AWS SDK for PHP Version 2

Migration Guide

AWS SDK for PHP Version 2: Migration Guide

Copyright © 2012 Amazon Web Services LLC or its affiliates. All rights reserved.

The following are trademarks or registered trademarks of Amazon: Amazon, Amazon.com, Amazon.com Design, Amazon DevPay, Amazon EC2, Amazon Web Services Design, AWS, CloudFront, EC2, Elastic Compute Cloud, Kindle, and Mechanical Turk. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Welcome

This guide shows how to migrate your code to use the new AWS SDK for PHP 2 and how the new SDK differs from the first version of the SDK.

Introduction

The PHP language and community have evolved significantly over the past few years. Since the inception of the AWS SDK for PHP, PHP has gone through two major version changes (versions 5.3 and 5.4) and many in the PHP community have unified behind the recommendations of the PHP Framework Interop Group. Consequently, we decided to make breaking changes to the SDK in order to align with the more modern patterns used in the PHP community.

For the new release, we rewrote the SDK from the ground up to address popular customer requests. The new SDK is built on top of the Guzzle HTTP client framework, which provides increased performance and enables event-driven customization. We also introduced high-level abstractions to make programming common tasks easy. The SDK is compatible with PHP 5.3.2 and newer, and follows the PSR-0 standard for namespaces and autoloading.

Which Services are Supported?

The AWS SDK for PHP 2 does not currently support all AWS services. However, we will incrementally add support for other services in upcoming releases to the Version 2 SDK.

Fortunately, it is straightforward to use the previous SDK side-by-side with this version. This enables using the SDK for the services it supports while relying on the prior SDK for those it does not. Please see the Side-by-Side Guide for details.

Follow our AWS SDK for PHP 2 GitHub repository to stay up-to-date with the latest changes.

What's New?

The following list describes new features that have been implemented in the AWS SDK for PHP 2.

- PHP 5.3 namespaces
- Follows PSR-0, PSR-1, and PSR-2 standards
- Built on Guzzle and utilizes the Guzzle feature set
- Persistent connection management for both serial and parallel requests
- Event hooks (via Symfony2 EventDispatcher) for event-driven, custom behavior
- Request and response entity bodies are stored in `php://temp` streams to reduce memory usage
- Transient networking and cURL failures are automatically retried using truncated exponential backoff
- Plug-ins for over-the-wire logging and response caching
- "Waiter" objects that allow you to poll a resource until it is in a desired state
- Resource iterator objects for easily iterating over paginated responses
- Service-specific sets of exceptions
- Modeled responses with a simpler interface
- Grouped constants (Enums) for service parameter options
- Flexible request batching system
- Service builder/container that supports easy configuration and dependency injection
- Full unit test suite with extensive code coverage
- Composer support (including PSR-0 compliance) for installing and autoloading SDK dependencies
- Phing `build.xml` for installing dev tools, driving testing, and producing `.phar` files
- Powerful Amazon DynamoDB batch write system
- Powerful Amazon Simple Storage Service (Amazon S3) and Amazon Glacier multipart upload system
- Redesigned DynamoDB Session Handler
- Improved multi-region support

What's Different?

The following list describes what's different between AWS SDK for PHP 2 and the previous version.

- **Architecture** – The new SDK is built on top of Guzzle and inherits its features and conventions. Every AWS service client extends the Guzzle client, defining operations through a service description file. The SDK has a much more robust and flexible object-oriented architecture, including the use of design patterns, event dispatching and dependency injection. As a result, many of the classes and methods from the previous SDK have been changed.
- **Project Dependencies** - Unlike the Version 1 of the SDK, the new SDK does not pre-package all of its dependencies in the repository. Dependencies are best resolved and autoloaded via Composer. However, when installing the SDK via the downloadable phar, the dependencies are resolved for you.
- **Namespaces** – The SDK's directory structure and namespaces are organized according to PSR-0 standards, making the SDK inherently modular. The `Aws\Common` namespace contains the core code of the SDK, and each service client is contained in its own separate namespace (e.g., `Aws\DynamoDb`).
- **Coding Standards** – The SDK adopts the PSR standards produced by the PHP Framework Interop Group. An immediately noticeable change is that all method names are now named using lower camel-case (e.g., `putObject` instead of `put_object`).
- **Required Regions** – The region must be explicitly provided to instantiate a client. The AWS region you select may affect your both performance and costs.
- **Client Factories** – Factory methods instantiate service clients and do the work of setting up the signature, exponential backoff settings, exception handler, and so forth. At a minimum you must provide your access key, secret key, and region to the client factory, but there are many other settings you can use to customize the client behavior.

```
$s3 = Aws\S3\S3Client::factory(array(
  'key'    => 'your-aws-access-key-id',
  'secret' => 'your-aws-secret-access-key',
  'region' => 'us-west-2',
));
```

- **Configuration / Service Builder** – A global configuration file can be used to inject credentials into clients automatically via the service builder. The service builder acts as a dependency injection container for the service clients. (**Note:** The SDK does not automatically attempt to load the configuration file like in Version 1 of the SDK.)

```
$aws = Aws\Common\Aws::factory('/path/to/custom/config.php');
$s3 = $aws->get('s3');
```

This technique is the preferred way for instantiating service clients. Your `config.php` might look similar to the following:

```
<?php
return array(
  'includes' => array('_aws'),
  'services' => array(
      'default_settings' => array(
          'params' => array(
              'key'    => 'your-aws-access-key-id',
              'secret' => 'your-aws-secret-access-key',
              'region' => 'us-west-2'
          )
      )
  )
);
```

The line that says `'includes' => array('_aws')` includes the default configuration file packaged with the SDK. This sets up all of the service clients for you so you can retrieve them by name with the `get()` method of the service builder.

- **Service Operations** – Executing operations in the new SDK is similar to how it was in the previous SDK, with two main differences. First, operations follow the lower camel-case naming convention. Second, a single array parameter is used to pass in all of the operation options. The following examples show the Amazon S3 `PutObject` operation performed in each SDK:

```
// Previous SDK - PutObject operation
$s3->create_object('bucket-name', 'object-key.txt', array(
  'body' => 'lorem ipsum'
));
```

```
// New SDK - PutObject operation
$result = $s3->putObject(array(
  'bucket' => 'bucket-name',
  'key'    => 'object-key.txt',
  'body'   => 'lorem ipsum'
));
```

In the new SDK, the `putObject()` method doesn't actually exist as a method on the client. It is implemented using the `__call()` magic method of the client and acts as a shortcut to instantiate a command, execute the command, and retrieve the result.

A `Command` object encapsulates the request and response of the call to AWS. From the `Command` object, you can call the `getResult()` method (as in the preceding example) to retrieve the parsed result, or you can call the `getResponse()` method to retrieve data about the response (e.g., the status code or the raw response).

The `Command` object can also be useful when you want to manipulate the command before execution or need to execute several commands in parallel. The following is an example of the same `PutObject` operation using the command syntax:

```
$command = $s3->getCommand('PutObject', array(
  'bucket' => 'bucket-name',
  'key'    => 'object-key.txt',
  'body'   => 'lorem ipsum'
));
$result = $command->execute();
```

Or you can use the chainable `set()` method on the `Command` object:

```
$s3->getCommand('PutObject')
  ->set('bucket', 'bucket-name')
  ->set('key', 'object-key.txt')
  ->set('body', 'lorem ipsum')
  ->execute();
```

- **Responses** – The format of responses has changed. Responses are no longer instances of the `CFResponse` object. The `Command` object (as seen in the preceding section) of the new SDK encapsulates the request and response, and is the object from which to retrieve the results.

```
// Previous SDK
$response = $s3->list_tables(); // Execute the operation and get the CFResponse
 object
$result = $response->body;     // Get the parsed response body as a SimpleXM
LElement

// New SDK
$result = $s3->listTables();     // Executes the operation and gets the re
sponse in normalized array-like object
```

The new syntax is similar, but a few fundamental differences exist between responses in the previous SDK and this version:

- The new SDK represents parsed responses (i.e., the results) as Guzzle `Model` objects instead of `CFSimpleXML` objects as in the prior version. These Model objects are easy to work with since they act like arrays. They also have helpful built-in features such as mapping and filtering. The content of the results will also look different in this version of the SDK. The SDK marshals responses into the models and then transforms them into more convenient structures based on the service description. The API documentation details the response of all operations.
- The new SDK uses exceptions to communicate errors and bad responses.

Instead of relying on the `CFResponse::isOK()` method of the previous SDK to determine if an operation is successful, the new SDK throws exceptions when the operation is *not* successful. Therefore, you can assume success if there was no exception thrown, but you will need to add `try...catch` logic to your application code in order to handle potential errors. The following is an example of how to handle the response of an Amazon DynamoDB `DescribeTable` call in the new SDK:

```
$tableName = 'my-table';
```

```
try {
    $result = $dynamoDb->describeTable(array('TableName' => $tableName));

   printf('The provisioned throughput for table "%s" is %d RCUs and %d WCUs.',

        $tableName,
        $result->getPath('Table/ProvisionedThroughput/ReadCapacityUnits'),
        $result->getPath('Table/ProvisionedThroughput/WriteCapacityUnits')
    );
} catch (Aws\DynamoDb\Exception\DynamoDbException $e) {
    printf('The provisioned throughput could not be determined for table
"%s".', $tableName);
}
```

You can also get the Guzzle response object back from the command. This is helpful if you need to retrieve the status code, additional data from the headers, or the raw response body.

```
$command = $dynamoDb->getCommand('DescribeTable', array('TableName' =>
$tableName));

$statusCode = $command->getResponse()->getStatusCode();
```

You can also get the response object and status code from the exception if one is thrown.

```
try {
    $result = $dynamoDb->describeTable(array(TableName' => 'my-table'));
} catch (Aws\DynamoDb\Exception\DynamoDbException $e) {
    $statusCode = $e->getResponse()->getStatusCode();
}
```

- **Iterators** The SDK provides iterator classes that make it easier to traverse results from list and describe type operations. Instead of having to code solutions that perform multiple requests in a loop and keep track of tokens or markers, the iterator classes do that for you. You can simply foreach over the iterator:

```
$objects = $s3->getIterator('ListObjects', array(
    'bucket' => 'my-bucket-name'
));

foreach ($objects as $object) {
    echo $object['Key'] . PHP_EOL;
}
```

Comparing Code Samples from Both SDKs

This section provides examples of how you would implement accessing AWS differently in the AWS SDK for PHP 2 versus the previous version.

Topics

Example 1 - Amazon S3 ListParts Operation

From Version 1 of the SDK

```
<?php

require '/path/to/sdk.class.php';
require '/path/to/config.inc.php';

$s3 = new AmazonS3();

$response = $s3->list_parts('my-bucket-name', 'my-object-key', 'my-upload-id',
 array(
    'max-parts' => 10
));

if ($response->isOK())
{
    // Loop through and display the part numbers
    foreach ($response->body->Part as $part) {
        echo "{$part->PartNumber}\n";
    }
}
else
```

```
{
    echo "Error during S3 ListParts operation.\n";
}
```

From Version 2 of the SDK

```
<?php

require '/path/to/vendor/autoload.php';

use Aws\Common\Aws;
use Aws\S3\Exception\S3Exception;

$aws = Aws::factory('/path/to/config.php');
$s3 = $aws->get('s3');

try {
    $result = $s3->listParts(array(
        'bucket'    => 'my-bucket-name',
        'key'       => 'my-object-key',
        'uploadId'  => 'my-upload-id',
        'max-parts' => 10
    ));

    // Loop through and display the part numbers
    foreach ($result['Part'] as $part) {
        echo "{$part[PartNumber]}\n";
    }
} catch (S3Exception $e) {
    echo "Error during S3 ListParts operation.\n";
}
```

Example 2 - Amazon DynamoDB Scan Operation

From Version 1 of the SDK

```
<?php

require '/path/to/sdk.class.php';
require '/path/to/config.inc.php';

$dynamo_db = new AmazonDynamoDB();

$start_key = null;
$people = array();

// Perform as many Scan operations as needed to acquire all the names of people
 that are 16 or older
do
{
```

```
    // Setup the parameters for the DynamoDB Scan operation
    $params = array(
        'TableName'       => 'people',
        'AttributesToGet' => array('id', 'age', 'name'),
        'ScanFilter'      => array(
            'age' => array(
                'ComparisonOperator' => AmazonDynamoDB::CONDITION_GREAT
ER_THAN_OR_EQUAL,
                'AttributeValueList' => array(
                    array(AmazonDynamoDB::TYPE_NUMBER => '16')
                )
            ),
        )
    );

    // Add the exclusive start key parameter if needed
    if ($start_key)
    {
        $params['ExclusiveStartKey'] = array(
            'HashKeyElement' => array(
                AmazonDynamoDB::TYPE_STRING => $start_key
            )
        );

        $start_key = null;
    }

    // Perform the Scan operation and get the response
    $response = $dynamo_db->scan($params);

    // If the response succeeded, get the results
    if ($response->isOK())
    {
        foreach ($response->body->Items as $item)
        {
            $people[] = (string) $item->name->{AmazonDynamoDB::TYPE_STRING};
        }

        // Get the last evaluated key if it is provided
        if ($response->body->LastEvaluatedKey)
        {
            $start_key = (string) $response->body->LastEvaluatedKey-
>HashKeyElement->{AmazonDynamoDB::TYPE_STRING};
        }
    }
    else
    {
        // Throw an exception if the response was not OK (200-level)
        throw new DynamoDB_Exception('DynamoDB Scan operation failed.');
    }
}
while ($start_key);

print_r($people);
```

From Version 2 of the SDK

```
<?php

require '/path/to/vendor/autoload.php';

use Aws\Common\Aws;
use Aws\DynamoDb\Enum\ComparisonOperator;
use Aws\DynamoDb\Enum\Type;

$aws = Aws::factory('/path/to/config.php');
$dynamodb = $aws->get('dynamodb');

// Create a ScanIterator and setup the parameters for the DynamoDB Scan operation
$scan = $dynamodb->getIterator('Scan', array(
    'TableName'       => 'people',
    'AttributesToGet' => array('id', 'age', 'name'),
    'ScanFilter'      => array(
        'age' => array(
            'ComparisonOperator' => ComparisonOperator::GE,
            'AttributeValueList' => array(
                array(Type::NUMBER => '16')
            )
        ),
    )
));

// Perform as many Scan operations as needed to acquire all the names of people
 that are 16 or older
$people = array();
foreach ($scan as $item) {
    $people[] = $item['name'][Type::STRING];
}

print_r($people);
```