AWS SDK for PHP Version 2

Side By Side Guide

AWS SDK for PHP Version 2: Side By Side Guide

Welcome

This guide helps you install, configure, and run Version 1 and Version 2 of the AWS SDK for PHP side-by-side within the same application or project. Please see the Migration Guide for more information on migrating code from the original AWS SDK for PHP to Version 2.

The AWS SDK for PHP 2 introduces many desirable new features and advantages, but the initial version of Version 2 does not support for *all* of the AWS services supported by Version 1. We will add support for other services in upcoming releases to the Version 2 SDK, but you can begin migrating code now by using the SDKs side-by-side.

Installing and Including the SDKs

To install and include the SDKs in your project, you must first choose whether or not to use Composer.

Using Composer

Using Composer is the recommended way to install both versions of the AWS SDK for PHP. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project requires and installs them into your project. In order to simultaneously use both versions of the SDK in the same project through Composer, you must do the following:

1. Add both of the SDKs as dependencies in your project's `composer.json` file.

```
{
    "require": {
        "amazonwebservices/aws-sdk-for-php": "*"
        "aws/aws-sdk-php": "*"
    }
}
```

Note: Consider tightening your dependencies to a known version when deploying mission critical applications (e.g., `2.0.*`).

2. Download and install Composer.

```
curl -s "http://getcomposer.org/installer" | php
```

3. Install your dependencies.

```
php composer.phar install
```

4. Require Composer's autoloader.

Composer also prepares an autoload file that's capable of autoloading all of the classes in any of the libraries that it downloads. To use it, just add the following line to your code's bootstrap process.

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

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Without Composer

Without Composer, you must manage your own project dependencies.

1. Download both of the SDKs (via PEAR or the AWS website) into a location accessible by your project. Make certain to use the pre-packaged `aws.phar` file, which includes all of the dependencies for the AWS SDK for PHP 2.
2. In your code's bootstrap process, you need to explicitly require the bootstrap file from Version 1 of the SDK and the `aws.phar` file containing Version 2 of the SDK:

```
// Include each of the SDK's bootstrap files to setup autoloading
require '/path/to/sdk.class.php'; // Load the Version 1 bootstrap file
require '/path/to/aws.phar';      // Load the Version 2 pre-packaged phar
file
```

Configuring and Instantiating the SDKs

How you configure and instantiate the SDKs is determined by whether or not you are using the service builder (`Aws\Common\Aws` class).

Instantiating Clients via the Service Builder

The service builder (`Aws\Common\Aws class`) in the AWS SDK for PHP 2 enables configuring all service clients with the same credentials. It also accepts additional settings for some or all of the clients. The service builder functionality is inherited from the Guzzle project.

You can pass the service builder a configuration file containing your credentials and other settings. It will then inject these into all of the service clients your code instantiates. For more information about the configuration file, please read the Getting Started Guide. When using both SDKs side-by-side, your configuration file must include the following line:

```
'includes' => array('_aws', '_sdk1'),
```

This will automatically set up the service clients from Version 1 of the SDK making them accessible through the service builder by keys such as `v1.s3` and `v1.cloudformation`. Here is an example configuration file that includes referencing the Version 1 of the SDK:

```
<?php return array(
    'includes' => array('_aws', '_sdk1'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'your-aws-access-key-id',
                'secret' => 'your-aws-secret-access-key'
                'region' => 'us-west-2',
            )
        )
```

```
    )
);
```

Your code must instantiate the service builder through its factory method by passing in the path of the configuration file. Your code then retrieves instances of the specific service clients from the returned builder object.

```
use Aws\Common\Aws;

// Instantiate the service builder
$aws = Aws::factory('/path/to/your/config.php');

// Instantiate S3 clients via the service builder
$s3v1 = $aws->get('v1.s3');  // All Version 1 clients are prefixed with "v1."
$s3v2 = $aws->get('s3');
```

Instantiating Clients via Client Factories

Your code can instantiate service clients using their respective `factory()` methods by passing in an array of configuration data, including your credentials. The `factory()` will work for clients in either versions of the SDK.

```
use Aws\S3\S3Client;

// Create an array of configuration options
$config = array(
    'key'    => 'your-aws-access-key-id',
    'secret' => 'your-aws-secret-access-key',
    'region' => 'us-west-2'
);

// Instantiate Amazon S3 clients from both SDKs via their factory methods
$s3v1 = AmazonS3::factory($config);
$s3v2 = S3Client::factory($config);
```

Optionally, you could alias the classes to make it clearer which version of the SDK they are from.

```
use AmazonS3 as S3ClientV1;
use Aws\S3\S3Client as S3ClientV2;

$config = array(
    'key'    => 'your-aws-access-key-id',
    'secret' => 'your-aws-secret-access-key',
    'region' => 'us-west-2'
);

$s3v1 = S3ClientV1::factory($config);
$s3v2 = S3ClientV2::factory($config);
```

Complete Examples

The following two examples fully demonstrate including, configuring, instantiating, and using both SDKs side-by-side. These examples adopt the recommended practices of using Composer and the service builder.

Example 1 - Dual Amazon S3 Clients

This example demonstrates using an Amazon S3 client from the AWS SDK for PHP 2 working side-by-side with an Amazon S3 client from the first PHP SDK.

```
<?php

require 'vendor/autoload.php';

$aws = Aws\Common\Aws::factory('/path/to/config.json');

$s3v1 = $aws->get('v1.s3');
$s3v2 = $aws->get('s3');

echo "ListBuckets with SDK Version 1:\n";
echo "------------------------\n";
$response = $s3v1->listBuckets();
if ($response->isOK()) {
    foreach ($response->body->Buckets->Bucket as $bucket) {
        echo "- {$bucket->Name}\n";
    }
} else {
    echo "Request failed.\n";
}
echo "\n";

echo "ListBuckets with SDK Version 2:\n";
echo "------------------------\n";
try {
    $result = $s3v2->listBuckets();
    foreach ($result['Buckets']['Bucket'] as $bucket) {
```

```
        echo "- {$bucket['Name']}\n";
    }
} catch (Aws\S3\Exception\S3Exception $e) {
    echo "Request failed.\n";
}
echo "\n";
```

Example 2 - Amazon DynamoDB and Amazon SNS Clients

This example shows how the AWS SDK for PHP 2 DynamoDB client works together with the SNS client from the original SDK. For this example, an ice cream parlor publishes a daily message (via SNS) containing its "flavors of the day" to subscribers. First, it retrieves the flavors of the day from its DynamoDB database using the AWS SDK for PHP 2 DynamoDB client. It then uses the SNS client from the first SDK to publish a message to its SNS topic.

```
<?php

require 'vendor/autoload.php';

$aws = Aws\Common\Aws::factory('/path/to/config.json');

// Instantiate the clients
$ddb = $aws->get('dynamodb');
$sns = $aws->get('v1.sns');
$sns->set_region(AmazonSNS::REGION_US_W2);

// Get today's flavors from DynamoDB using Version 2 of the SDK
$date = new DateTime();

$result = $ddb->getItem(array(
    'TableName' => 'flavors-of-the-day',
    'Key' => array(
        'HashKeyElement'  => array('N' => $date->format('n')),
        'RangeKeyElement' => array('N' => $date->format('j'))
    )
));
$flavors = $result->getPath('Item/flavors/SS');

// Generate the message
$today = $date->format('l, F jS');
$message = "It's {$today}, and here are our flavors of the day:\n";
foreach ($flavors as $flavor) {
    $message .= "- {$flavor}\n";
}
$message .= "\nCome visit Mr. Foo\'s Ice Cream Parlor on 5th and Pine!\n";
echo "{$message}\n";

// Send today's flavors to subscribers using Version 1 of the SDK
$response = $sns->publish('flavors-of-the-day-sns-topic', $message, array(
    'Subject' => 'Flavors of the Day - Mr. Foo\'s Ice Cream Parlor'
));
```

```
if ($response->isOK()) {
    echo "Sent the flavors of the day to your subscribers.\n";
} else {
    echo "There was an error sending the flavors of the day to your sub
scribers.\n";
}
```

Final Notes

Remember that **instantiating clients from the original SDK using the service builder from AWS SDK for PHP 2 does not change how those clients work**. For example, notice the differences in response handling between SDK versions. For a full list of differences between the versions, please see the Migration Guide.

For more information about using the AWS SDK for PHP 2, see the Getting Started Guide. For information about using the original version of the SDK, please see the Version 1 API Documentation and Version 1 SDK README.