not all of these are supported natively.
  - More on this in a minute.

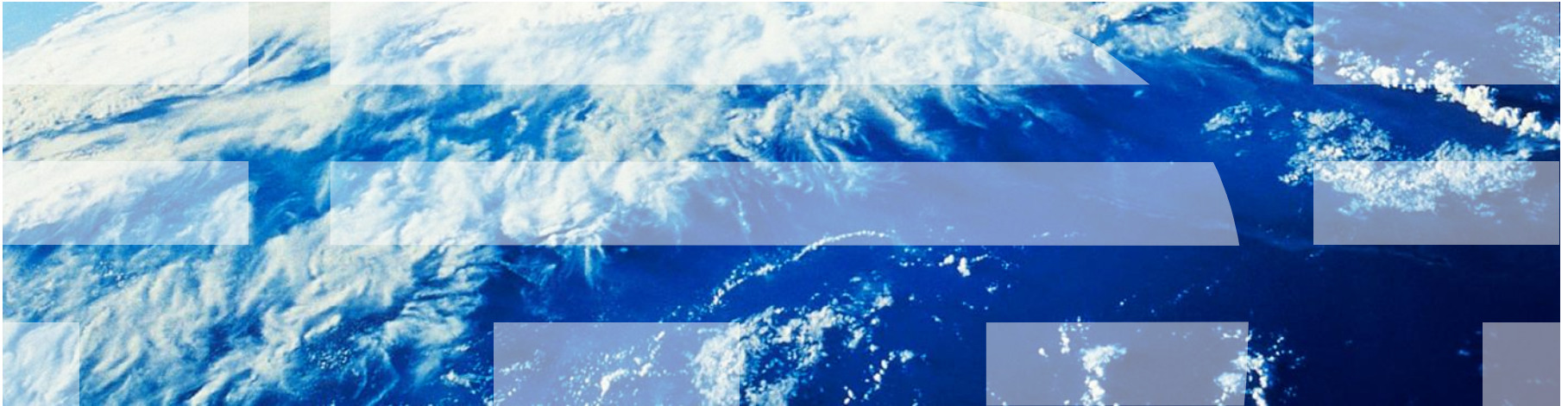
# Demo

- We'll take a quick look at how the Simple Cloud storage API uses properties files to work with multiple cloud storage providers.

# Issues

- Not all storage services support renaming files.
  - You can hack this, but....
- Not all storage services support listing containers.
- In a few minutes we'll look at a way around this.

# Metadata



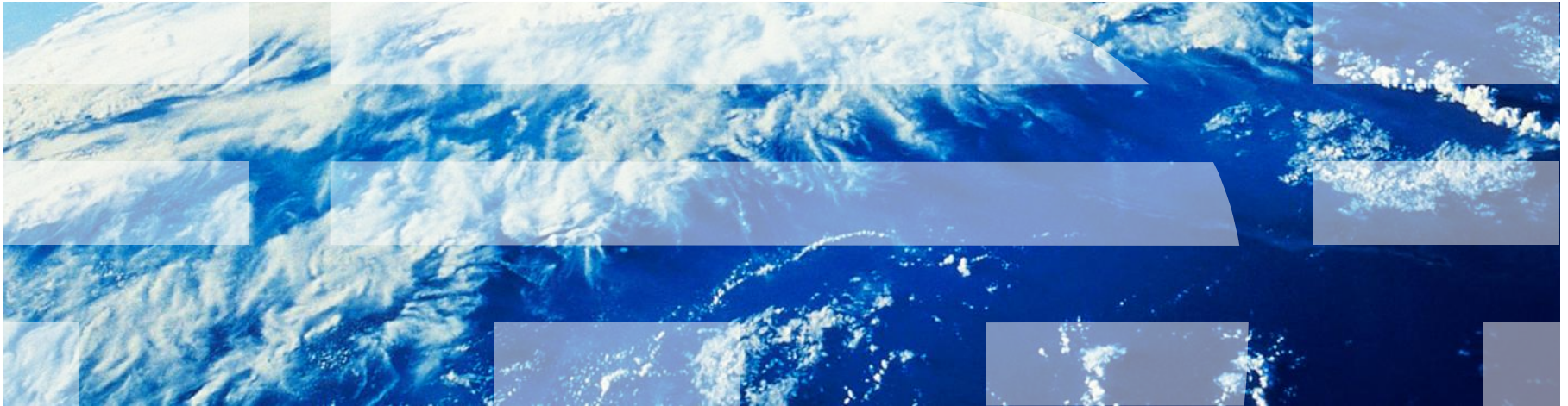
# Metadata

- Many features of cloud storage services are managed by metadata.
- The S3 adapter, for example, supports four values for access control:
  - **S3\_ACL\_PRIVATE**
  - **S3\_ACL\_PUBLIC\_READ**
  - **S3\_ACL\_PUBLIC\_WRITE**
  - **S3\_ACL\_AUTH\_READ**

# Metadata

- The `fetchMetadata()` method lets you get metadata about a particular object.
  - `$metadata = $sc->fetchMetadata();`
- As the Simple Cloud API continues to evolve, working with metadata will become a very important way of handling differences between providers.

# Going beyond the basics



## Going beyond the basics

- Although the Simple Cloud API provides many useful functions, there are times when you need to use functions it doesn't provide.
- The `getClient()` function lets you do this.

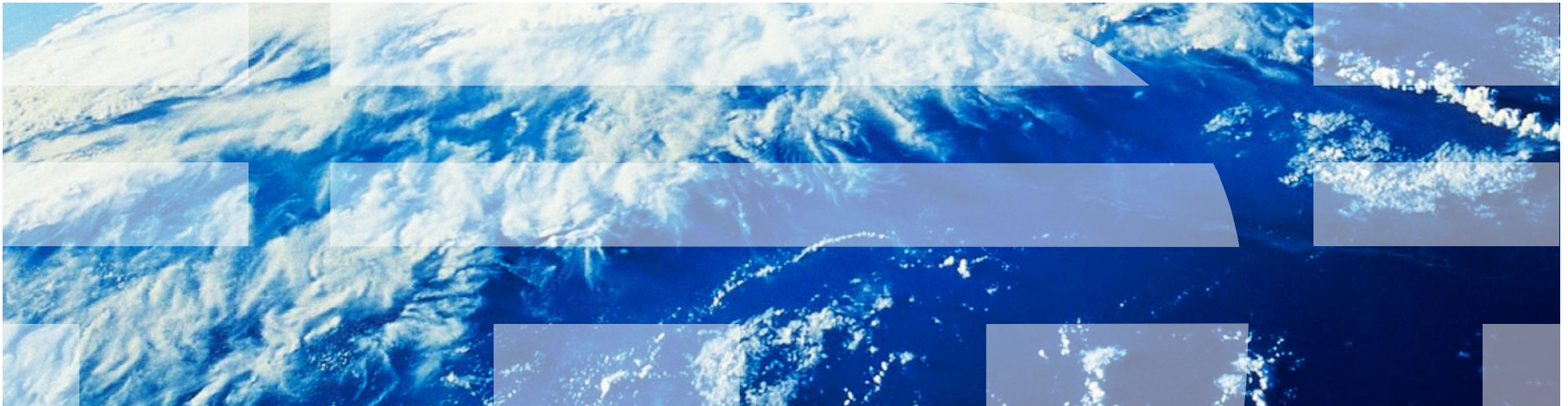
```
if ($sc->getClient() instanceof
    Zend_Service_Nirvanix) {
    $sc->getClient()->[CDN function]();
}
```

# Issues

- Not all storage services support renaming files.
  - You can hack this, but....
- Not all storage services support listing containers.
- The `getClient()` method gives you a reasonable way to work around these inconsistencies.
- There are other approaches:

```
if ($sc->getClient()->
    supportsRenaming())
```
- We need your input!

# Building your own adapter



# Writing your own adapter

- To write your own storage adapter, you have to implement all of the methods of the **Zend\_Cloud\_StorageService\_Adapter** interface.
- If the cloud vendor you're targeting already has a PHP library for the service, you're 80% there.

# An OpenStack adapter

- Here's how to build an Open Stack adapter:
  1. Figure out how to map the native PHP cloud class to the Open Stack API
  2. Figure out what parameters the adapter needs
  3. Wire everything together.
    - With any luck, most of the code you'll need will be in the native class.

# Mapping the API

- The OpenStack API is based on the Rackspace API
- Some operations supported by the Rackspace API:
  - `list_objects()`
  - `create_object()`
  - `get_object()`
  - `delete_object()`

# Mapping the API

```
listItems () = list_objects ()  
storeItem () = create_object ()  
fetchItem () = get_object ()  
deleteItem () = delete_object ()
```

# Handling properties

- The next step is to figure out what properties the adapter needs to initialize itself.
  - Rackspace needs a username, a password, an API key and a remote directory.
  - Because OpenStack is deployed locally, we also need the URL of the OpenStack storage server.

# The property values

- Once again , `openstack.ini`:

```
storage_adapter =  
"Zend_Cloud_StorageService_Adapter_Rackspace"  
api_key = "d9b67018f861f12b23379330a"  
auth_username = "johndoe"  
auth_password = "Uk3XJkf0w"  
remote_directory = "Pictures"  
local_url = "http://localhost..."
```

# Handling the property values

- In your constructor you need to initialize those values from the `Zend_Config_Ini` object:

```
const USERNAME           = 'auth_username';  
const PASSWORD          = 'auth_password';  
const APP_KEY           = 'api_key';  
const REMOTE_DIRECTORY  = 'remote_directory';  
const LOCAL_URL         = 'local_url';
```

## Writing the code

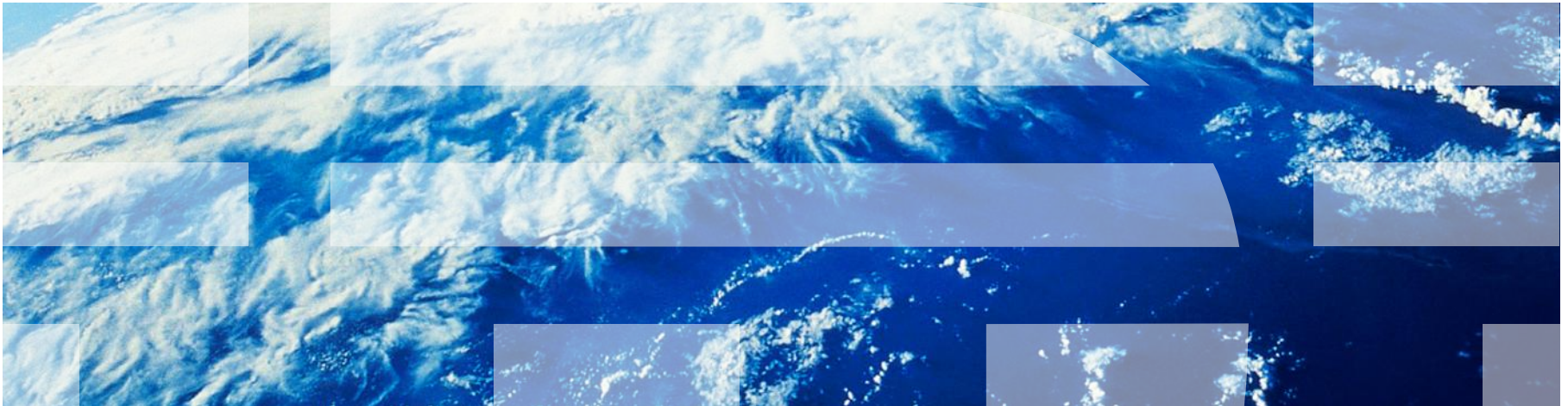
- With the property values set, you need to create the appropriate objects for the native adapter.
- For Rackspace and OpenStack:
  - Create a **CF\_Authentication** object.
  - Once you're authenticated, create a **CF\_Connection** object.
  - Once you have the **CF\_Connection**, get the container you're working with.

## Writing the code

- Create the native adapter inside your Simple Cloud adapter, then implement the Simple Cloud storage interface by wiring it to the native code:

```
public function listItems
    ($path, $options = null) {
    $resultArray = $this->
        _remoteDirectory->list_objects();
    return $resultArray;
}
```

# Summary



# Summary

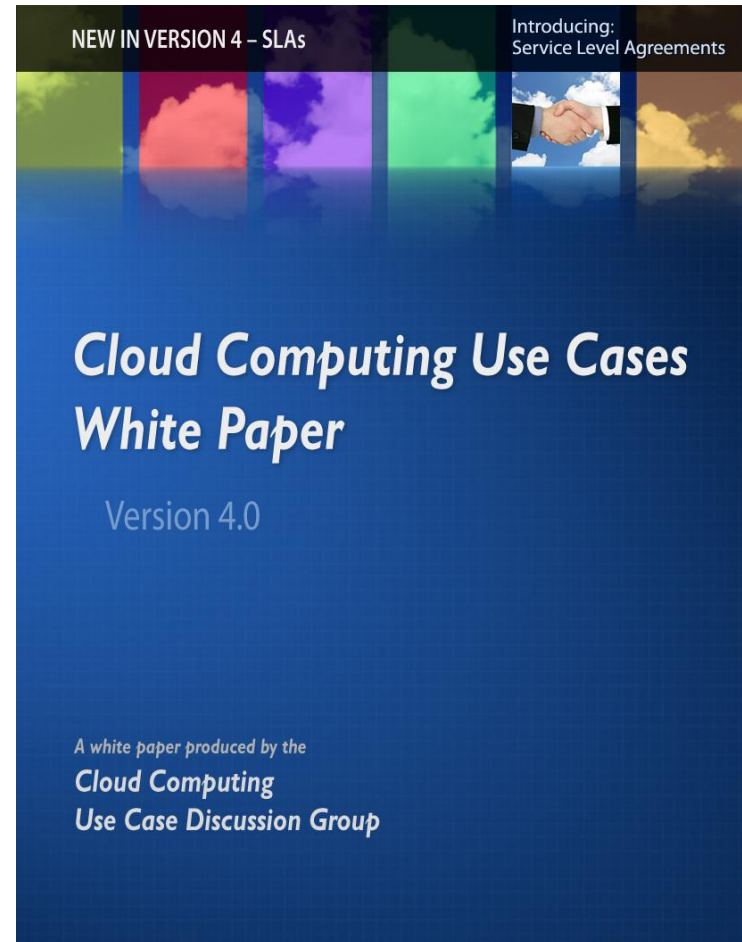
- The Storage API provides many useful functions that let you work with multiple cloud storage providers.
  - You get both portability and interoperability.
  - You can use the native adapter if you need to.
  - You can also write your own adapters.
- **You focus on writing code that matters.**

# Get Involved!

- Simple Cloud API
  - Download the code
  - Build a prototype
  - Submit requirements / new adapters / bug reports
  - Visit [simplecloud.org](http://simplecloud.org)
- Watch for future Simple Cloud Webinars
  - We'll look at the details of the Queue and Document (database) APIs

# cloudusecases.org

- The Cloud Computing Use Cases group is focused on documenting customer requirements.
- Join us!
- Cloud computing will be the biggest change to IT since the rise of the Web.
  - But to make the most of it, we have to keep things open.
  - And everybody has to get involved to make that happen.
- Version 5, **Moving to the Cloud**, is underway.



# Join us!

- ZendCon 2010, the premier PHP conference, is the first week of November in Santa Clara.
- This year the show also features the ZendCon UnConference and a Cloud Camp.
- See [zendcon.com](http://zendcon.com) for all the details.



Zend/PHP Conference & Expo



**November 1-4**  
Santa Clara, CA

# Thanks!

