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.
Table of Contents

AWS SDK for PHP ................................................................................................................................ 1
Getting Started .................................................................................................................................. 1
SDK Guides ...................................................................................................................................... 1
Service-Specific Features ............................................................................................................. 1
Examples ....................................................................................................................................... 2
Reference ...................................................................................................................................... 2
API Documentation ..................................................................................................................... 2
Maintenance and support for SDK major versions ....................................................................... 2
Getting Started ............................................................................................................................... 4
Prerequisites .................................................................................................................................. 4
  Requirements ................................................................................................................................. 4
  Recommendations ......................................................................................................................... 4
  Compatibility Test .......................................................................................................................... 5
Installing the SDK ........................................................................................................................... 5
  Install AWS SDK for PHP as a dependency via Composer .......................................................... 5
  Installing by Using the Packaged Phar ......................................................................................... 6
  Installing by Using the ZIP file ...................................................................................................... 6
Basic Usage .................................................................................................................................... 7
  Prerequisites ................................................................................................................................. 7
  Including the SDK in Your Code .................................................................................................... 7
  Usage Summary ............................................................................................................................. 7
  Creating a Client ............................................................................................................................. 7
  Using the Sdk Class ....................................................................................................................... 8
  Executing Service Operations ..................................................................................................... 9
Asynchronous Requests .................................................................................................................. 10
Working with Result Objects ......................................................................................................... 11
Handling Errors ............................................................................................................................. 12
Upgrading from Version 2 .............................................................................................................. 14
  Introduction .................................................................................................................................. 14
  What’s New in Version 3? ............................................................................................................. 14
  What’s Different from Version 2? .................................................................................................. 15
  Comparing Code Samples from Both Versions of the SDK .......................................................... 20
Configuring the SDK ..................................................................................................................... 23
Configuration Options ................................................................................................................... 23
  api_provider ................................................................................................................................. 24
  credentials .................................................................................................................................. 24
  debug .......................................................................................................................................... 25
  stats ............................................................................................................................................ 27
  endpoint ..................................................................................................................................... 28
  endpoint_provider ......................................................................................................................... 28
  endpoint_discovery ....................................................................................................................... 29
  handler ....................................................................................................................................... 30
  http .......................................................................................................................................... 30
  http_handler ................................................................................................................................. 36
  profile ......................................................................................................................................... 37
  region .......................................................................................................................................... 37
  retries .......................................................................................................................................... 37
  scheme .......................................................................................................................................... 39
  service ......................................................................................................................................... 40
  signature_provider ....................................................................................................................... 40
  signature_version ......................................................................................................................... 40
  ua_append .................................................................................................................................. 40
  use_aws_shared_config_files ...................................................................................................... 41
  validate ....................................................................................................................................... 41
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity and Access Management</td>
<td>292</td>
</tr>
<tr>
<td>Compliance Validation</td>
<td>292</td>
</tr>
<tr>
<td>Resilience</td>
<td>293</td>
</tr>
<tr>
<td>Infrastructure Security</td>
<td>293</td>
</tr>
<tr>
<td>S3 Encryption Client Migration</td>
<td>293</td>
</tr>
<tr>
<td>Migration Overview</td>
<td>293</td>
</tr>
<tr>
<td>Update Existing Clients to Read New Formats</td>
<td>294</td>
</tr>
<tr>
<td>Migrate Encryption and Decryption Clients to V2</td>
<td>294</td>
</tr>
<tr>
<td>Migration Examples</td>
<td>295</td>
</tr>
<tr>
<td>FAQ</td>
<td>297</td>
</tr>
<tr>
<td>What methods are available on a client?</td>
<td>297</td>
</tr>
<tr>
<td>What do I do about a cURL SSL certificate error?</td>
<td>297</td>
</tr>
<tr>
<td>What API versions are available for a client?</td>
<td>297</td>
</tr>
<tr>
<td>What Region versions are available for a client?</td>
<td>297</td>
</tr>
<tr>
<td>Why can’t I upload or download files larger than 2 GB?</td>
<td>298</td>
</tr>
<tr>
<td>How can I see what data is sent over the wire?</td>
<td>298</td>
</tr>
<tr>
<td>How can I set arbitrary headers on a request?</td>
<td>298</td>
</tr>
<tr>
<td>How can I sign an arbitrary request?</td>
<td>298</td>
</tr>
<tr>
<td>How can I modify a command before sending it?</td>
<td>298</td>
</tr>
<tr>
<td>What is a CredentialsException?</td>
<td>299</td>
</tr>
<tr>
<td>Does the AWS SDK for PHP work on HHVM?</td>
<td>299</td>
</tr>
<tr>
<td>How do I disable SSL?</td>
<td>299</td>
</tr>
<tr>
<td>What do I do about a “Parse error”?</td>
<td>299</td>
</tr>
<tr>
<td>Why is the Amazon S3 client decompressing gzipped files?</td>
<td>300</td>
</tr>
<tr>
<td>How do I disable body signing in Amazon S3?</td>
<td>300</td>
</tr>
<tr>
<td>How is retry scheme handled in the AWS SDK for PHP?</td>
<td>300</td>
</tr>
<tr>
<td>How do I handle exceptions with error codes?</td>
<td>301</td>
</tr>
<tr>
<td>Glossary</td>
<td>302</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>304</td>
</tr>
<tr>
<td>PHP SDK Forum</td>
<td>304</td>
</tr>
<tr>
<td>PHP SDK v3 and Developer Guide on GitHub</td>
<td>304</td>
</tr>
<tr>
<td>PHP SDK on Gitter</td>
<td>304</td>
</tr>
<tr>
<td>Document History</td>
<td>305</td>
</tr>
</tbody>
</table>
What Is the AWS SDK for PHP Version 3?

The AWS SDK for PHP Version 3 enables PHP developers to use Amazon Web Services in their PHP code, and build robust applications and software using services like Amazon S3, Amazon DynamoDB, S3 Glacier, etc. You can get started in minutes by installing the SDK through Composer — by requiring the `aws/aws-sdk-php` package — or by downloading the standalone `aws.zip` or `aws.phar` file.

Not all services are immediately available in the SDK. To find out which services are currently supported by the AWS SDK for PHP, see Service Name and API Version. For information about the AWS SDK for PHP Version 3 on GitHub, see Additional Resources (p. 304).

Note
If you’re migrating your code from using Version 2 of the SDK to Version 3, be sure to read Upgrading from Version 2 of the AWS SDK for PHP (p. 14).

Getting Started

- Requirements and Recommendations for the AWS SDK for PHP Version 3 (p. 4)
- Installing the AWS SDK for PHP Version 3 (p. 5)
- Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7)
- Upgrading from Version 2 of the AWS SDK for PHP (p. 14)

SDK Guides

- Configuration for the AWS SDK for PHP (p. 23)
- Credentials for the AWS SDK for PHP (p. 42)
- Command Objects in the AWS SDK for PHP (p. 58)
- Promises in the AWS SDK for PHP (p. 63)
- Handlers and Middleware in the AWS SDK for PHP (p. 68)
- Streams in the AWS SDK for PHP (p. 75)
- Paginators in the AWS SDK for PHP (p. 78)
- Waiters in the AWS SDK for PHP (p. 80)
- JMESPath Expressions in the AWS SDK for PHP (p. 83)
- SDK Metrics (p. 87)

Service-Specific Features

- Using AWS Cloud9 with the AWS SDK for PHP (p. 96)
- Using the DynamoDB Session Handler with AWS SDK for PHP (p. 97)
- Amazon S3 Multi-Region Client (p. 102)
- Amazon S3 Stream Wrapper (p. 103)
Examples

- Amazon S3 Transfer Manager (p. 108)
- Amazon S3 Client Side Encryption (p. 111)
- Amazon CloudFront (p. 116)
- Signing Custom Amazon CloudSearch Domain Requests (p. 136)
- Amazon CloudWatch Examples (p. 137)
- Amazon EC2 (p. 154)
- Signing an Amazon Elasticsearch Service Search Request (p. 164)
- AWS Identity and Access Management Examples (p. 165)
- AWS Key Management Service (p. 183)
- Amazon Kinesis Examples (p. 199)
- AWS Elemental MediaConvert Examples (p. 211)
- Amazon S3 Examples (p. 218)
- Amazon Simple Email Services Examples (p. 245)
- Amazon SNS Examples (p. 268)
- Amazon SQS Examples (p. 281)

For additional examples, see the AWS Code Sample Catalog.

Reference

- FAQ (p. 297)
- Glossary (p. 302)
- Contributing to the SDK
- Guzzle Documentation

Submit issues on GitHub:

- Submit documentation issues
- Report a bug or request a feature

API Documentation

Find API documentation for the SDK at https://docs.aws.amazon.com/sdk-for-php/latest/reference/.

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Shared Configuration and Credentials Reference Guide:

- AWS SDKs and Tools Maintenance Policy
• AWS SDKs and Tools Version Support Matrix
Getting Started with the AWS SDK for PHP Version 3

This chapter is dedicated to getting you up and running with the AWS SDK for PHP Version 3.

Topics

- Requirements and Recommendations for the AWS SDK for PHP Version 3 (p. 4)
- Installing the AWS SDK for PHP Version 3 (p. 5)
- Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7)
- Upgrading from Version 2 of the AWS SDK for PHP (p. 14)

Requirements and Recommendations for the AWS SDK for PHP Version 3

For best results with AWS SDK for PHP, ensure your environment supports the following requirements and recommendations.

Requirements

To use the AWS SDK for PHP, you must be using PHP version 5.5.0 or later with the SimpleXML PHP extension enabled. If you need to sign private Amazon CloudWatch URLs, you also need the OpenSSL PHP extension.

Recommendations

In addition to the minimum requirements, we recommend you also install, uninstall, and use the following.

<table>
<thead>
<tr>
<th>Install cURL 7.16.2 or later</th>
<th>Use a recent version of cURL compiled with OpenSSL/NSS and zlib. If cURL isn't installed on your system and you don't configure a custom http_handler for your client, the SDK uses the PHP stream wrapper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use OPCache</td>
<td>Use the OPcache extension to improve PHP performance by storing precompiled script bytecode in shared memory. This removes the need for PHP to load and parse scripts on each request. This extension is typically enabled by default.</td>
</tr>
<tr>
<td></td>
<td>When running Amazon Linux, you need to install the php56-opcache or php55-opcache yum package to use the OPCODE extension.</td>
</tr>
</tbody>
</table>
Uninstall Xdebug

Xdebug can help identify performance bottlenecks. However, if performance is critical to your application, don't install the Xdebug extension in your production environment. Loading the extension slows SDK performance considerably.

Use a Composer classmap autoloader

Autoloaders load classes as they are required by a PHP script. Composer generates an autoloader that can autoload the PHP scripts of your application and all other PHP scripts required by your application, including the AWS SDK for PHP.

For production environments, we recommend you use a classmap autoloader to improve autoloader performance. You can generate a classmap autoloader by passing the `--optimize-autoloader` option to Composer's install command.

Compatibility Test

Run the `compatibility-test.php` file in the SDK to verify your system can run the SDK. In addition to meeting the SDK’s minimum system requirements, the compatibility test checks for optional settings and makes recommendations that can help improve performance. The compatibility test outputs results either to the command line or a web browser. When reviewing test results in a browser, successful checks appear in green, warnings in purple, and failures in red. When running from the command line, the result of a check appears on a separate line.

When reporting an issue with the SDK, sharing the output of the compatibility test helps identify the underlying cause.

Installing the AWS SDK for PHP Version 3

You can install the AWS SDK for PHP Version 3:

- As a dependency via Composer
- As a prepackaged phar of the SDK
- As a ZIP file of the SDK

Before you install AWS SDK for PHP Version 3 ensure your environment is using PHP version 5.5 or later. Learn more about environment requirements and recommendations (p. 4).

**Note**

Installing the SDK via the .phar and .zip methods requires the Multibyte String PHP extension to be installed and enabled separately.

Install AWS SDK for PHP as a dependency via Composer

Composer is the recommended way to install the AWS SDK for PHP. Composer is a tool for PHP that manages and installs the dependencies of your project.
For more information on how to install Composer, configure autoloading, and follow other best practices for defining dependencies, see getcomposer.org.

**Install Composer**

If Composer is not already in your project, download and install Composer.

For **Windows**, download and run the Composer-Setup.exe.

For **Linux**, follow the Command-line installation on the Download Composer page.

**Add AWS SDK for PHP as a dependency via Composer**

If Composer is already installed globally on your system, run the following in the base directory of your project to install AWS SDK for PHP as a dependency:

```
composer require aws/aws-sdk-php
```

Otherwise type this Composer command to install the latest version of the AWS SDK for PHP as a dependency.

```
php -d memory_limit=-1 composer.phar require aws/aws-sdk-php
```

**Add autoloader to your php scripts**

To utilize the AWS SDK for PHP in your scripts, include the autoloader in your scripts, as follows.

```php
<?php
    require '/path/to/vendor/autoload.php';
?>
```

**Installing by Using the Packaged Phar**

Each release of the AWS SDK for PHP includes a prepackaged phar (PHP archive) that contains all the classes and dependencies you need to run the SDK. Additionally, the phar automatically registers a class autoloader for the AWS SDK for PHP and all its dependencies.

You can download the packaged phar and include it in your scripts.

```php
<?php
    require '/path/to/aws.phar';
?>
```

**Note**

Using PHP with the Suhosin patch is not recommended, but is common on Ubuntu and Debian distributions. In this case, you might need to enable the use of phars in the suhosin.ini. If you don’t do this, including a phar file in your code will cause a silent failure. To modify suhosin.ini, add the following line.

```
suhosin.executor.include.whitelist = phar
```

**Installing by Using the ZIP file**

The AWS SDK for PHP includes a ZIP file containing all the classes and dependencies you need to run the SDK. Additionally, the ZIP file includes a class autoloader for the AWS SDK for PHP and its dependencies.
To install the SDK, download the .zip file, and then extract it into your project at a location you choose. Then include the autoloader in your scripts, as follows.

```php
<?php
    require '/path/to/aws-autoloader.php';
?>
```

Basic Usage Patterns of the AWS SDK for PHP
Version 3

This topic focuses on basic usage patterns of the AWS SDK for PHP.

**Prerequisites**

- Download and installed the SDK (p. 5)
- Retrieve your AWS access keys.

**Including the SDK in Your Code**

No matter which technique you used to install the SDK, you can include the SDK in your code with just a single `require` statement. See the following table for the PHP code that best fits your installation technique. Replace any instances of `/path/to/` with the actual path on your system.

<table>
<thead>
<tr>
<th>Installation Technique</th>
<th>Require Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Composer</td>
<td><code>require '/path/to/vendor/autoload.php';</code></td>
</tr>
<tr>
<td>Using the phar</td>
<td><code>require '/path/to/aws.phar';</code></td>
</tr>
<tr>
<td>Using the ZIP</td>
<td><code>require '/path/to/aws-autoloader.php';</code></td>
</tr>
</tbody>
</table>

In this topic, we show examples that assume the Composer installation method. If you’re using a different installation method, you can refer back to this section to find the correct `require` code to use.

**Usage Summary**

To use the SDK to interact with an AWS service, instantiate a `Client` object. Client objects have methods that correspond one to one with operations in the service’s API. To execute a particular operation, you call its corresponding method. This method either returns an array-like `Result` object on success, or throws an `Exception` on failure.

**Creating a Client**

You can create a client by passing an associative array of options to a client's constructor.

**Imports**

```php
require 'vendor/autoload.php';
```
Sample Code

```php
// Create a S3Client
$s3 = new Aws\S3\S3Client([ 'version' => 'latest', 'region' => 'us-east-2' ]); 
```

Notice that we did not explicitly provide credentials to the client. That’s because the SDK should detect the credentials from environment variables (via AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY), an AWS credentials INI file in your HOME directory, AWS Identity and Access Management (IAM) instance profile credentials, or credential providers (p. 50).

All of the general client configuration options are described in detail in the configuration guide (p. 23). The array of options provided to a client can vary based on which client you’re creating. These custom client configuration options are described in the API documentation for each client.

Using the Sdk Class

The Aws\Sdk class acts as a client factory and is used to manage shared configuration options across multiple clients. The same options that can be provided to a specific client constructor can also be supplied to the Aws\Sdk class. These options are then applied to each client constructor.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
// The same options that can be provided to a specific client constructor can also be supplied to the Aws\Sdk class.
// Use the us-west-2 region and latest version of each client.
$sharedConfig = [
    'region' => 'us-west-2',
    'version' => 'latest'
];

// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk($sharedConfig);

// Create an Amazon S3 client using the shared configuration data.
$client = $sdk->createS3();
```

Options that are shared across all clients are placed in root-level key-value pairs. Service-specific configuration data can be provided in a key that is the same as the namespace of a service (e.g., “S3”, “DynamoDb”, etc.).

```php
$sdk = new Aws\Sdk([ 'region' => 'us-west-2', 'version' => 'latest', ];
```
Service-specific configuration values are a union of the service-specific values and the root-level values (i.e., service-specific values are shallow-merged onto root-level values).

**Note**
We highly recommended that you use the `Sdk` class to create clients if you're using multiple client instances in your application. The `Sdk` class automatically uses the same HTTP client for each SDK client, allowing SDK clients for different services to perform nonblocking HTTP requests. If the SDK clients don't use the same HTTP client, then HTTP requests sent by the SDK client might block promise orchestration between services.

### Executing Service Operations

You can execute a service operation by calling the method of the same name on a client object. For example, to perform the Amazon S3 `PutObject` operation, you must call the `Aws\S3\S3Client::putObject()` method.

### Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
```

### Sample Code

```php
// Use the us-east-2 region and latest version of each client.
$sharedConfig = [
    'profile' => 'default',
    'region' => 'us-east-2',
    'version' => 'latest'
];

// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk($sharedConfig);

// Use an Aws\Sdk class to create the S3Client object.
$s3Client = $sdk->createS3();

// Send a PutObject request and get the result object.
$result = $s3Client->putObject([  
    'Bucket' => 'my-bucket',
    'Key' => 'my-key',
    'Body' => 'this is the body!'  
]);

// Download the contents of the object.
$result = $s3Client->getObject([  
    'Bucket' => 'my-bucket',
    'Key' => 'my-key'  
]);

// Print the body of the result by indexing into the result object.
echo $result['Body'];
```
Operations available to a client and the structure of the input and output are defined at runtime based on a service description file. When creating a client, you must provide a version (e.g., "2006-03-01" or "latest"). The SDK finds the corresponding configuration file based on the provided version.

Operation methods like `putObject()` all accept a single argument, an associative array that represents the parameters of the operation. The structure of this array (and the structure of the result object) is defined for each operation in the SDK's API Documentation (e.g., see the API docs for `putObject` operation).

**HTTP Handler Options**

You can also fine-tune how the underlying HTTP handler executes the request by using the special `@http` parameter. The options you can include in the `@http` parameter are the same as the ones you can set when you instantiate the client with the "http" client option (p. 30).

```php
// Send the request through a proxy
$result = $s3Client->putObject(
    ['Bucket' => 'my-bucket',
     'Key'    => 'my-key',
     'Body'   => 'this is the body!',
     '@http'  => [
         'proxy' => 'http://192.168.16.1:10'
     ]
);```

**Asynchronous Requests**

You can send commands concurrently using the asynchronous features of the SDK. You can send requests asynchronously by suffixing an operation name with `Async`. This initiates the request and returns a promise. The promise is fulfilled with the result object on success or rejected with an exception on failure. This enables you to create multiple promises and have them send HTTP requests concurrently when the underlying HTTP handler transfers the requests.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk([  'region'   => 'us-west-2',
     'version' => 'latest'
]);
// Use an Aws\Sdk class to create the S3Client object.
$s3Client = $sdk->createS3();
//Listing all S3 Bucket
#CompleteSynchronously = $s3Client->listBucketsAsync();
// Block until the result is ready.
#CompleteSynchronously = #CompleteSynchronously->wait();```

You can force a promise to complete synchronously by using the `wait` method of the promise. Forcing the promise to complete also "unwraps" the state of the promise by default, meaning it will either
return the result of the promise or throw the exception that was encountered. When calling `wait()` on a promise, the process blocks until the HTTP request is completed and the result is populated or an exception is thrown.

When using the SDK with an event loop library, don't block on results. Instead, use the `then()` method of a result to access a promise that is resolved or rejected when the operation completes.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk([  
    'region' => 'us-west-2',  
    'version' => 'latest'
]);
// Use an Aws\Sdk class to create the S3Client object.
$s3Client = $sdk->createS3();

$promise = $s3Client->listBucketsAsync();
$promise
    ->then(function ($result) {  
        echo 'Got a result: ' . var_export($result, true);  
    })
    ->otherwise(function ($reason) {  
        echo 'Encountered an error: ' . $reason->getMessage();  
    });
```

Working with Result Objects

Executing a successful operation returns an `Aws\Result` object. Instead of returning the raw XML or JSON data of a service, the SDK coerces the response data into an associative array structure. It normalizes some aspects of the data based on its knowledge of the specific service and the underlying response structure.

You can access data from the `AWSResult` object like an associative PHP array.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
// Use the us-east-2 region and latest version of each client.
$sharedConfig = [
    'profile' => 'default',
    'region' => 'us-east-2',
    'version' => 'latest'
];
// Create an SDK class used to share configuration across clients.
```
$sdk = new Aws\Sdk($sharedConfig);

// Use an Aws\Sdk class to create the S3Client object.
$s3 = $sdk->createS3();
$result = $s3->listBuckets();
foreach ($result['Buckets'] as $bucket) {
    echo $bucket['Name'] . "\n";
}

// Convert the result object to a PHP array
$array = $result->toArray();

The contents of the result object depend on the operation that was executed and the version of a service. The result structure of each API operation is documented in the API docs for each operation.

The SDK is integrated with JMESPath, a DSL used to search and manipulate JSON data or, in our case, PHP arrays. The result object contains a `search()` method you can use to more declaratively extract data from the result.

Sample Code

```php
$s3 = $sdk->createS3();
$result = $s3->listBuckets();

$names = $result->search('Buckets[].Name');
```

### Handling Errors

#### Synchronous Error Handling

If an error occurs while performing an operation, an exception is thrown. For this reason, if you need to handle errors in your code, use `try`/`catch` blocks around your operations. The SDK throws service-specific exceptions when an error occurs.

The following example uses the `Aws\S3\S3Client`. If there is an error, the exception thrown will be of the type `Aws\S3\Exception\S3Exception`. All service-specific exceptions that the SDK throws extend from the `Aws\Exception\AwsException` class. This class contains useful information about the failure, including the request-id, error code, and error type. Note for some services which support it, response data is coerced into an associative array structure (similar to `Aws\Result` objects), which can be accessed like a normal PHP associative array. The `toArray()` method will return any such data, if it exists.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\Exception\S3Exception;
```

**Sample Code**

```php
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk(['
    'region' => 'us-west-2',
    'version' => 'latest'
]);
```
// Use an Aws\Sdk class to create the S3Client object.
$s3Client = $sdk->createS3();

try {
    $s3Client->createBucket(['Bucket' => 'my-bucket']);
} catch (S3Exception $e) {
    // Catch an S3 specific exception.
    echo $e->getMessage();
} catch (AwsException $e) {
    // This catches the more generic AwsException. You can grab information
    // from the exception using methods of the exception object.
    echo $e->getAwsRequestId() . "\n";
    echo $e->getAwsErrorType() . "\n";
    echo $e->getAwsErrorCode() . "\n";

    // This dumps any modeled response data, if supported by the service
    // Specific members can be accessed directly (e.g. $e['MemberName'])
    var_dump($e->toArray());
}

Asynchronous Error Handling

Exceptions are not thrown when sending asynchronous requests. Instead, you must use the then() or
otherwise() method of the returned promise to receive the result or error.

Imports

require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\Exception\S3Exception;

Sample Code

//Asynchronous Error Handling

$promise = $s3Client->createBucketAsync(['Bucket' => 'my-bucket']);
$promise->otherwise(function ($reason) {
    var_dump($reason);
});

// This does the same thing as the "otherwise" function.
$promise->then(null, function ($reason) {
    var_dump($reason);
});

You can “unwrap” the promise and cause the exception to be thrown instead.

Imports
use Aws\S3\Exception\S3Exception;

Sample Code

$promise = $s3Client->createBucketAsync(['Bucket' => 'my-bucket']);

//throw exception
try {
    $result = $promise->wait();
} catch (S3Exception $e) {
    echo $e->getMessage();
}

Upgrading from Version 2 of the AWS SDK for PHP

This topic shows how to migrate your code to use version 3 of the AWS SDK for PHP and how the new version differs from version 2 of the SDK.

Note
The basic usage pattern of the SDK (i.e., $result = $client->operation($params);) has not changed from version 2 to version 3, which should result in a smooth migration.

Introduction

Version 3 of the AWS SDK for PHP represents a significant effort to improve the capabilities of the SDK, incorporate over two years of customer feedback, upgrade our dependencies, improve performance, and adopt the latest PHP standards.

What’s New in Version 3?

Version 3 of the AWS SDK for PHP follows the PSR-4 and PSR-7 standards and will follow the SemVer standard going forward.

Other new features include

- Middleware system for customizing service client behavior
- Flexible paginators for iterating through paginated results
- Ability to query data from result and paginator objects with JMESPath
- Easy debugging via the 'debug' configuration option

Decoupled HTTP layer

- Guzzle 6 is used by default to send requests, but Guzzle 5 is also supported.
- The SDK will work in environments where cURL is not available.
- Custom HTTP handlers are also supported.

Asynchronous requests

- Features like waiters and multipart uploaders can also be used asynchronously.
• Asynchronous workflows can be created using promises and coroutines.
• Performance of concurrent or batched requests is improved.

What’s Different from Version 2?

Project Dependencies are Updated

The dependencies of the SDK have changed in this version.

• The SDK now requires PHP 5.5+. We use generators liberally within the SDK code.
• We’ve upgraded the SDK to use Guzzle 6 (or 5), which provides the underlying HTTP client implementation used by the SDK to send requests to the AWS services. The latest version of Guzzle brings with it a number of improvements, including asynchronous requests, swappable HTTP handlers, PSR-7 compliance, better performance, and more.
• The PSR-7 package from the PHP-FIG (psr/http-message) defines interfaces for representing HTTP requests, HTTP responses, URLs, and streams. These interfaces are used across the SDK and Guzzle, which provides interoperability with other PSR-7 compliant packages.
• Guzzle's PSR-7 implementation (guzzlehttp/psr7) provides an implementation of the interfaces in PSR-7, and several helpful classes and functions. Both the SDK and Guzzle 6 rely on this package heavily.
• Guzzle's Promises/A+ implementation (guzzlehttp/promises) is used throughout the SDK and Guzzle to provide interfaces for managing asynchronous requests and coroutines. While Guzzle's multi-cURL HTTP handler ultimately implements the non-blocking I/O model that allows for asynchronous requests, this package provides the ability to program within that paradigm. See Promises in the AWS SDK for PHP Version 3 (p. 63) for more details.
• The PHP implementation of JMESPath (mtdowling/jmespath.php) is used in the SDK to provide the data querying ability of the Aws\Result::search() and Aws\ResultPaginator::search() methods. See JMESPath Expressions in the AWS SDK for PHP Version 3 (p. 83) for more details.

Region and Version Options Are Now Required

When instantiating a client for any service, specify the ‘region’ and ‘version’ options. In version 2 of the AWS SDK for PHP, ‘version’ was completely optional, and ‘region’ was sometimes optional. In version 3, both are always required. Being explicit about both of these options allows you to lock into the API version and AWS Region you are coding against. When new API versions are created or new AWS Regions become available, you will be isolated from potentially breaking changes until you are ready to explicitly update your configuration.

Note
If you’re not concerned about which API version you are using, you can just set the ‘version’ option to ‘latest’. However, we recommend that you set the API version numbers explicitly for production code.

Not all services are available in all AWS Regions. You can find a list of available Regions using the Regions and Endpoints reference.

For services that are available only via a single, global endpoint (e.g., Amazon Route 53, AWS Identity and Access Management, and Amazon CloudFront), instantiate clients with their configured Region set to us-east-1.

Important
The SDK also includes multi-region clients, which can dispatch requests to different AWS Regions based on a parameter (@region) supplied as a command parameter. The Region used by default by these clients is specified with the region option supplied to the client constructor.
Client Instantiation Uses the Constructor

In version 3 of the AWS SDK for PHP, the way you instantiate a client has changed. Instead of the `factory` methods in version 2, you can simply instantiate a client by using the `new` keyword.

```php
use Aws\DynamoDb\DynamoDbClient;

// Version 2 style
$client = DynamoDbClient::factory([  
    'region'  => 'us-east-2'
]);

// Version 3 style
$client = new DynamoDbClient([  
    'region'  => 'us-east-2',  
    'version' => '2012-08-10'
]);
```

**Note**

Instantiating a client using the `factory()` method still works. However, it's considered deprecated.

Client Configuration Has Changed

The client configuration options in version 3 of the AWS SDK for PHP have changed a little from version 2. See the Configuration for the AWS SDK for PHP Version 3 (p. 23) page for a description of all supported options.

**Important**

In version 3, 'key' and 'secret' are no longer valid options at the root level, but you can pass them in as part of the 'credentials' option. One reason we made this was to discourage developers from hard-coding their AWS credentials into their projects.

The Sdk Object

Version 3 of the AWS SDK for PHP introduces the `Aws\Sdk` object as a replacement to `Aws\Common\Aws`. The `Sdk` object acts as a client factory and is used to manage shared configuration options across multiple clients.

Although the `Aws` class in version 2 of the SDK worked like a service locator (it always returned the same instance of a client), the `Sdk` class in version 3 returns a new instance of a client every time it's used.

The `Sdk` object also doesn't support the same configuration file format from version 2 of the SDK. That configuration format was specific to Guzzle 3 and is now obsolete. Configuration can be done more simply with basic arrays, and is documented in Using the Sdk Class (p. 8).

Some API Results Have Changed

To provide consistency in how the SDK parses the result of an API operation, Amazon ElastiCache, Amazon RDS, and Amazon Redshift now have an additional wrapping element on some API responses.

For example, calling the Amazon RDS `DescribeEngineDefaultParameters` result in version 3 now includes a wrapping "EngineDefaults" element. In version 2, this element was not present.

```php
$client = new Aws\Rds\RdsClient([  
    'region'  => 'us-west-1',  
    'version' => '2014-09-01'
]);
```
The following operations are affected and now contain a wrapping element in the output of the result (provided below in parentheses):

- **Amazon ElastiCache**
  - AuthorizeCacheSecurityGroupIngress (CacheSecurityGroup)
  - CopySnapshot (Snapshot)
  - CreateCacheCluster (CacheCluster)
  - CreateCacheParameterGroup (CacheParameterGroup)
  - CreateCacheSecurityGroup (CacheSecurityGroup)
  - CreateCacheSubnetGroup (CacheSubnetGroup)
  - CreateReplicationGroup (ReplicationGroup)
  - CreateSnapshot (Snapshot)
  - DeleteCacheCluster (CacheCluster)
  - DeleteReplicationGroup (ReplicationGroup)
  - DeleteSnapshot (Snapshot)
  - DescribeEngineDefaultParameters (EngineDefaults)
  - ModifyCacheCluster (CacheCluster)
  - ModifyCacheSubnetGroup (CacheSubnetGroup)
  - ModifyReplicationGroup (ReplicationGroup)
  - PurchaseReservedCacheNodesOffering (ReservedCacheNode)
  - RebootCacheCluster (CacheCluster)
  - RevokeCacheSecurityGroupIngress (CacheSecurityGroup)

- **Amazon RDS**
  - AddSourceIdentifierToSubscription (EventSubscription)
  - AuthorizeDBSecurityGroupIngress (DBSecurityGroup)
  - CopyDBParameterGroup (DBParameterGroup)
  - CopyDBSnapshot (DBSnapshot)
  - CopyOptionGroup (OptionGroup)
  - CreateDBInstance (DBInstance)
  - CreateDBInstanceReadReplica (DBInstance)
  - CreateDBParameterGroup (DBParameterGroup)
  - CreateDBSecurityGroup (DBSecurityGroup)
  - CreateDBSnapshot (DBSnapshot)
  - CreateDBSubnetGroup (DBSubnetGroup)
  - CreateEventSubscription (EventSubscription)
  - CreateOptionGroup (OptionGroup)
  - DeleteDBInstance (DBInstance)
  - DeleteDBSnapshot (DBSnapshot)
• DeleteEventSubscription (EventSubscription)
• DescribeEngineDefaultParameters (EngineDefaults)
• ModifyDBInstance (DBInstance)
• ModifyDBSubnetGroup (DBSubnetGroup)
• ModifyEventSubscription (EventSubscription)
• ModifyOptionGroup (OptionGroup)
• PromoteReadReplica (DBInstance)
• PurchaseReservedDBInstancesOffering (ReservedDBInstance)
• RebootDBInstance (DBInstance)
• RemoveSourceIdentifierFromSubscription (EventSubscription)
• RestoreDBInstanceFromDBSnapshot (DBInstance)
• RestoreDBInstanceToPointInTime (DBInstance)
• RevokeDBSecurityGroupIngress (DBSecurityGroup)

• Amazon Redshift
  • AuthorizeClusterSecurityGroupIngress (ClusterSecurityGroup)
  • AuthorizeSnapshotAccess (Snapshot)
  • CopyClusterSnapshot (Snapshot)
  • CreateCluster (Cluster)
  • CreateClusterParameterGroup (ClusterParameterGroup)
  • CreateClusterSecurityGroup (ClusterSecurityGroup)
  • CreateClusterSnapshot (Snapshot)
  • CreateClusterSubnetGroup (ClusterSubnetGroup)
  • CreateEventSubscription (EventSubscription)
  • CreateHsmClientCertificate (HsmClientCertificate)
  • CreateHsmConfiguration (HsmConfiguration)
  • DeleteCluster (Cluster)
  • DeleteClusterSnapshot (Snapshot)
  • DescribeDefaultClusterParameters (DefaultClusterParameters)
  • DisableSnapshotCopy (Cluster)
  • EnableSnapshotCopy (Cluster)
  • ModifyCluster (Cluster)
  • ModifyClusterSubnetGroup (ClusterSubnetGroup)
  • ModifyEventSubscription (EventSubscription)
  • ModifySnapshotCopyRetentionPeriod (Cluster)
  • PurchaseReservedNodeOffering (ReservedNode)
  • RebootCluster (Cluster)
  • RestoreFromClusterSnapshot (Cluster)
  • RevokeClusterSecurityGroupIngress (ClusterSecurityGroup)
  • RevokeSnapshotAccess (Snapshot)
  • RotateEncryptionKey (Cluster)

Enum Classes Have Been Removed

We have removed the Enum classes (e.g., Aws\S3\Enum\CannedAcl) that existed in version 2 of the AWS SDK for PHP. Enums were concrete classes within the public API of the SDK that contained constants representing groups of valid parameter values. Because these enums are specific to API
versions, can change over time, can conflict with PHP reserved words, and ended up not being very useful, we have removed them in version 3. This supports the data-driven and API version agnostic nature of version 3.

Instead of using values from Enum objects, use the literal values directly (e.g., CannedAcl::PUBLIC_READ → 'public-read').

**Fine-Grained Exception Classes Have Been Removed**

We have removed the fine-grained exception classes that existed in each service's namespaces (e.g., Aws\Rds\Exception\{SpecificError}Exception) for very similar reasons that we removed Enums. The exceptions thrown by a service or operation are dependent on which API version is used (they can change from version to version). Also, the complete list of the exceptions that can be thrown by a given operation is not available, which made version 2's fine-grained exception classes incomplete.

Handle errors by catching the root exception class for each service (e.g., Aws\Rds\Exception\RdsException). You can use the getAwsErrorCode() method of the exception to check for specific error codes. This is functionally equivalent to catching different exception classes, but provides that function without adding bloat to the SDK.

**Static Facade Classes Have Been Removed**

In version 2 of the AWS SDK for PHP, there was an obscure feature inspired by Laravel that allowed you to call enableFacades() on the Aws class to enable static access to the various service clients. This feature goes against PHP best practices, and we stopped documenting it over a year ago. In version 3, this feature is removed completely. Retrieve your client objects from the Aws\Sdk object and use them as object instances, not static classes.

**Paginators Supersede iterators**

Version 2 of the AWS SDK for PHP had a feature named *iterators*. These were objects that were used for iterating over paginated results. One complaint we had about these was that they were not flexible enough, because the iterator only emitted specific values from each result. If there were other values you needed from the results, you could only retrieve them via event listeners.

In version 3, iterators have been replaced with **Paginators** (p. 78). Their purpose is similar, but paginators are more flexible. This is because they yield result objects instead of values from a response.

The following examples show how paginators are different from iterators, by demonstrating how to retrieve paginated results for the S3 ListObjects operation in both version 2 and version 3.

```php
// Version 2
$objects = $s3Client->getIterator('ListObjects', ['Bucket' => 'my-bucket']);
foreach ($objects as $object) {
    echo $object['Key'] . "\n";
}
```

```php
// Version 3
$results = $s3Client->getPaginator('ListObjects', ['Bucket' => 'my-bucket']);
foreach ($results as $result) {
    // You can extract any data that you want from the result.
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
}
```

Paginators supersede iterators.
Comparing Code Samples from Both Versions of the SDK

The following examples show some of the ways in which using version 3 of the AWS SDK for PHP might differ from version 2.

Example: Amazon S3 ListObjects Operation

From Version 2 of the SDK

```php
<?php
require '/path/to/vendor/autoload.php';

$results = $s3Client->getPaginator('ListObjects', ['Bucket' => 'my-bucket']);
foreach ($results->search('Contents[].Key') as $key) {
    echo $key . "\n";
}
```
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$s3 = S3Client::factory(
    ['profile' => 'my-credential-profile',
     'region' => 'us-east-1'])
);

try {
    $result = $s3->listObjects(
        ['Bucket' => 'my-bucket-name',
         'Key' => 'my-object-key'])
   );
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}

From Version 3 of the SDK

Key differences:

• Use new instead of factory() to instantiate the client.
• The 'version' and 'region' options are required during instantiation.

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

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$s3 = new S3Client(
    ['profile' => 'my-credential-profile',
     'region' => 'us-east-1',
     'version' => '2006-03-01'])
);

try {
    $result = $s3->listObjects(
        ['Bucket' => 'my-bucket-name',
         'Key' => 'my-object-key'])
   );
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}

Example: Instantiating a Client with global Configuration

From Version 2 of the SDK

<?php return array(
From Version 3 of the SDK

Key differences:

- Use the Aws\Sdk class instead of Aws\Common\Aws.
- There’s no configuration file. Use an array for configuration instead.
- The ‘version’ option is required during instantiation.
- Use the createService() methods instead of get('<service>').
Configuring the AWS SDK for PHP Version 3

The AWS SDK for PHP consists of various features and components. Each of the following topics describe the components that are used in the SDK.

Topics
- Configuration for the AWS SDK for PHP Version 3 (p. 23)
- Credentials for the AWS SDK for PHP Version 3 (p. 42)
- Command Objects in the AWS SDK for PHP Version 3 (p. 58)
- Promises in the AWS SDK for PHP Version 3 (p. 63)
- Handlers and Middleware in the AWS SDK for PHP Version 3 (p. 68)
- Streams in the AWS SDK for PHP Version 3 (p. 75)
- Paginators in the AWS SDK for PHP Version 3 (p. 78)
- Waiters in the AWS SDK for PHP Version 3 (p. 80)
- JMESPath Expressions in the AWS SDK for PHP Version 3 (p. 83)
- SDK Metrics in the AWS SDK for PHP Version 3 (p. 87)

Configuration for the AWS SDK for PHP Version 3

This guide describes client constructor options. These options can be provided in a client constructor or provided to the `Aws\Sdk` class. The array of options provided to a specific type of client can vary, based on which client you are creating. These custom client configuration options are described in the API documentation of each client.

Note that some configuration options will check and use default values based on environment variables or an AWS configuration file. By default, the configuration file being checked will be `.aws/config` in your home directory, commonly `~/.aws/config`. However, you can use the environment variable `AWS_CONFIG_FILE` to set where your default config file location is. This may be especially useful if you are restricting file access to certain directories with `open_basedir` and the like.

Configuration Options
- `api_provider` (p. 24)
- `credentials` (p. 24)
- `debug` (p. 25)
- `stats` (p. 27)
- `endpoint` (p. 28)
- `endpoint_provider` (p. 28)
- `endpoint_discovery` (p. 29)
- `handler` (p. 30)
- `http` (p. 30)
- `http_handler` (p. 36)
The following example shows how to pass options into an Amazon S3 client constructor.

```php
use Aws\S3\S3Client;

$options = [
    'region'            => 'us-west-2',
    'version'           => '2006-03-01',
    'signature_version' => 'v4'
];

$s3Client = new S3Client($options);
```

See the basic usage guide (p. 7) for more information about constructing clients.

**api_provider**

Type

callable

A PHP callable that accepts a type, service, and version argument, and returns an array of corresponding configuration data. The type value can be one of api, waiter, or paginator.

By default, the SDK uses an instance of Aws\Api\FileSystemApiProvider that loads API files from the src/data folder of the SDK.

**credentials**

Type

array|Aws\CacheInterface|Aws\Credentials\CredentialsInterface|bool|callable

Pass an Aws\Credentials\CredentialsInterface object to use a specific credentials instance.

```php
$credentials = new Aws\Credentials\Credentials('key', 'secret');

$s3 = new Aws\S3\S3Client([ 
    'version' => 'latest',
    'region'  => 'us-west-2',
    'credentials' => $credentials
]);
```
If you don’t provide a `credentials` option, the SDK attempts to load credentials from your environment in the following order:

1. Load credentials from environment variables (p. 44).
2. Load credentials from a `credentials.ini` file (p. 44).
3. Load credentials from an IAM role (p. 46).

Pass `false` to use null credentials and not sign requests.

```php
$s3 = new Aws\S3\S3Client(
    [
        'version' => 'latest',
        'region'  => 'us-west-2',
        'credentials' => false
    ]);
```

Pass a callable credential provider (p. 50) function to create credentials using a function.

```php
use Aws\Credentials\CredentialProvider;

// Only load credentials from environment variables
$provider = CredentialProvider::env();

$s3 = new Aws\S3\S3Client(
    [
        'version' => 'latest',
        'region'  => 'us-west-2',
        'credentials' => $provider
    ]);
```

Pass credentials cached to an instance of `Aws\CacheInterface` to cache the values returned by the default provider chain across multiple processes.

```php
use Aws\Credentials\CredentialProvider;
use Aws\DoctrineCacheAdapter;
use Doctrine\Common\Cache\ApcuCache;

$cache = new DoctrineCacheAdapter(new ApcuCache);
$provider = CredentialProvider::defaultProvider();
$cachedProvider = CredentialProvider::cache($provider, $cache);

$s3 = new Aws\S3\S3Client(
    [
        'version' => 'latest',
        'region'  => 'us-west-2',
        'credentials' => $cachedProvider
    ]);
```

You can find more information about providing credentials to a client in the Credentials for the AWS SDK for PHP Version 3 (p. 42) guide.

**Note**

Credentials are loaded and validated lazily when they are used.

### debug

**Type**

`bool|array`
Outputs debug information about each transfer. Debug information contains information about each state change of a transaction as it is prepared and sent over the wire. Also included in the debug output is information about the specific HTTP handler used by a client (e.g., debug cURL output).

Set to `true` to display debug information when sending requests.

```php
$s3 = new Aws\S3\S3Client([  'version' => 'latest',  'region'  => 'us-west-2',  'debug'   => true]);

// Perform an operation to see the debug output
$s3->listBuckets();
```

Alternatively, you can provide an associative array with the following keys.

**logfn (callable)**

Function that is invoked with log messages. By default, PHP’s `echo` function is used.

**stream_size (int)**

When the size of a stream is greater than this number, the stream data is not logged. Set to 0 to not log any stream data.

**scrub_auth (bool)**

Set to `false` to disable the scrubbing of auth data from the logged messages (meaning your AWS access key ID and signature will be passed through to the `logfn`).

**http (bool)**

Set to `false` to disable the "debug" feature of lower-level HTTP handlers (e.g., verbose cURL output).

**auth_headers (array)**

Set to a key-value mapping of headers you want to replace mapped to the value you want to replace them with. These values are not used unless `scrub_auth` is set to `true`.

**auth_strings (array)**

Set to a key-value mapping of regular expressions to map to their replacements. These values are used by the authentication data scrubber if `scrub_auth` is set to `true`.

```php
$s3 = new Aws\S3\S3Client([  'version' => 'latest',  'region'  => 'us-west-2',  'debug'   => [    'logfn'        => function ($msg) { echo $msg . "\n"; },    'stream_size'  => 0,    'scrub_auth'   => true,    'http'         => true,    'auth_headers' => [      'X-My-Secret-Header' => '[REDACTED]',    ],    'auth_strings' => [      '/SuperSecret=[A-Za-z0-9]{20}/i' => 'SuperSecret=[REDACTED]',    ]  ]);```
// Perform an operation to see the debug output
$s3->listBuckets();

**Note**
The debug output is extremely useful when diagnosing issues in the AWS SDK for PHP. Please provide the debug output for an isolated failure case when opening issues on the SDK.

## stats

**Type**

bool|array

Binds transfer statistics to errors and results returned by SDK operations.

Set to `true` to gather transfer statistics on requests sent.

```php
$s3 = new Aws\S3\S3Client([  'version' => 'latest',  'region' => 'us-west-2',  'stats' => true  ]);  
  // Perform an operation  
  $result = $s3->listBuckets();  
  // Inspect the stats  
  $stats = $result['@metadata']['transferStats'];
```

Alternatively, you can provide an associative array with the following keys.

### retries (bool)

Set to `true` to enable reporting on retries attempted. Retry statistics are collected by default and returned.

### http (bool)

Set to `true` to enable collecting statistics from lower-level HTTP adapters (e.g., values returned in GuzzleHttpTransferStats). HTTP handlers must support an `__on_transfer_stats` option for this to have an effect. HTTP stats are returned as an indexed array of associative arrays; each associative array contains the transfer stats returned for a request by the client’s HTTP handler. Disabled by default.

If a request was retried, each request's transfer stats are returned, with `$result['@metadata']['transferStats']['http'][0]` containing the stats for the first request, `$result['@metadata']['transferStats']['http'][1]` containing the statistics for the second request, and so on.

### timer (bool)

Set to `true` to enable a command timer that reports the total wall clock time spent on an operation in seconds. Disabled by default.
'stats' => [
    'retries' => true,
    'timer' => false,
    'http' => true,
],
});

// Perform an operation
$result = $s3->listBuckets();
// Inspect the HTTP transfer stats
/stats = $result['@metadata']['transferStats']['http'];
// Inspect the number of retries attempted
/stats = $result['@metadata']['transferStats']['retries_attempted'];
// Inspect the total backoff delay inserted between retries
/stats = $result['@metadata']['transferStats']['total_retry_delay'];

endpoint

Type

string

The full URI of the web service. This is required for services, such as MediaConvert, that use account-specific endpoints. For these services, request this endpoint using the describeEndpoints method.

This is only required when connecting to a custom endpoint (e.g., a local version of Amazon S3 or Amazon DynamoDB Local).

Here's an example of connecting to Amazon DynamoDB Local:

```php
$client = new Aws\DynamoDb\DynamoDbClient([  'version' => '2012-08-10',  'region' => 'us-east-1'  'endpoint' => 'http://localhost:8000']);
```

See the [AWS Regions and Endpoints](https://aws.amazon.com/about-aws/global-infrastructure/regions-endpoints/) for a list of available AWS Regions and endpoints.

endpoint_provider

Type

callable

An optional PHP callable that accepts a hash of options, including a “service” and “region” key. It returns NULL or a hash of endpoint data, of which the “endpoint” key is required.

Here's an example of how to create a minimal endpoint provider.

```php
$provider = function (array $params) {  if ($params['service'] == 'foo') {  return ['endpoint' => $params['region'] . '.example.com'];  }  // Return null when the provider cannot handle the parameters  return null;  });
```
endpoint_discovery

Type

array\Aws\CacheInterface|\Aws\EndpointDiscovery\ConfigurationInterface|callable

Endpoint discovery identifies and connects to the correct endpoint for a service API that supports endpoint discovery. For services that support but don't require endpoint discovery, enable endpoint_discovery during client creation. If a service does not support endpoint discovery this configuration is ignored.

\Aws\EndpointDiscovery\ConfigurationInterface

An optional configuration provider that enables automatic connection to the appropriate endpoint of a service API for operations the service specifies.

The \Aws\EndpointDiscovery\Configuration object accepts two options, including a Boolean value, "enabled", that indicates if endpoint discovery is enabled, and an integer "cache_limit" that indicates the maximum number of keys in the endpoint cache.

For each client created, pass an \Aws\EndpointDiscovery\Configuration object to use a specific configuration for endpoint discovery.

```php
use \Aws\EndpointDiscovery\Configuration;
use \Aws\S3\S3Client;

#enabled = true;
#cache_limit = 1000;
$config = new \Aws\EndpointDiscovery\Configuration (
    $enabled,
    $cache_limit
);

$s3 = new \Aws\S3\S3Client(
    
    'version' => 'latest',
    'region' => 'us-east-2',
    'endpoint_discovery' => $config,

]);
```

Pass an instance of \Aws\CacheInterface to cache the values returned by endpoint discovery across multiple processes.

```php
use \Aws\DoctrineCacheAdapter;
use \Aws\S3\S3Client;
use Doctrine\Common\Cache\ApcuCache;

$s3 = new \S3\S3Client(
    
    'version' => 'latest',
    'region' => 'us-west-2',
    'endpoint_discovery' => new \DoctrineCacheAdapter(new ApcuCache),

];
```

Pass an array to endpoint discovery.

```php
use \Aws\S3\S3Client;
```
$s3 = new S3Client([  
    'version' => 'latest',  
    'region' => 'us-west-2',  
    'endpoint_discovery' => [  
        'enabled' => true,  
        'cache_limit' => 1000  
    ],  
]);

## handler

**Type**

`callable`

A handler that accepts a command object and request object, and that returns a promise (GuzzleHttp\Promises\PromiseInterface) that is fulfilled with an Aws\Result\ResultInterface object or rejected with an Aws\Exception\AwsException. A handler does not accept a next handler as it is terminal and expected to fulfill a command. If no handler is provided, a default Guzzle handler is used.

You can use the Aws\MockHandler to return mocked results or throw mock exceptions. You enqueue results or exceptions, and the MockHandler will dequeue them in FIFO order.

```php
use Aws\Result;
use Aws\MockHandler;
use Aws\DynamoDb\DynamoDbClient;
use Aws\CommandInterface;
use Psr\Http\Message\RequestInterface;
use Aws\Exception\AwsException;

$mock = new MockHandler();

// Return a mocked result
$mock->append(new Result(['foo' => 'bar']));

// You can provide a function to invoke; here we throw a mock exception
$mock->append(function (CommandInterface $cmd, RequestInterface $req) {
    return new AwsException('Mock exception', $cmd);
});

// Create a client with the mock handler
$client = new DynamoDbClient([  
    'region' => 'us-west-2',  
    'version' => 'latest',  
    'handler' => $mock  
]);

// Result object response will contain ['foo' => 'bar']
$result = $client->listTables();

// This will throw the exception that was enqueued
$client->listTables();
```

## http

**Type**

`array`
Set to an array of HTTP options that are applied to HTTP requests and transfers created by the SDK.

The SDK supports the following configuration options:

**cert**

Type

string|array

Specify the PEM formatted client side certificate.

- Set as a string for the path to only the certificate file.

```php
use Aws\S3\S3Client;
$client = new S3Client(["region" => 'us-west-2', "version" => 'latest', 'http' => ['cert' => '/path/to/cert.pem']]));
```

- Set as an array containing the path and password.

```php
use Aws\S3\S3Client;
$client = new S3Client(["region" => 'us-west-2', "version" => 'latest', 'http' => [{
    'cert' => ['/path/to/cert.pem', 'password']
}]]));
```

**connect_timeout**

A float describing the number of seconds to wait while trying to connect to a server. Use 0 to wait indefinitely (the default behavior).

```php
use Aws\DynamoDb\DynamoDbClient;

// Timeout after attempting to connect for 5 seconds
$client = new DynamoDbClient(["region" => 'us-west-2', "version" => 'latest', 'http' => [{
    'connect_timeout' => 5
}]]));
```

**debug**

Type

bool|resource
Instructs the underlying HTTP handler to output debug information. The debug information provided by different HTTP handlers will vary.

- Pass `true` to write debug output to STDOUT.
- Pass a resource as returned by `fopen` to write debug output to a specific PHP stream resource.

```
\public function debug($pass)
```

**decode_content**

Type: `bool`

Instructs the underlying HTTP handler to inflate the body of compressed responses. When not enabled, compressed response bodies might be inflated with a `GuzzleHttp\Psr7\InflateStream`.

**Note**

Content decoding is enabled by default in the SDK's default HTTP handler. For backward compatibility reasons, this default cannot be changed. If you store compressed files in Amazon S3, we recommend that you disable content decoding at the S3 client level.

```
use Aws\S3\S3Client;
use GuzzleHttp\Psr7\InflateStream;

$client = new S3Client([  
    'region' => 'us-west-2',  
    'version' => 'latest',  
    'http' => ['decode_content' => false],  
]);

$result = $client->getObject([  
    'Bucket' => 'my-bucket',  
    'Key' => 'massize_gzipped_file.tgz'  
]);

$compressedBody = $result['Body']; // This content is still gzipped
$inflatedBody = new InflateStream($result['Body']); // This is now readable
```

delay

Type: `int`

The number of milliseconds to delay before sending the request. This is often used for delaying before retrying a request.

```
\public function delay($milliseconds)
```

**expect**

Type: `bool|string`

This option is passed through to the underlying HTTP handler. By default, Expect: 100-Continue header is set when the body of the request exceeds 1 MB. `true` or `false` enables or disables the header on all requests. If an integer is used, only requests with bodies that exceed this setting will use the header. When used as an integer, if the body size is unknown the Expect header will be sent.

```
\public function expect($pass)
```
Warning
Disabling the Expect header can prevent the service from returning authentication or other errors. This option should be configured with caution.

**progress**

Type

callable

Defines a function to invoke when transfer progress is made. The function accepts the following arguments:

1. The total number of bytes expected to be downloaded.
2. The number of bytes downloaded so far.
3. The number of bytes expected to be uploaded.
4. The number of bytes uploaded so far.

```php
use Aws\S3\S3Client;

$client = new S3Client(
    [
        'region' => 'us-west-2',
        'version' => 'latest'
    ]);

// Apply the http option to a specific command using the "@http" command parameter
$result = $client->getObject(
    [
        'Bucket' => 'my-bucket',
        'Key' => 'large.mov',
        '@http' => [
            'progress' => function ($expectedDl, $dl, $expectedUl, $ul) {
                printf("%s of %s downloaded, %s of %s uploaded.\n", $expectedDl, $dl, $expectedUl, $ul);
            }
        ]
    ]);
```

**proxy**

Type

string|array

You can connect to an AWS service through a proxy by using the `proxy` option.

- Provide a string value to connect to a proxy for all types of URIs. The proxy string value can contain a scheme, user name, and password. For example, "http://username:password@192.168.16.1:10".
- Provide an associative array of proxy settings where the key is the scheme of the URI, and the value is the proxy for the given URI (i.e., you can give different proxies for "http" and "https" endpoints).
use Aws\DynamoDb\DynamoDbClient;

// Send requests through a single proxy
$client = new DynamoDbClient([  'region' => 'us-west-2',  'version' => 'latest',  'http' => [    'proxy' => 'http://192.168.16.1:10'  ]]);

// Send requests through a different proxy per scheme

You can use the HTTP_PROXY environment variable to configure an “http” protocol-specific proxy, and the HTTPS_PROXY environment variable to configure an “https” specific proxy.

sink
Type
resource|string|Psr\Http\Message\StreamInterface

The sink option controls where the response data of an operation is downloaded to.

- Provide a resource as returned by fopen to download the response body to a PHP stream.
- Provide the path to a file on disk as a string value to download the response body to a specific file on disk.
- Provide a Psr\Http\Message\StreamInterface to download the response body to a specific PSR stream object.

Note
The SDK downloads the response body to a PHP temp stream by default. This means that the data stays in memory until the size of the body reaches 2 MB, at which point the data is written to a temporary file on disk.

synchronous
Type
bool

The synchronous option informs the underlying HTTP handler that you intend to block the result.
**stream**

Type

`bool`

Set to `true` to tell the underlying HTTP handler that you want to stream the response body of a response from the web service, rather than download it all up front. For example, this option is relied on in the Amazon S3 stream wrapper class to ensure that the data is streamed.

**timeout**

Type

`float`

A float describing the timeout of the request in seconds. Use 0 to wait indefinitely (the default behavior).

```php
use Aws\DynamoDb\DynamoDbClient;

// Timeout after 5 seconds
$client = new DynamoDbClient(
    ['region' => 'us-west-2',
     'version' => 'latest',
     'http'    => [
         'timeout' => 5
     ]
];
```

**verify**

Type

`bool|string`

You can customize the peer SSL/TLS certificate verification behavior of the SDK using the `verify` http option.

- Set to `true` to enable SSL/TLS peer certificate verification and use the default CA bundle provided by the operating system.
- Set to `false` to disable peer certificate verification. (This is not secure!)
- Set to a string to provide the path to a CA cert bundle to enable verification using a custom CA bundle.

If the CA bundle cannot be found for your system and you receive an error, provide the path to a CA bundle to the SDK. If you do not need a specific CA bundle, Mozilla provides a commonly used CA bundle which you can download [here](https://curl.haxx.se/docs/caextract.html) (this is maintained by the maintainer of cURL). Once you have a CA bundle available on disk, you can set the `openssl.cafile` PHP .ini setting to point to the path to the file, allowing you to omit the `verify` request option. You can find much more detail on SSL certificates on the cURL website.

```php
use Aws\DynamoDb\DynamoDbClient;

// Use a custom CA bundle
$client = new DynamoDbClient(
    ['region' => 'us-west-2',
     'version' => 'latest',
     'http'    => [
         'openssl.cafile' => '/path/to/ca/bundle.pem'
     ]
];
```
http_handler

Type
callable

The http_handler option is used to integrate the SDK with other HTTP clients. An http_handler option is a function that accepts a Psr\Http\Message\RequestInterface object and an array of http options applied to the command, and returns a GuzzleHttp\Promise\PromiseInterface object that is fulfilled with a Psr\Http\Message\ResponseInterface object or rejected with an array of the following exception data:

- exception - (\Exception) the exception that was encountered.
- response - (Psr\Http\Message\ResponseInterface) the response that was received (if any).
- connection_error - (bool) set to true to mark the error as a connection error. Setting this value to true also allows the SDK to automatically retry the operation, if needed.

The SDK automatically converts the given http_handler into a normal handler option by wrapping the provided http_handler with a Aws\WrappedHttpHandler object.

By default, the SDK uses Guzzle as its HTTP handler. You can supply a different HTTP handler here, or provide a Guzzle client with your own custom defined options.

Setting TLS version

One use case is to set the TLS version used by Guzzle with Curl, assuming Curl is installed in your environment. Note the Curl version constraints for what version of TLS is supported. By default, the latest version is used. If the TLS version is explicitly set, and the remote server doesn't support this version, it will produce an error instead of using an earlier TLS version.

You can determine the TLS version being used for a given client operation by setting the debug client option to true and examining the SSL connection output. That line might look something like:

SSL connection using TLSv1.2

Example setting TLS 1.2 with Guzzle 6:

```php
use Aws\DynamoDb\DynamoDbClient;
use Aws\Handler\GuzzleV6\GuzzleHandler;
use GuzzleHttp\Client;

//handler = new GuzzleHandler(
   new Client(
      'curl' => [
         CURLOPT_SSLVERSION => CURL_SSLVERSION_TLSv1_2
      ]
   ));
```
$client = new DynamoDbClient([  
    'region' => 'us-west-2',  
    'version' => 'latest',  
    'http_handler' => $handler  
]);

Note
The http_handler option supersedes any provided handler option.

profile
Type
string

Enables you to specify which profile to use when credentials are created from the AWS credentials file in your HOME directory. This setting overrides the AWS_PROFILE environment variable.

Note
Specifying "profile" will cause the "credentials" key to be ignored.

// Use the "production" profile from your credentials file
$ec2 = new Aws\Ec2\Ec2Client([  
    'version' => '2014-10-01',  
    'region' => 'us-west-2',  
    'profile' => 'production'  
]);

See Credentials for the AWS SDK for PHP Version 3 (p. 42) for more information about configuring credentials and the .ini file format.

region
Type
string

AWS Region to connect to. See the AWS Regions and Endpoints for a list of available Regions.

// Set the Region to the EU (Frankfurt) Region
$s3 = new Aws\S3\S3Client([  
    'region' => 'eu-central-1',  
    'version' => '2006-03-01'  
]);

retries
Type
int|array|Aws\CacheInterface|Aws\Retry\ConfigurationInterface|callable
Default

```
int(3)
```

Configures the retry mode and maximum number of allowed retries for a client. Pass 0 to disable retries.

The three retry modes are:

- **legacy** - the default legacy retry implementation
- **standard** - adds a retry quota system to prevent retries that are unlikely to succeed
- **adaptive** - builds on the standard mode, adding a client-side rate limiter. Note this mode is considered experimental.

The configuration for retries consists of the mode and the max attempts to be used for each request. The configuration can be set in a couple of different locations, in the following order of precedence.

**Order of Precedence**

The order of precedence for retry configuration is as follows (1 overrides 2-3, etc.):

1. Client configuration option
2. Environment variables
3. AWS Shared config file

**Environment variables**

- `AWS_RETRY_MODE` - set to legacy, standard, or adaptive.
- `AWS_MAX_ATTEMPTS` - set to an integer value for the max attempts per request

**Shared config file keys**

- `retry_mode` - set to legacy, standard, or adaptive.
- `max_attempts` - set to an integer value for the max attempts per request

**Client configuration**

The following example disables retries for the Amazon DynamoDB client.

```php
// Disable retries by setting "retries" to 0
$client = new Aws\DynamoDb\DynamoDbClient([  'version' => '2012-08-10',  'region' => 'us-west-2',  'retries' => 0  ]);%
```

The following example passes in an integer, which will default to legacy mode with the passed in number of retries

```php
// Disable retries by setting "retries" to 0
$client = new Aws\DynamoDb\DynamoDbClient([  'version' => '2012-08-10',  'region' => 'us-west-2',  'retries' => 3  ]);%
```
The `Aws\Retry\Configuration` object accepts two parameters, the retry mode and an integer for the maximum attempts per request. This example passes in an `Aws\Retry\Configuration` object for retry configuration.

```php
euse Aws\EndpointDiscovery\Configuration;
euse Aws\S3\S3Client;
$enabled = true;
$cache_limit = 1000;
$config = new Aws\Retry\Configuration('adaptive', 10);
$s3 = new Aws\S3\S3Client(
    ['version' => 'latest',
     'region' => 'us-east-2',
     'retries' => $config,
    ]);}
```

This example passes in an array for retry configuration.

```php
euse Aws\S3\S3Client;
$s3 = new S3Client(
    ['version' => 'latest',
     'region' => 'us-west-2',
     'retries' => [
         'mode' => 'standard',
         'max_attempts' => 7
     ],
    ]);}
```

This examples passes an instance of `Aws\CacheInterface` to cache the values returned by the default retry configuration provider.

```php
euse Aws\DoctrineCacheAdapter;
euse Aws\S3\S3Client;
euse Doctrine\Common\Cache\ApcuCache;
$s3 = new S3Client(
    ['version' => 'latest',
     'region' => 'us-west-2',
     'endpoint_discovery' => new DoctrineCacheAdapter(new ApcuCache),
    ]);}
```

**scheme**

Type  
string  

Default  
string(5) "https"
URI scheme to use when connecting. The SDK uses "https" endpoints (i.e., uses SSL/TLS connections) by default. You can attempt to connect to a service over an unencrypted "http" endpoint by setting `scheme` to "http".

```php
$s3 = new Aws\S3\S3Client([  'version' => '2006-03-01',  'region' => 'us-west-2',  'scheme' => 'http']);
```

See the [AWS Regions and Endpoints](https://aws.amazon.com/about-aws/global-infrastructure/regions-endpoints) for a list of endpoints and whether a service supports the `http` scheme.

### service

**Type**

string

**Required**

true

Name of the service to use. This value is supplied by default when using a client provided by the SDK (i.e., `Aws\S3\S3Client`). This option is useful when testing a service that has not yet been published in the SDK but that you have available on disk.

### signature_provider

**Type**

callable

A callable that accepts a signature version name (e.g., `v4`), a service name, and AWS Region and returns a `Aws\Signature\SignatureInterface` object or NULL if the provider is able to create a signer for the given parameters. This provider is used to create signers used by the client.

There are various functions provided by the SDK in the `Aws\Signature\SignatureProvider` class that can be used to create customized signature providers.

### signature_version

**Type**

string

A string representing a custom signature version to use with a service (e.g., `v4`, etc.). Per operation signature version MAY override this requested signature version, if needed.

The following examples show how to configure an Amazon S3 client to use `signature version 4`:

```php
// Set a preferred signature version
$s3 = new Aws\S3\S3Client([  'version' => '2006-03-01',  'region' => 'us-west-2',  'scheme' => 'http']);
```
'signature_version' => 'v4'
]);

**Note**
The signature_provider used by your client MUST be able to create the signature_version option you provide. The default signature_provider used by the SDK can create signature objects for "v4" and "anonymous" signature versions.

**ua_append**

Type

string|string[]

Default

[]

A string or array of strings that are added to the user-agent string passed to the HTTP handler.

**use_aws_shared_config_files**

Type

bool|array

Default

bool(true)

Set to false to disable checking for shared config file in ‘~/.aws/config’ and ‘~/.aws/credentials’. This will override the AWS_CONFIG_FILE environment variable.

**validate**

Type

bool|array

Default

bool(true)

Set to false to disable client-side parameter validation. You might find that turning validation off will slightly improve client performance, but the difference is negligible.

```php
// Disable client-side validation
$s3 = new Aws\S3\S3Client([ 'version' => '2006-03-01', 'region' => 'eu-west-1', 'validate' => false
]);
```

Set to an associative array of validation options to enable specific validation constraints:

- **required** - Validate that required parameters are present (on by default).
version

Type

string

Required

true

The version of the web service to use (e.g., 2006-03-01).

A “version” configuration value is required. Specifying a version constraint ensures that your code will not be affected by a breaking change made to the service. For example, when using Amazon S3, you can lock your API version to 2006-03-01.

$s3 = new Aws\S3\S3Client([  
    'version' => '2006-03-01',  
    'region'  => 'us-east-1'  
]);

A list of available API versions can be found on each client’s API documentation page. If you are unable to load a specific API version, you might need to update your copy of the SDK.

You can provide the string latest to the “version” configuration value to use the most recent available API version that your client’s API provider can find (the default api_provider scans the src/data directory of the SDK for API models).

// Use the latest version available
$s3 = new Aws\S3\S3Client([  
    'version' => 'latest',  
    'region'  => 'us-east-1'  
]);

Warning

We do not recommend Using latest in a production application because pulling in a new minor version of the SDK that includes an API update could break your production application.

Credentials for the AWS SDK for PHP Version 3

To make requests to Amazon Web Services, supply AWS access keys, also known as credentials, to the AWS SDK for PHP.
You can do this in the following ways:

- Use the default credential provider chain (*recommended*).
- Use a specific credential provider or provider chain (or create your own).
- Supply the credentials yourself. These can be root account credentials, IAM credentials, or temporary credentials retrieved from AWS STS.

**Important**

For security, we strongly recommend that you use IAM users instead of the root account for AWS access. For more information, see IAM Best Practices in the IAM User Guide.

### Using the Default Credential Provider Chain

When you initialize a new service client without providing any credential arguments, the SDK uses the default credential provider chain to find AWS credentials. The SDK uses the first provider in the chain that returns credentials without an error.

The default provider chain looks for and uses credentials as follows, in this order:

1. **Use credentials from environment variables** (p. 44).
   
   Setting environment variables is useful if you’re doing development work on a machine other than an Amazon EC2 instance.

2. **Use the AWS shared credentials file and profiles** (p. 44).
   
   This credentials file is the same one used by other SDKs and the AWS CLI. If you’re already using a shared credentials file, you can use that file for this purpose.

   We use this method in most of our PHP code examples.

3. **Assume an IAM role** (p. 46).

   IAM roles provide applications on the instance with temporary security credentials to make AWS calls. For example, IAM roles offer an easy way to distribute and manage credentials on multiple Amazon EC2 instances.

### Other Ways to Add Credentials

You can also add credentials in these ways:

- **Using a credential provider** (p. 50).
  
  Provide custom logic for credentials when constructing the client.

- **Using temporary credentials from AWS STS** (p. 56).

  When using a multi-factor authentication (MFA) token for two-factor authentication, use AWS STS to give the user temporary credentials to access AWS services or use the AWS SDK for PHP.

- **Using hard-coded credentials** (p. 58) (not recommended).

  **Warning**

  Hard-coding your credentials can be dangerous, because it’s easy to accidentally commit your credentials into an SCM repository. This can potentially expose your credentials to more people than you intend. It can also make it difficult to rotate credentials in the future. Do not submit code with hard-coded credentials to your source control.
Using Credentials from Environment Variables

Using environment variables to contain your credentials prevents you from accidentally sharing your AWS secret access key. We recommend that you never add your AWS access keys directly to the client in any production files. Many developers have had their account compromised by leaked keys.

To authenticate to Amazon Web Services, the SDK first checks for credentials in your environment variables. The SDK uses the `getenv()` function to look for the AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN environment variables. These credentials are referred to as environment credentials.

If you’re hosting your application on AWS Elastic Beanstalk, you can set the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables through the AWS Elastic Beanstalk console so that the SDK can use those credentials automatically.

You can also set the environment variables in the command line, as shown here.

**Linux**

```bash
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
# The access key for your AWS account.
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
# The secret access key for your AWS account.
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
# The session key for your AWS account. This is needed only when you are using temporary credentials.
# The AWS_SECURITY_TOKEN environment variable can also be used, but is only supported for backward compatibility purposes.
# AWS_SESSION_TOKEN is supported by multiple AWS SDKs other than PHP.
```

**Windows**

```bash
C:\> SET AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
# The access key for your AWS account.
C:\> SET AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
# The secret access key for your AWS account.
C:\> SET AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
# The session key for your AWS account. This is needed only when you are using temporary credentials.
# The AWS_SECURITY_TOKEN environment variable can also be used, but is only supported for backward compatibility purposes.
# AWS_SESSION_TOKEN is supported by multiple AWS SDKs besides PHP.
```

Using the AWS Credentials File and Credential Profiles

A credentials file is a plaintext file that contains your access keys. The file must:

- Be on the same machine on which you’re running your application.
- Be named credentials.
- Be located in the .aws/ folder in your home directory.
The home directory can vary by operating system. On Windows, you can refer to your home directory by using the environment variable %UserProfile%. On Unix-like systems, you can use the environment variable $HOME or ~ (tilde).

If you already use this file for other SDKs and tools (like the AWS CLI), you don’t need to change anything to use the files in this SDK. If you use different credentials for different tools or applications, you can use profiles to configure multiple access keys in the same configuration file.

We use this method in all our PHP code examples.

Using an AWS credentials file offers the following benefits:

• Your projects’ credentials are stored outside of your projects, so there is no chance of accidentally committing them into version control.
• You can define and name multiple sets of credentials in one place.
• You can easily reuse the same credentials among projects.
• Other AWS SDKs and tools support, this same credentials file. This allows you to reuse your credentials with other tools.

The format of the AWS credentials file should look something like the following.

```
[default]
aws_access_key_id = YOUR_AWS_ACCESS_KEY_ID
aws_secret_access_key = YOUR_AWS_SECRET_ACCESS_KEY

[project1]
aws_access_key_id = ANOTHER_AWS_ACCESS_KEY_ID
aws_secret_access_key = ANOTHER_AWS_SECRET_ACCESS_KEY
```

Each section (e.g., [default], [project1]), represents a separate credential profile. You can reference profiles from an SDK configuration file, or when you are instantiating a client, by using the profile option.

```
use Aws\DynamoDb\DynamoDbClient;

// Instantiate a client with the credentials from the project1 profile
$client = new DynamoDbClient(
    [        'profile' => 'project1',        'region' => 'us-west-2',        'version' => 'latest',    ]);`
```

If no credentials or profiles were explicitly provided to the SDK and no credentials were defined in environment variables, but a credentials file is defined, the SDK uses the "default" profile. You can change the default profile by specifying an alternate profile name in the AWS_PROFILE environment variable.

### Assume Role with Profile

You can configure the AWS SDK for PHP to use an IAM role by defining a profile for the role in ~/.aws/credentials.

Create a new profile with the role_arn for the role you will assume. Also include the source_profile of a profile with credentials that have permissions to assume the IAM role.

Profile in ~/.aws/credentials:
Assume an IAM Role

By setting the AWS_PROFILE environment variable, or profile option when instantiating a client, the role specified in `project1` will be assumed, using the default profile as the source credentials.

Roles can also be assumed for profiles defined in `~/.aws/config`. Setting the environment variable `AWS_SDK_LOAD_NONDEFAULT_CONFIG` enables loading profiles for assuming a role from `~/.aws/config`. When enabled, profiles from both `~/.aws/config` and `~/.aws/credentials` will be loaded. Profiles from `~/.aws/credentials` are loaded last and will take precedence over a profile from `~/.aws/config` with the same name. Profiles from either location can serve as the source_profile or the profile to be assumed.

Profile in `~/.aws/config`:

```
[profile project1]
role_arn = arn:aws:iam::123456789012:role/testing
source_profile = default
role_session_name = OPTIONAL_SESSION_NAME
```

Profile in `~/.aws/credentials`:

```
[project2]
aws_access_key_id = YOUR_AWS_ACCESS_KEY_ID
aws_secret_access_key = YOUR_AWS_SECRET_ACCESS_KEY
```

Using the above files, `project1` will be assumed using `project2` as the source credentials.

Assuming IAM Roles

Using IAM Roles for Amazon EC2 Instance Variable Credentials

If you're running your application on an Amazon EC2 instance, the preferred way to provide credentials to make calls to AWS is to use an IAM role to get temporary security credentials.

When you use IAM roles, you don’t need to worry about credential management from your application. They allow an instance to "assume" a role by retrieving temporary credentials from the Amazon EC2 instance's metadata server.

The temporary credentials, often referred to as instance profile credentials, allow access to the actions and resources that the role's policy allows. Amazon EC2 handles all the legwork of securely authenticating instances to the IAM service to assume the role, and periodically refreshing the retrieved role credentials. This keeps your application secure with almost no work on your part.

**Note**

Instance profile credentials and other temporary credentials generated by the AWS Security Token Service (AWS STS) are not supported by every service. To determine whether the service you're using supports temporary credentials, see **AWS Services that Support AWS STS**.

To avoid hitting the metadata service every time, you can pass an instance of `Aws\Cache\interface` in as the 'credentials' option to a client constructor. This lets the SDK
Create and Assign an IAM Role to an Amazon EC2 Instance

1. Create an IAM client.

   **Imports**
   ```php
   require 'vendor/autoload.php';
   use Aws\Iam\IamClient;
   ``

   **Sample Code**
   ```php
   $client = new IamClient(['region' => 'us-west-2', 'version' => '2010-05-08']);
   ```

2. Create an IAM role with the permissions for the actions and resources you'll use.

   **Sample Code**
   ```php
   $result = $client->createRole(['AssumeRolePolicyDocument' => 'IAM JSON Policy', 'Description' => 'Description of Role', 'RoleName' => 'RoleName']);
   ```

3. Create an IAM instance profile and store the Amazon Resource Name (ARN) from the result.

   **Note**
   If you use the IAM console instead of the AWS SDK for PHP, the console creates an instance profile automatically and gives it the same name as the role to which it corresponds.

   **Sample Code**
   ```php
   $IPN = 'InstanceProfileName';
   $result = $client->createInstanceProfile(['InstanceProfileName' => $IPN]);
   $ARN = $result['Arn'];
   $InstanceID = $result['InstanceProfileId'];
   ```

4. Create an Amazon EC2 client.

   **Imports**
   ```php
   require 'vendor/autoload.php';
   use Aws\Ec2\Ec2Client;
   ``

   **Sample Code**
   ```php
   $ec2Client = new Ec2Client(['region' => 'us-west-2', 'version' => '2016-11-15'],
   ```
5. Add the instance profile to a running or stopped Amazon EC2 instance. Use the instance profile name of your IAM role.

**Sample Code**

```php
$result = $ec2Client->associateIamInstanceProfile([ 'IamInstanceProfile' => [ 'Arn' => $ARN, 'Name' => $IPN, ], 'InstanceId' => $InstanceID ]);```

For more information, see [IAM Roles for Amazon EC2](https://docs.aws.amazon.com/Iam/latest/UserGuide/id_roles_for_amazon_resources.html).

### Using IAM Roles for Amazon ECS Tasks

By using IAM roles for Amazon Elastic Container Service (Amazon ECS) tasks, you can specify an IAM role that the containers in a task can use. This is a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with an ECS task definition or `RunTask` API operation.

**Note**

Instance profile credentials and other temporary credentials generated by AWS STS are not supported by every service. To determine whether the service you're using supports temporary credentials, see [AWS Services that Support AWS STS](https://docs.aws.amazon.com/Iam/latest/UserGuide/id_usersスタッフ.html).

For more information, see [IAM Roles for Amazon EC2 Container Service Tasks](https://docs.aws.amazon.com/Iam/latest/UserGuide/id_roles_for_amazon_resources.html).

### Assuming an IAM Role in Another AWS Account

When you work in an AWS account (Account A) and want to assume a role in another account (Account B), you must first create an IAM role in Account B. This role allows entities in your account (Account A) to perform specific actions in Account B. For more information about cross-account access, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](https://docs.aws.amazon.com/Iam/latest/UserGuide/id_roles_for_amazon_resources.html).

After you create a role in Account B, record the Role ARN. You will use this ARN when you assume the role from Account A. You assume the role using the AWS credentials associated with your entity in Account A.

Create an AWS STS client with credentials for your AWS account. In the following, we used a credentials profile, but you can use any method. With the newly created AWS STS client, call assume-role and provide a custom sessionName. Retrieve the new temporary credentials from the result. By default credentials last an hour.

**Sample Code**

```php
#stsClient = new Aws\Sts\StsClient([ 'profile' => 'default', 'region' => 'us-east-2', 'version' => '2011-06-15' ]); #ARN = "arn:aws:iam::123456789012:role/xaccounts3access";```
# Assume an IAM Role

```php
$sessionName = "s3-access-example";
$result = $stsClient->AssumeRole([  'RoleArn'  => $ARN,  'RoleSessionName' => $sessionName,
]);

$s3Client = new S3Client([  'version'  => '2006-03-01',  'region' => 'us-west-2',  'credentials' => [    'key'  => $result['Credentials']['AccessKeyId'],    'secret' => $result['Credentials']['SecretAccessKey'],    'token' => $result['Credentials']['SessionToken']  ]
]);
```

For more information, see Using IAM Roles or AssumeRole in the AWS SDK for PHP API Reference.

## Using an IAM Role with Web Identity

Web Identity Federation enables customers to use third-party identity providers for authentication when accessing AWS resources. Before you can assume a role with web identity, you must create an IAM role and configure a web identity provider (IdP). For more information, see Creating a Role for Web Identity or OpenID Connect Connect Federation (Console).

After creating an identity provider and creating a role for your web identity, use an AWS STS client to authenticate a user. Provide the webIdentityToken and ProviderId for your identity, and the Role ARN for the IAM role with permissions for the user.

### Sample Code

```php
$stsClient = new Aws\Sts\StsClient([  'profile' => 'default',  'region' => 'us-east-2',  'version' => '2011-06-15'
]);

$ARN = "arn:aws:iam::123456789012:role/xaccounts3access";
$s3SessionName = "s3-access-example";
$duration = 3600;

$result = $stsClient->AssumeRoleWithWebIdentity([  'WebIdentityToken' => "FACEBOOK_ACCESS_TOKEN",  'ProviderId' => "graph.facebook.com",  'RoleArn' => $ARN,  'RoleSessionName' => $sessionName,
]);

$s3Client = new S3Client([  'version' => '2006-03-01',  'region' => 'us-west-2',  'credentials' => [    'key'  => $result['Credentials']['AccessKeyId'],    'secret' => $result['Credentials']['SecretAccessKey'],    'token' => $result['Credentials']['SessionToken']  ]
]);
```

For more information, see AssumeRoleWithWebIdentity—Federation Through a Web-based Identity Provider or AssumeRoleWithWebIdentity in the AWS SDK for PHP API Reference.
Using a Credential Provider

A credential provider is a function that returns a `GuzzleHttp\Promise\PromiseInterface` that is fulfilled with an `Aws\Credentials\CredentialsInterface` instance or rejected with an `Aws\Exception\CredentialsException`. You can use credential providers to implement your own custom logic for creating credentials or to optimize credential loading.

Credential providers are passed into the `credentials` client constructor option. Credential providers are asynchronous, which forces them to be lazily evaluated each time an API operation is invoked. As such, passing in a credential provider function to an SDK client constructor doesn't immediately validate the credentials. If the credential provider doesn't return a credentials object, an API operation will be rejected with an `Aws\Exception\CredentialsException`.

```php
use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;

// Use the default credential provider
$provider = CredentialProvider::defaultProvider();

// Pass the provider to the client
$client = new S3Client([  
    'region' => 'us-west-2',  
    'version' => '2006-03-01',  
    'credentials' => $provider
]);
```

Built-In Providers in the SDK

The SDK provides several built-in providers that you can combine with any custom providers.

**Important**

Credential providers are invoked every time an API operation is performed. If loading credentials is an expensive task (e.g., loading from disk or a network resource), or if credentials are not cached by your provider, consider wrapping your credential provider in an `Aws\Credentials\CredentialProvider::memoize` function. The default credential provider used by the SDK is automatically memoized.

**assumeRole provider**

If you use `Aws\Credentials\AssumeRoleCredentialProvider` to create credentials by assuming a role, you need to provide `'client'` information with an `StsClient` object and `'assume_role_params'` details, as shown.

**Note**

To avoid unnecessarily fetching AWS STS credentials on every API operation, you can use the `memoize` function to handle automatically refreshing the credentials when they expire. See the following code for an example.

```php
use Aws\Credentials\CredentialProvider;
use Aws\Credentials\InstanceProfileProvider;
use Aws\Credentials\AssumeRoleCredentialProvider;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;

// Passing Aws\Credentials\AssumeRoleCredentialProvider options directly
$profile = new InstanceProfileProvider();
$ARN = "arn:aws:iam::123456789012:role/xaccounts3access";
$sessionName = "s3-access-example";
```
Using a Credential Provider

```php
$assumeRoleCredentials = new AssumeRoleCredentialProvider([  
    'client' => new StsClient([  
        'region' => 'us-east-2',  
        'version' => '2011-06-15',  
        'credentials' => $profile  
    ]),  
    'assume_role_params' => [  
        'RoleArn' => $ARN,  
        'RoleSessionName' => $sessionName,  
    ],  
]);

// To avoid unnecessarily fetching STS credentials on every API operation,  
// the memoize function handles automatically refreshing the credentials when they expire  
$provider = CredentialProvider::memoize($assumeRoleCredentials);

$client = new S3Client([  
    'region'      => 'us-east-2',  
    'version'     => '2006-03-01',  
    'credentials' => $provider  
]);
```

For more information regarding 'assume_role_params', see **AssumeRole**.

## Chaining Providers

You can chain credential providers by using the `Aws\Credentials\CredentialProvider::chain()` function. This function accepts a variadic number of arguments, each of which are credential provider functions. This function then returns a new function that's the composition of the provided functions, such that they are invoked one after the other until one of the providers returns a promise that is fulfilled successfully.

The `defaultProvider` uses this composition to check multiple providers before failing. The source of the `defaultProvider` demonstrates the use of the `chain` function.

```php
// This function returns a provider  
public static function defaultProvider(array $config = [])  
{  
    // This function is the provider, which is actually the composition  
    // of multiple providers. Notice that we are also memoizing the result by  
    // default.  
    return self::memoize(  
        self::chain(  
            self::env(),  
            self::ini(),  
            self::instanceProfile($config)  
        )  
    );  
}
```

## Creating a Custom Provider

Credential providers are simply functions that when invoked return a promise (`GuzzleHttp\Promise\PromiseInterface`) that is fulfilled with an `Aws\Credentials\CredentialsInterface` object or rejected with an `Aws\Exception\CredentialsException`.

A best practice for creating providers is to create a function that is invoked to create the actual credential provider. As an example, here's the source of the `env` provider (slightly modified for example purposes). Notice that it is a function that returns the actual provider function. This allows you to easily compose credential providers and pass them around as values.
use GuzzleHttp\Promise;
use GuzzleHttp\Promise\RejectedPromise;

// This function CREATES a credential provider
public static function env()
{
    // This function IS the credential provider
    return function () {
        // Use credentials from environment variables, if available
        $key = getenv(self::ENV_KEY);
        $secret = getenv(self::ENV_SECRET);
        if ($key && $secret) {
            return Promise\promise_for(
                new Credentials($key, $secret, getenv(self::ENV_SESSION))
            );
        }
        $msg = 'Could not find environment variable ' . 'credentials in ' . self::ENV_KEY . '/' . self::ENV_SECRET;
        return new RejectedPromise(new CredentialsException($msg));
    }
}

defaultProvider provider

Aws\Credentials\CredentialProvider::defaultProvider is the default credential provider. This provider is used if you omit a credentials option when creating a client. It first attempts to load credentials from environment variables, then from an .ini file (an .aws/credentials file first, followed by an .aws/config file), and then from an instance profile (EcsCredentials first, followed by Ec2 metadata).

Note
The result of the default provider is automatically memoized.

eCsCredentials provider

Aws\Credentials\CredentialProvider::ecsCredentials attempts to load credentials by a GET request, whose URI is specified by the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI in the container.

use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;
$provider = CredentialProvider::ecsCredentials();
// Be sure to memoize the credentials
$memoizedProvider = CredentialProvider::memoize($provider);
$client = new S3Client([  'region' => 'us-west-2',  'version' => '2006-03-01',  'credentials' => $memoizedProvider]);

env provider

Aws\Credentials\CredentialProvider::env attempts to load credentials from environment variables.

use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;
Using a Credential Provider

$client = new S3Client(
    'region'      => 'us-west-2',
    'version'     => '2006-03-01',
    'credentials' => CredentialProvider::env()
);

assume role with web identity provider

Aws\Credentials\CredentialProvider::assumeRoleWithWebIdentityCredentialProvider attempts to load credentials by assuming a role. If the environment variables AWS_ROLE_ARN and AWS_WEB_IDENTITY_TOKEN_FILE are present, the provider will attempt to assume the role specified at AWS_ROLE_ARN using the token on disk at the full path specified in AWS_WEB_IDENTITY_TOKEN_FILE. If environment variables are used, the provider will attempt to set the session from the AWS_ROLE_SESSION_NAME environment variable.

If environment variables are not set, the provider will use the default profile, or the one set as AWS_PROFILE. The provider reads profiles from ~/.aws/credentials and ~/.aws/config by default, and can read from profiles specified in the filename config option. The provider will assume the role in role_arn of the profile, reading a token from the full path set in web_identity_token_file. role_session_name will be used if set on the profile.

The provider is called as part of the default chain and can be called directly.

use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;

$provider = CredentialProvider::assumeRoleWithWebIdentityCredentialProvider();
// Cache the results in a memoize function to avoid loading and parsing
// the ini file on every API operation
$provider = CredentialProvider::memoize($provider);

$client = new S3Client([  
    'region' => 'us-west-2',  
    'version' => '2006-03-01',  
    'credentials' => $provider
]);

By default, this credential provider will inherit the configured region which will be used by the StsClient to assume the role. Optionally, a full StsClient can be provided. Credentials should be set as false on any provided StsClient.

use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;

$stsClient = new StsClient([  
    'region' => 'us-west-2',  
    'version' => 'latest',  
    'credentials' => false
]);

$provider = CredentialProvider::assumeRoleWithWebIdentityCredentialProvider([  
    'stsClient' => $stsClient
]);
// Cache the results in a memoize function to avoid loading and parsing
// the ini file on every API operation
$provider = CredentialProvider::memoize($provider);

$client = new S3Client([  
    'region' => 'us-west-2',
]);
ini provider

`Aws\Credentials\CredentialProvider::ini` attempts to load credentials from an ini credential file (p. 44). By default, the SDK attempts to load the “default” profile from a file located at `~/.aws/credentials`.

```php
use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;

$provider = CredentialProvider::ini();
// Cache the results in a memoize function to avoid loading and parsing
// the ini file on every API operation
$provider = CredentialProvider::memoize($provider);

$client = new S3Client([  
    'region' => 'us-west-2',  
    'version' => '2006-03-01',  
    'credentials' => $provider  
]);
```

You can use a custom profile or .ini file location by providing arguments to the function that creates the provider.

```php
$profile = 'production';
$path = '/full/path/to/credentials.ini';

$provider = CredentialProvider::ini($profile, $path);
$provider = CredentialProvider::memoize($provider);

$client = new S3Client([  
    'region' => 'us-west-2',  
    'version' => '2006-03-01',  
    'credentials' => $provider  
]);
```

process provider

`Aws\Credentials\CredentialProvider::process` attempts to load credentials from a credential_process specified in an ini credential file (p. 44). By default, the SDK attempts to load the “default” profile from a file located at `~/.aws/credentials`. The SDK will call the credential_process command exactly as given and then read JSON data from stdout. The credential_process must write credentials to stdout in the following format:

```json
{
    "Version": 1,
    "AccessKeyId": "",
    "SecretAccessKey": "",
    "SessionToken": "",
    "Expiration": ""
}
```

SessionToken and Expiration are optional. If present, the credentials will be treated as temporary.
use Aws\S3\S3Client;

$provider = CredentialProvider::process();
// Cache the results in a memoize function to avoid loading and parsing
// the ini file on every API operation
$provider = CredentialProvider::memoize($provider);

$client = new S3Client(
    ['region' => 'us-west-2',
     'version' => '2006-03-01',
     'credentials' => $provider
    ]);

You can use a custom profile or .ini file location by providing arguments to the function that creates the provider.

$profile = 'production';
$path = '/full/path/to/credentials.ini';

$provider = CredentialProvider::process($profile, $path);
$provider = CredentialProvider::memoize($provider);

$client = new S3Client(
    ['region' => 'us-west-2',
     'version' => '2006-03-01',
     'credentials' => $provider
    ]);

**instanceProfile provider**

Aws\Credentials\CredentialProvider::instanceProfile attempts to load credentials from Amazon EC2 instance profiles.

use Aws\Credentials\CredentialProvider;
use Aws\S3\S3Client;

$provider = CredentialProvider::instanceProfile();
// Be sure to memoize the credentials
$memoizedProvider = CredentialProvider::memoize($provider);

$client = new S3Client(
    ['region' => 'us-west-2',
     'version' => '2006-03-01',
     'credentials' => $memoizedProvider
    ]);

By default, the provider retries fetching credentials up to three times. The number of retries can be set with the retries option, and disabled entirely by setting the option to 0.

use Aws\Credentials\CredentialProvider;

$provider = CredentialProvider::instanceProfile([‘retries’ => 0]);
$memoizedProvider = CredentialProvider::memoize($provider);

**Note**

You can disable this attempt to load from Amazon EC2 instance profiles by setting the AWS_EC2_METADATA_DISABLED environment variable to true.
Memoizing Credentials

At times you might need to create a credential provider that remembers the previous return value. This can be useful for performance when loading credentials is an expensive operation or when using the \Aws\Sdk class to share a credential provider across multiple clients. You can add memoization to a credential provider by wrapping the credential provider function in a \memoize function.

```php
use Aws\Credentials\CredentialProvider;

$provider = CredentialProvider::instanceProfile();
// Wrap the actual provider in a memoize function
$provider = CredentialProvider::memoize($provider);

// Pass the provider into the Sdk class and share the provider
// it will use the previously returned credentials as long as
// they haven't yet expired.
$sdk = new Aws\Sdk(['credentials' => $provider]);

$s3 = $sdk->getS3(['region' => 'us-west-2', 'version' => 'latest']);
$ec2 = $sdk->getEc2(['region' => 'us-west-2', 'version' => 'latest']);
assert($s3->getCredentials() === $ec2->getCredentials());
```

When the memoized credentials are expired, the memoize wrapper invokes the wrapped provider in an attempt to refresh the credentials.

Using Temporary Credentials from AWS STS

AWS Security Token Service (AWS STS) enables you to request limited privilege, temporary credentials for IAM users, or for users that you authenticate via identity federation.

One common use case for temporary credentials is to grant mobile or client-side applications access to AWS resources by authenticating users through third-party identity providers (see Web Identity Federation).

**Note**

Temporary credentials generated by AWS STS are not supported by every service. To determine whether the service you are using supports temporary credentials, see IAM Temporary Security Credentials.

Getting Temporary Credentials

AWS STS has several operations that return temporary credentials, but the \GetSessionToken operation is the simplest to demonstrate. Assuming you have an instance of \Aws\Sts\StsClient stored in the $stsClient variable, you call it as follows.

```php
$result = $stsClient->getSessionToken();
```

The result for \GetSessionToken and the other AWS STS operations always contains a 'Credentials' value. If you print the result (e.g., `print_r($result)`), it looks like the following.

```php
Array
(...
  [Credentials] => Array
    (...[SessionToken] => '<base64 encoded session token value>'
    [SecretAccessKey] => '<temporary secret access key value>'
  )
)
Providing Temporary Credentials to the AWS SDK for PHP

You can use temporary credentials with another AWS client by instantiating the client and passing in the values received from AWS STS directly.

```
use Aws\S3\S3Client;
$result = $stsClient->getSessionToken();

$s3Client = new S3Client([  
    'version' => '2006-03-01',  
    'region' => 'us-west-2',  
    'credentials' => [  
        'key' => $result['Credentials']['AccessKeyId'],  
        'secret' => $result['Credentials']['SecretAccessKey'],  
        'token' => $result['Credentials']['SessionToken']  
    ]
]);
```

You can also construct an `Aws\Credentials\Credentials` object and use that when instantiating the client.

```
use Aws\Credentials\Credentials;
use Aws\S3\S3Client;
$result = $stsClient->getSessionToken();

$credentials = new Credentials(  
    $result['Credentials']['AccessKeyId'],  
    $result['Credentials']['SecretAccessKey'],  
    $result['Credentials']['SessionToken']
);

$s3Client = new S3Client([  
    'version' => '2006-03-01',  
    'region' => 'us-west-2',  
    'credentials' => $credentials  
]);
```

However, the best way to provide temporary credentials is to use the `createCredentials()` helper method included with the `StsClient`. This method extracts the data from an AWS STS result and creates the `Credentials` object for you.

```
$result = $stsClient->getSessionToken();
$credentials = $stsClient->createCredentials($result);

$s3Client = new S3Client([  
    'version' => '2006-03-01',  
    'region' => 'us-west-2',  
    'credentials' => $credentials  
]);
```

For more information about why you might need to use temporary credentials in your application or project, see [Scenarios for Granting Temporary Access](#) in the AWS STS documentation.
Using Hard-Coded Credentials

When testing new services or debugging issues, developers often want to include AWS credentials when constructing the client. See the following for an example of how to authenticate to AWS, but do so with caution. Credentials for the AWS SDK for PHP Version 3 (p. 42) lists many recommended ways to add credentials to your project safely.

**Warning**

Hard-coding your credentials can be dangerous because it’s easy to commit your credentials into an SCM repository accidentally. Adding credentials directly in your production code can potentially expose your credentials to more people than you intend. It can also make it difficult to rotate credentials in the future.

If you decide to hard-code credentials to an SDK client, provide an associative array of “key”, "secret", and optional “token” key-value pairs to the “credentials” option of a client constructor.

```php
// Hard-coded credentials
$s3Client = new S3Client(
    
    'version' => 'latest',
    'region'  => 'us-west-2',
    'credentials' => [ 
        'key'    => 'my-access-key-id',
        'secret' => 'my-secret-access-key',
    ],
);
```

Creating Anonymous Clients

In some cases, you might want to create a client that is not associated with any credentials. This enables you to make anonymous requests to a service.

For example, you can configure both Amazon S3 objects and Amazon CloudSearch domains to allow anonymous access.

To create an anonymous client, you set the 'credentials' option to false.

```php
$s3Client = new S3Client(
    'version' => 'latest',
    'region'  => 'us-west-2',
    'credentials' => false
);

// Makes an anonymous request. The object would need to be publicly readable for this to succeed.
$result = $s3Client->getObject(
    'Bucket' => 'my-bucket',
    'Key'    => 'my-key',
);
```

Command Objects in the AWS SDK for PHP Version 3

The AWS SDK for PHP uses the command pattern to encapsulate the parameters and handler that will be used to transfer an HTTP request at a later point in time.
Implicit Use of Commands

If you examine any client class, you can see that the methods corresponding to API operations don't actually exist. They are implemented using the \_\_call() magic method. These pseudo-methods are actually shortcuts that encapsulate the SDK's use of command objects.

You don't typically need to interact with command objects directly. When you call methods like \texttt{Aws\S3\S3Client::putObject()}, the SDK actually creates a \texttt{Aws\CommandInterface} object based on the provided parameters, executes the command, and returns a populated \texttt{Aws\ResultInterface} object (or throws an exception on error). A similar flow occurs when calling any of the Async methods of a client (e.g., \texttt{Aws\S3\S3Client::putObjectAsync()}); the client creates a command based on the provided parameters, serializes an HTTP request, initiates the request, and returns a promise.

The following examples are functionally equivalent.

```php
$s3Client = new Aws\S3\S3Client([  'version' => '2006-03-01',  'region' => 'us-standard',]);
$params = [  'Bucket' => 'foo',  'Key'    => 'baz',  'Body'   => 'bar'];

// Using operation methods creates a command implicitly
$result = $s3Client->putObject($params);

// Using commands explicitly
$command = $s3Client->getCommand('PutObject', $params);
$result = $s3Client->execute($command);
```

Command Parameters

All commands support a few special parameters that are not part of a service's API but instead control the SDK's behavior.

@\texttt{http}

Using this parameter, it's possible to fine-tune how the underlying HTTP handler executes the request. The options you can include in the \texttt{@http} parameter are the same as the ones you can set when you instantiate the client with the “http” client option (p. 30).

```php
// Configures the command to be delayed by 500 milliseconds
COMMAND[ '@http' ] = [  'delay' => 500,
];
```

@\texttt{retries}

Like the “\texttt{retries}” client option (p. 37), \texttt{@retries} controls how many times a command can be retried before it is considered to have failed. Set it to 0 to disable retries.

```php
// Disable retries
COMMAND[ '@retries' ] = 0;
```
Creating Command Objects

You can create a command using a client's `getCommand()` method. It doesn't immediately execute or transfer an HTTP request, but is only executed when it is passed to the `execute()` method of the client. This gives you the opportunity to modify the command object before executing the command.

```php
$command = $s3Client->getCommand('ListObjects');
$command['MaxKeys'] = 50;
$command['Prefix'] = 'foo/baz/';
$result = $s3Client->execute($command);

// You can also modify parameters
$command = $s3Client->getCommand('ListObjects', [
    'MaxKeys' => 50,
    'Prefix' => 'foo/baz/',
]);
$command['MaxKeys'] = 100;
$result = $s3Client->execute($command);
```

Command Handler

When a command is created from a client, it is given a clone of the client's `Aws\HandlerList` object. The command is given a clone of the client's handler list to allow a command to use custom middleware and handlers that do not affect other commands that the client executes.

This means that you can use a different HTTP client per command (e.g., `Aws\MockHandler`) and add custom behavior per command through middleware. The following example uses a `MockHandler` to create mock results instead of sending actual HTTP requests.

```php
use Aws\Result;
use Aws\MockHandler;

// Create a mock handler
$mock = new MockHandler();
// Enqueue a mock result to the handler
$mock->append(new Result(['foo' => 'bar']));
// Create a "ListObjects" command
$command = $s3Client->getCommand('ListObjects');
// Associate the mock handler with the command
$command->getHandlerList()->setHandler($mock);
// Executing the command will use the mock handler, which returns the
// mocked result object
$result = $s3Client->execute($command);
echo $result['foo']; // Outputs 'bar'
```

In addition to changing the handler that the command uses, you can also inject custom middleware to the command. The following example uses the `tap` middleware, which functions as an observer in the handler list.

```php
use Aws\CommandInterface;
use Aws\Middleware;
use Psr\Http\Message\RequestInterface;

$command = $s3Client->getCommand('ListObjects');
```
$list = $command->getHandlerList();

// Create a middleware that just dumps the command and request that is
// about to be sent
$middleware = Middleware::tap(
    function (CommandInterface $command, RequestInterface $request) {
        var_dump($command->toArray());
        var_dump($request);
    });

// Append the middleware to the "sign" step of the handler list. The sign
// step is the last step before transferring an HTTP request.
$list->append('sign', $middleware);

// Now transfer the command and see the var_dump data
$s3Client->execute($command);

CommandPool

The `Aws\CommandPool` enables you to execute commands concurrently using an iterator that yields `Aws\CommandInterface` objects. The `CommandPool` ensures that a constant number of commands are executed concurrently while iterating over the commands in the pool (as commands complete, more are executed to ensure a constant pool size).

Here's a very simple example of just sending a few commands using a `CommandPool`.

```php
use Aws\S3\S3Client;
use Aws\CommandPool;

// Create the client
$client = new S3Client([  
    'region'  => 'us-standard',
    'version' => '2006-03-01'
]);

$bucket = 'example';
$commands = [
    $client->getCommand('HeadObject', ['Bucket' => $bucket, 'Key' => 'a']),
    $client->getCommand('HeadObject', ['Bucket' => $bucket, 'Key' => 'b']),
    $client->getCommand('HeadObject', ['Bucket' => $bucket, 'Key' => 'c'])
];

$pool = new CommandPool($client, $commands);

// Initiate the pool transfers
$promise = $pool->promise();

// Force the pool to complete synchronously
$promise->wait();
```

That example is pretty underpowered for the `CommandPool`. Let's try a more complex example. Let's say you want to upload files on disk to an Amazon S3 bucket. To get a list of files from disk, we can use PHP's `DirectoryIterator`. This iterator yields `SplFileInfo` objects. The `CommandPool` accepts an iterator that yields `Aws\CommandInterface` objects, so we map over the `SplFileInfo` objects to return `Aws\CommandInterface` objects.

```php
<?php
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
```
use Aws\S3\S3Client;
use Aws\CommandPool;
use Aws\CommandInterface;
use Aws\ResultInterface;
use GuzzleHttp\Promise\PromiseInterface;

// Create the client
$client = new S3Client([  
  'region'  => 'us-standard',
  'version' => '2006-03-01'
]);

$fromDir = '/path/to/dir';
$toBucket = 'my-bucket';

// Create an iterator that yields files from a directory
/files = new DirectoryIterator($fromDir);

// Create a generator that converts the SplFileInfo objects into
// Aws\CommandInterface objects. This generator accepts the iterator that
// yields files and the name of the bucket to upload the files to.
$commandGenerator = function (
// $files as a file) {
  foreach ($files as $file) {  
    // Skip "." and ".." files
    if ($file->isDot()) {  
      continue;
    }
    $filename = $file->getFilename();
    // Yield a command that is executed by the pool
    yield $client->getCommand('PutObject', [
      'Bucket' => $bucket,
      'Key'  => $file->getFilename(),
      'Body' => fopen($filename, 'r')
    ]);  
  }
}

// Now create the generator using the files iterator
$commands = $commandGenerator($files, $bucket);

// Create a pool and provide an optional array of configuration
$pool = new CommandPool($client, $commands, [
  // Only send 5 files at a time (this is set to 25 by default)
  'concurrency' => 5,
  // Invoke this function before executing each command
  'before' => function (CommandInterface $cmd, $iterKey) {
    echo "About to send {$iterKey}: ";
    print_r($cmd->toArray(), true) . "\n";
  },
  // Invoke this function for each successful transfer
  'fulfilled' => function (ResultInterface $result, $iterKey, $aggregatePromise) {
    echo "Completed {$iterKey}: \n";
  },
  // Invoke this function for each failed transfer
  'rejected' => function (AwsException $reason, $iterKey, $aggregatePromise) {
    echo "Failed {$iterKey}: \n";
  },
]);
// Initiate the pool transfers
$promise = $pool->promise();

// Force the pool to complete synchronously
$promise->wait();

// Or you can chain the calls off of the pool
$promise->then(function() { echo "Done\n"; });

CommandPool Configuration

The 
Aws\CommandPool
 constructor accepts various configuration options.

concurrency (callable|int)

Maximum number of commands to execute concurrently. Provide a function to resize the pool dynamically. The function is provided the current number of pending requests and is expected to return an integer representing the new pool size limit.

before (callable)

Function to invoke before sending each command. The before function accepts the command and the key of the iterator of the command. You can mutate the command as needed in the before function before sending the command.

fulfilled (callable)

Function to invoke when a promise is fulfilled. The function is provided the result object, ID of the iterator that the result came from, and the aggregate promise that can be resolved or rejected if you need to short-circuit the pool.

rejected (callable)

Function to invoke when a promise is rejected. The function is provided an 
Aws\Exception
 object, ID of the iterator that the exception came from, and the aggregate promise that can be resolved or rejected if you need to short-circuit the pool.

Manual Garbage Collection Between Commands

If you are hitting the memory limit with large command pools, this may be due to cyclic references generated by the SDK not yet having been collected by the PHP garbage collector when your memory limit was hit. Manually invoking the collection algorithm between commands may allow the cycles to be collected before hitting that limit. The following example creates a
CommandPool
 that invokes the collection algorithm using a callback before sending each command. Note that invoking the garbage collector does come with a performance cost, and optimal usage will depend on your use case and environment.

```php
$pool = new CommandPool($client, $commands, [
    'concurrency' => 25,
    'before' => function (CommandInterface $cmd, $iterKey) {
        gc_collect_cycles();
    }
]);
```

Promises in the AWS SDK for PHP Version 3

The AWS SDK for PHP uses promises to allow for asynchronous workflows, and this asynchronicity allows HTTP requests to be sent concurrently. The promise specification used by the SDK is Promises/A+. 
What Is a Promise?

A *promise* represents the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its `then` method. This method registers callbacks to receive either a promise’s eventual value or the reason why the promise can’t be fulfilled.

The AWS SDK for PHP relies on the `guzzlehttp/promises` Composer package for its promises implementation. Guzzle promises support blocking and non-blocking workflows and can be used with any non-blocking event loop.

**Note**
HTTP requests are sent concurrently in the AWS SDK for PHP using a single thread, in which non-blocking calls are used to transfer one or more HTTP requests while reacting to state changes (e.g., fulfilling or rejecting promises).

Promises in the SDK

Promises are used throughout the SDK. For example, promises are used in most high-level abstractions provided by the SDK: *paginators* (p. 80), *waiters* (p. 82), *command pools* (p. 61), *multipart uploads* (p. 224), *S3 directory/bucket transfers* (p. 108), and so on.

All of the clients that the SDK provides return promises when you invoke any of the `Async` suffixed methods. For example, the following code shows how to create a promise for getting the results of an Amazon DynamoDB `DescribeTable` operation.

```php
$client = new Aws\DynamoDb\DynamoDbClient([          'region' => 'us-west-2',          'version' => 'latest',      ]);      // This will create a promise that will eventually contain a result      $promise = $client->describeTableAsync(['TableName' => 'mytable']);
```

Notice that you can call either `describeTable` or `describeTableAsync`. These methods are magic `__call` methods on a client that are powered by the API model and version number associated with the client. By calling methods like `describeTable` without the `Async` suffix, the client will block while it sends an HTTP request and either return an `Aws\ResultInterface` object or throw an `Aws\Exception\AwsException`. By suffixing the operation name with `Async` (i.e., `describeTableAsync`) the client will create a promise that is eventually fulfilled with an `Aws\ResultInterface` object or rejected with an `Aws\Exception\AwsException`.

**Important**
When the promise is returned, the result might have already arrived (for example, when using a mock handler), or the HTTP request might not have been initiated.

You can register a callback with the promise by using the `then` method. This method accepts two callbacks, `onFulfilled` and `onRejected`, both of which are optional. The `onFulfilled` callback is invoked if the promise is fulfilled, and the `onRejected` callback is invoked if the promise is rejected (meaning it failed).

```php
$promise->then(      function ($value) {          echo "The promise was fulfilled with \$value";      },      function ($reason) {          echo "The promise was rejected with \$reason";      }    );
```
Executing Commands Concurrently

Multiple promises can be composed together such that they are executed concurrently. This can be achieved by integrating the SDK with a non-blocking event loop, or by building up multiple promises and waiting on them to complete concurrently.

```php
use GuzzleHttp\Promise;

$sdk = new Aws\Sdk([  'version' => 'latest',  'region' => 'us-west-2']);
/sdk = $sdk->createS3();
$ddb = $sdk->createDynamoDb();

$promises = [  'buckets' => $s3->listBucketsAsync(),  'tables' => $ddb->listTablesAsync(),];

// Wait on both promises to complete and return the results
$results = Promise\unwrap($promises);

// Notice that this method will maintain the input array keys
var_dump($results['buckets']->toArray());
var_dump($results['tables']->toArray());
```

Note
---
The [CommandPool](#) provides a more powerful mechanism for executing multiple API operations concurrently.

Chaining Promises

One of the best aspects of promises is that they are composable, allowing you to create transformation pipelines. Promises are composed by chaining `then` callbacks with subsequent `then` callbacks. The return value of a `then` method is a promise that is fulfilled or rejected based on the result of the provided callbacks.

```php
promise = $client->describeTableAsync(['TableName' => 'mytable']);
promise
  ->then(
    function ($value) {
      $value['AddedAttribute'] = 'foo';
      return $value;
    },
    function ($reason) use ($client) {
      // The call failed. You can recover from the error here and
      // return a value that will be provided to the next successful
      // then() callback. Let's retry the call.
      return $client->describeTableAsync(['TableName' => 'mytable']);
    }
  )
  ->then(
    function ($value) {
      // This is only invoked when the previous then callback is
      // fulfilled. If the previous callback returned a promise, then
      // this callback is invoked only after that promise is
      // fulfilled.
      echo $value['AddedAttribute']; // outputs "foo"
    },
  );
```
function ($reason) {
    // The previous callback was rejected (failed).
}

Note
The return value of a promise callback is the $value argument that is supplied to downstream promises. If you want to provide a value to downstream promise chains, you must return a value in the callback function.

Rejection Forwarding
You can register a callback to invoke when a promise is rejected. If an exception is thrown in any callback, the promise is rejected with the exception and the next promises in the chain are rejected with the exception. If you return a value successfully from an $onRejected callback, the next promises in the promise chain is fulfilled with the return value from the $onRejected callback.

Waiting on Promises
You can synchronously force promises to complete by using a promise's wait method.

```php
$promise = $client->listTablesAsync();
$result = $promise->wait();
```

If an exception is encountered while invoking the wait function of a promise, the promise is rejected with the exception and the exception is thrown.

```php
use Aws\Exception\AwsException;

try {
    $result = $promise->wait();
} catch (AwsException $e) {
    // Handle the error
}
```

Calling wait on a promise that has been fulfilled doesn't trigger the wait function. It simply returns the previously delivered value.

```php
$promise = $client->listTablesAsync();
$result = $promise->wait();
assert($result === $promise->wait());
```

Calling wait on a promise that has been rejected throws an exception. If the rejection reason is an instance of \Exception the reason is thrown. Otherwise, a GuzzleHttp\Promise\RejectionException is thrown and the reason can be obtained by calling the getReason method of the exception.

Note
API operation calls in the AWS SDK for PHP are rejected with subclasses of the Aws\Exception\AwsException class. However, it's possible that the reason delivered to a then method is different because the addition of a custom middleware that alters a rejection reason.

Canceling Promises
Promises can be canceled using the cancel() method of a promise. If a promise has already been resolved, calling cancel() will have no effect. Canceling a promise cancels the promise and any
Combining Promises

You can combine promises into aggregate promises to build more sophisticated workflows. The 
guzzlehttp/promise package contains various functions that you can use to combine promises.

You can find the API documentation for all of the promise collection functions at namespace-
GuzzleHttp.Promise.

each and each_limit

Use the CommandPool (p. 61) when you have a task queue of Aws\CommandInterface commands
to perform concurrently with a fixed pool size (the commands can be in memory or yielded by a lazy
iterator). The CommandPool ensures that a fixed number of commands are sent concurrently until the
supplied iterator is exhausted.

The CommandPool works only with commands that are executed by the same client. You can use
the GuzzleHttp\Promise\each_limit function to perform send commands of different clients
concurrently using a fixed pool size.

```php
use GuzzleHttp\Promise;

$sdk = new Aws\Sdk([    'version' => 'latest',    'region' => 'us-west-2']);

$s3 = $sdk->createS3();
$dynamodb = $sdk->createDynamoDb();

// Create a generator that yields promises
$promiseGenerator = function () use ($s3, $dynamodb) {    yield $s3->listBucketsAsync();    yield $dynamodb->listTablesAsync();    // yield other promises as needed...
};

// Execute the tasks yielded by the generator concurrently while limiting the
// maximum number of concurrent promises to 5
$promise = Promise\each_limit($promiseGenerator(), 5);

// Waiting on an EachPromise will wait on the entire task queue to complete
$promise->wait();
```

Promise Coroutines

One of the more powerful features of the Guzzle promises library is that it allows you to use promise
coroutines that make writing asynchronous workflows seem more like writing traditional synchronous
workflows. In fact, the AWS SDK for PHP uses coroutine promises in most of the high-level abstractions.

Imagine you wanted to create several buckets and upload a file to the bucket when the bucket becomes
available, and you’d like to do this all concurrently so that it happens as fast as possible. You can do this
easily by combining multiple coroutine promises together using the all() promise function.

```php
use GuzzleHttp\Promise;
```
$uploadFn = function ($bucket) use ($s3Client) {
    return Promise\coroutine(function () use ($bucket, $s3Client) {
        // You can capture the result by yielding inside of parens
        $result = (yield $s3Client->createBucket([\'Bucket\' => $bucket]));
        // Wait on the bucket to be available
        $waiter = $s3Client->getWaiter(\'BucketExists\', [\'Bucket\' => $bucket]);
        // Wait until the bucket exists
        yield $waiter->promise();
        // Upload a file to the bucket
        yield $s3Client->putObjectAsync([
            \'Bucket\' => $bucket,
            \'Key\'    => \'_placeholder\',
            \'Body\'   => \'Hi!\'
        ]);};
    });
}

// Create the following buckets
$buckets = [\'foo\', \'baz\', \'bar\'];
$promises = [];

// Build an array of promises
foreach ($buckets as $bucket) {
    $promises[] = $uploadFn($bucket);
}

// Aggregate the promises into a single "all" promise
$aggregate = Promise\all($promises);

// You can then() off of this promise or synchronously wait
$aggregate->wait();

Handlers and Middleware in the AWS SDK for PHP Version 3

The primary mechanism for extending the AWS SDK for PHP is through handlers and middleware. Each SDK client class owns an Aws\HandlerList instance that is accessible through the getHandlerList() method of a client. You can retrieve a client’s HandlerList and modify it to add or remove client behavior.

Handlers

A handler is a function that performs the actual transformation of a command and request into a result. A handler typically sends HTTP requests. Handlers can be composed with middleware to augment their behavior. A handler is a function that accepts an Aws\CommandInterface and a Psr\Http\Message\RequestInterface and returns a promise that is fulfilled with an Aws\ResultInterface or rejected with an Aws\Exception\AwsException reason.

Here’s a handler that returns the same mock result for each call.

```php
use Aws\CommandInterface;
use Aws\Result;
use Psr\Http\Message\RequestInterface;
use GuzzleHttp\Promise;

$myHandler = function (CommandInterface $cmd, RequestInterface $request) {
    $result = new Result([\'foo\' => \'bar\']);
    return Promise\promise_for($result);
};
```
You can then use this handler with an SDK client by providing a `handler` option in the constructor of a client.

```php
// Set the handler of the client in the constructor
$s3 = new Aws\S3\S3Client(
    ['region'  => 'us-east-1',
     'version' => '2006-03-01',
     'handler' => $myHandler
]);
```

You can also change the handler of a client after it is constructed using the `setHandler` method of an `Aws\ClientInterface`.

```php
// Set the handler of the client after it is constructed
$s3->getHandlerList()->setHandler($myHandler);
```

## Mock Handler

We recommend using the `MockHandler` when writing tests that use the SDK. You can use the `Aws\MockHandler` to return mocked results or throw mock exceptions. You enqueue results or exceptions, and the MockHandler dequeues them in FIFO order.

```php
use Aws\Result;
s用自己的MockHandler;
use Aws\DynamoDb\DynamoDbClient;
use Aws\CommandInterface;
use Psr\Http\Message\RequestInterface;
use Aws\Exception\AwsException;

$mock = new MockHandler();
// Return a mocked result
$mock->append(new Result(['foo' => 'bar']));
// You can provide a function to invoke; here we throw a mock exception
$mock->append(function (CommandInterface $cmd, RequestInterface $req) {
    return new AwsException('Mock exception', $cmd);
});
// Create a client with the mock handler
$client = new DynamoDbClient(
    ['region'  => 'us-west-2',
     'version' => 'latest',
     'handler' => $mock
]);
// Result object response will contain ['foo' => 'bar']
$result = $client->listTables();
// This will throw the exception that was enqueued
$client->listTables();
```

## Middleware

Middleware is a special type of high-level function that augments the behavior of transferring a command, and delegates to a "next" handler. Middleware functions accept an `Aws\CommandInterface`.
and a \Psr\Http\Message\RequestInterface and return a promise that is fulfilled with an \Aws\ResultInterface or rejected with an \Aws\Exception\AwsException reason.

A middleware is a higher-order function that modifies a command, request, or result as it passes through the middleware. A middleware takes the following form.

```php
use Aws\CommandInterface;
use Psr\Http\Message\RequestInterface;

$middleware = function () {
    return function (callable $handler) use ($fn) {
        return function (
            CommandInterface $command,
            RequestInterface $request = null
        ) use ($handler, $fn) {
            // Do something before calling the next handler
            // ...
            $promise = $fn($command, $request);
            // Do something in the promise after calling the next handler
            // ...
            return $promise;
        };
    };
};
```

A middleware receives a command to execute and an optional request object. The middleware can choose to augment the request and command or leave them as-is. A middleware then invokes the next handler in the chain or can choose to short-circuit the next handler and return a promise. The promise that is created by invoking the next handler can then be augmented using the `then` method of the promise to modify the eventual result or error before returning the promise back up the stack of middleware.

**HandlerList**

The SDK uses an \Aws\HandlerList to manage the middleware and handlers used when executing a command. Each SDK client owns a HandlerList, and this HandlerList is cloned and added to each command that a client creates. You can attach a middleware and default handler to use for each command created by a client by adding a middleware to the client's HandlerList. You can add and remove middleware from specific commands by modifying the HandlerList owned by a specific command.

A HandlerList represents a stack of middleware that are used to wrap a handler. To help manage the list of middleware and the order in which they wrap a handler, the HandlerList breaks the middleware stack into named steps that represents part of the lifecycle of transferring a command:

1. init - Add default parameters
2. validate - Validate required parameters
3. build - Serialize an HTTP request for sending
4. sign - Sign the serialized HTTP request
5. <handler> (not a step, but performs the actual transfer)

**init**

This lifecycle step represents the initialization of a command, and a request has not yet been serialized. This step is typically used to add default parameters to a command.

You can add a middleware to the init step using the `appendInit` and `prependInit` methods, where `appendInit` adds the middleware to the end of the prepend list while `prependInit` adds the middleware to the front of the prepend list.
use Aws\Middleware;

$middleware = Middleware::tap(function ($cmd, $req) {
    // Observe the step
});

// Append to the end of the step with a custom name
$client->getHandlerList()->appendInit($middleware, 'custom-name');
// Prepend to the beginning of the step
$client->getHandlerList()->prependInit($middleware, 'custom-name');

validate

This lifecycle step is used for validating the input parameters of a command.

You can add a middleware to the validate step using the appendValidate and prependValidate methods, where appendValidate adds the middleware to the end of the validate list while prependValidate adds the middleware to the front of the validate list.

use Aws\Middleware;

$middleware = Middleware::tap(function ($cmd, $req) {
    // Observe the step
});

// Append to the end of the step with a custom name
$client->getHandlerList()->appendValidate($middleware, 'custom-name');
// Prepend to the beginning of the step
$client->getHandlerList()->prependValidate($middleware, 'custom-name');

build

This lifecycle step is used to serialize an HTTP request for the command being executed. Downstream lifecycle events will receive a command and PSR-7 HTTP request.

You can add a middleware to the build step using the appendBuild and prependBuild methods, where appendBuild adds the middleware to the end of the build list while prependBuild adds the middleware to the front of the build list.

use Aws\Middleware;

$middleware = Middleware::tap(function ($cmd, $req) {
    // Observe the step
});

// Append to the end of the step with a custom name
$client->getHandlerList()->appendBuild($middleware, 'custom-name');
// Prepend to the beginning of the step
$client->getHandlerList()->prependBuild($middleware, 'custom-name');

sign

This lifecycle step is typically used to sign HTTP requests before they are sent over the wire. You should typically refrain from mutating an HTTP request after it is signed to avoid signature errors.

This is the last step in the HandlerList before the HTTP request is transferred by a handler.

You can add a middleware to the sign step using the appendSign and prependSign methods, where appendSign adds the middleware to the end of the sign list while prependSign adds the middleware to the front of the sign list.
use Aws\Middleware;

$middleware = Middleware::tap(function ($cmd, $req) {
    // Observe the step
});

// Append to the end of the step with a custom name
$client->getHandlerList()->appendSign($middleware, 'custom-name');
// Prepend to the beginning of the step
$client->getHandlerList()->prependSign($middleware, 'custom-name');

### Available Middleware

The SDK provides several middleware that you can use to augment the behavior of a client or to observe the execution of a command.

#### mapCommand

The Aws\Middleware::mapCommand middleware is useful when you need to modify a command before the command is serialized as an HTTP request. For example, mapCommand could be used to perform validation or add default parameters. The mapCommand function accepts a callable that accepts an Aws\CommandInterface object and returns an Aws\CommandInterface object.

use Aws\Middleware;
use Aws\CommandInterface;

// Here we've omitted the require Bucket parameter. We'll add it in the
// custom middleware.
$command = $s3Client->getCommand('HeadObject', ['Key' => 'test']);

// Apply a custom middleware named "add-param" to the "init" lifecycle step
$command->getHandlerList()->appendInit(
    Middleware::mapCommand(function (CommandInterface $command) {
        $command['Bucket'] = 'mybucket';
        // Be sure to return the command!
        return $command;
    }),
    'add-param'
);

#### mapRequest

The Aws\Middleware::mapRequest middleware is useful when you need to modify a request after it is serialized but before it is sent. For example, this can be used to add custom HTTP headers to a request. The mapRequest function accepts a callable that accepts a Psr\Http\Message\RequestInterface argument and returns a Psr\Http\Message\RequestInterface object.

use Aws\Middleware;
use Psr\Http\Message\RequestInterface;

// Create a command so that we can access the handler list
$command = $s3Client->getCommand('HeadObject', [
    'Key' => 'test',
    'Bucket' => 'mybucket'
]);

// Apply a custom middleware named "add-header" to the "build" lifecycle step
$command->getHandlerList()->appendBuild(
    Middleware::mapRequest(function (RequestInterface $request) {
        // Add custom headers
        $request->setHeader('User-Agent', 'My User Agent');
        // Be sure to return the request!
        return $request;
    }),
    'add-header'
);
Middleware::mapRequest(function (RequestInterface $request) {
    // Return a new request with the added header
    return $request->withHeader('X-Foo-Baz', 'Bar');
});

'add-header'};

Now when the command is executed, it is sent with the custom header.

Important
Notice that the middleware was appended to the handler list at the end of build step. This is to ensure that a request has been built before this middleware is invoked.

mapResult

The Aws\Middleware::mapResult middleware is useful when you need to modify the result of a command execution. The mapResult function accepts a callable that accepts an Aws\ResultInterface argument and returns an Aws\ResultInterface object.

use Aws\Middleware;
use Aws\ResultInterface;

$command = $s3Client->getCommand('HeadObject', [
    'Key'    => 'test',
    'Bucket' => 'mybucket'
]);

$command->getHandlerList()->appendSign(Middleware::mapResult(function (ResultInterface $result) {
    // Add a custom value to the result
    $result['foo'] = 'bar';
    return $result;
});

Now when the command is executed, the returned result will contain a foo attribute.

history

The history middleware is useful for testing that the SDK executed the commands you expected, sent the HTTP requests you expected, and received the results you expected. It’s essentially a middleware that acts similarly to the history of a web browser.

use Aws\History;
use Aws\Middleware;

$ddb = new Aws\DynamoDb\DynamoDbClient([  
    'version' => 'latest',
    'region'  => 'us-west-2'
]);

// Create a history container to store the history data
$history = new History();

// Add the history middleware that uses the history container
$ddb->getHandlerList()->appendSign(Middleware::history($history));

An Aws\History history container stores 10 entries by default before purging entries. You can customize the number of entries by passing in the number of entries to persist to the constructor.
// Create a history container that stores 20 entries
$history = new History(20);

You can inspect the history container after executing requests that pass the history middleware.

// The object is countable, returning the number of entries in the container
count($history);

// The object is iterable, yielding each entry in the container
foreach ($history as $entry) {
    // You can access the command that was executed
    var_dump($entry['command']);
    // The request that was serialized and sent
    var_dump($entry['request']);
    // The result that was received (if successful)
    var_dump($entry['result']);
    // The exception that was received (if a failure occurred)
    var_dump($entry['exception']);
}

// You can get the last Aws\CommandInterface that was executed. This method
// will throw an exception if no commands have been executed.
$command = $history->getLastCommand();

// You can get the last request that was serialized. This method will throw an exception
// if no requests have been serialized.
$request = $history->getLastRequest();

// You can get the last return value (an Aws\ResultInterface or Exception).
// The method will throw an exception if no value has been returned for the last
// executed operation (e.g., an async request has not completed).
$result = $history->getLastReturn();

// You can clear out the entries using clear
$history->clear();

tap

The tap middleware is used as an observer. You can use this middleware to invoke functions when
sending commands through the chain of middleware. The tap function accepts a callable that accepts
the Aws\CommandInterface and an optional Psr\Http\Message\RequestInterface that is being
executed.

use Aws\Middleware;
$s3 = new Aws\S3\S3Client(
    'region' => 'us-east-1',
    'version' => '2006-03-01'
);
$handlerList = $s3->getHandlerList();

// Create a tap middleware that observes the command at a specific step
$handlerList->appendInit(   Middleware::tap(function (CommandInterface $cmd, RequestInterface $req = null) {
        echo 'About to send: ' . $cmd->getName() . "\n";
        if ($req) {
            echo 'HTTP method: ' . $request->getMethod() . "\n";
        }
    }));
Creating Custom handlers

A handler is simply a function that accepts an \texttt{Aws\CommandInterface} object and \texttt{Psr\Http\Message\RequestInterface} object, and returns a \texttt{GuzzleHttp\Promise\PromiseInterface} that is fulfilled with an \texttt{Aws\ResultInterface} or rejected with an \texttt{Aws\Exception\AwsException}.

Although the SDK has several @http options, a handler only needs to know how to use the following options:

- connect\_timeout (p. 31)
- debug (p. 31)
- decode\_content (p. 32) (optional)
- delay (p. 32)
- progress (p. 33) (optional)
- proxy (p. 33)
- sink (p. 34)
- synchronous (p. 34) (optional)
- stream (p. 35) (optional)
- timeout (p. 35)
- verify (p. 35)
- http\_stats\_receiver (optional) - A function to invoke with an associative array of HTTP transfer statistics if requested using the \texttt{stats} (p. 27) configuration parameter.

Unless the option is specified as optional, a handler MUST be able to handle the option or it MUST return a rejected promise.

In addition to handling specific @http options, a handler MUST add a \texttt{User-Agent} header that takes the following form, where "3.X" can be replaced with \texttt{Aws\Sdk::\textasciitilde VERSION} and "HandlerSpecificData/\textasciitilde version \ldots" should be replaced with your handler-specific User-Agent string.

\texttt{User-Agent: aws-sdk-php/3.X HandlerSpecificData/version \ldots}

Streams in the AWS SDK for PHP Version 3

As part of its integration of the \texttt{PSR-7} HTTP message standard, the AWS SDK for PHP uses the \texttt{PSR-7 Stream\Interface} internally as its abstraction over PHP streams. Any command with an input field defined as a blob, such as the \texttt{Body} parameter on an \texttt{S3::PutObject command}, can be satisfied with a string, a PHP stream resource, or an instance of \texttt{Psr\Http\Message\StreamInterface}.

\textbf{Warning}

The SDK takes ownership of any raw PHP stream resource supplied as an input parameter to a command. The stream is consumed and closed on your behalf. If you need to share a stream between an SDK operation and your code, wrap it in an instance of \texttt{GuzzleHttp\Par\Stream} before including it as a command parameter. The SDK consumes the stream, so your code needs to account for movement of the stream's internal cursor. Guzzle streams call \texttt{fclose} on the underlying stream resource when they are destroyed by PHP's garbage collector, so you do not need to close the stream yourself.

Stream Decorators

Guzzle provides several stream decorators that you can use to control how the SDK and Guzzle interact with the streaming resource provided as an input parameter to a command. These decorators can modify
how handlers are able to read and seek on a given stream. The following is a partial list; more can be found on the GuzzleHttpPsr7 repository.

### AppendStream

**GuzzleHttp\Psr7\AppendStream**

Reads from multiple streams, one after the other.

```php
use GuzzleHttp\Psr7;

$a = Psr7\stream_for('abc, ');  
$b = Psr7\stream_for('123.');  
$composed = new Psr7\AppendStream([$a, $b]);

$composed->addStream(Psr7\stream_for(' Above all listen to me'));

echo $composed(); // abc, 123. Above all listen to me.
```

### CachingStream

**GuzzleHttp\Psr7\CachingStream**

Used to allow seeking over previously read bytes on non-seekable streams. This can be useful when transferring a non-seekable entity body fails due to needing to rewind the stream (for example, resulting from a redirect). Data that is read from the remote stream is buffered in a PHP temp stream so that previously read bytes are cached first in memory, then on disk.

```php
use GuzzleHttp\Psr7;

$original = Psr7\stream_for(fopen('http://www.google.com', 'r'));  
$stream = new Psr7\CachingStream($original);

$stream->read(1024);

echo $stream->tell(); // 1024

$stream->seek(0);

echo $stream->tell(); // 0
```

### InflateStream

**GuzzleHttp\Psr7\InflateStream**

Uses PHP's zlib.inflate filter to inflate or deflate gzipped content.

This stream decorator skips the first 10 bytes of the given stream to remove the gzip header, converts the provided stream to a PHP stream resource, and then appends the zlib.inflate filter. The stream is then converted back to a Guzzle stream resource to be used as a Guzzle stream.

### LazyOpenStream

**GuzzleHttp\Psr7\LazyOpenStream**

Lazily reads or writes to a file that is opened only after an I/O operation takes place on the stream.
use GuzzleHttp\Psr7;

$stream = new Psr7\LazyOpenStream('/path/to/file', 'r');
// The file has not yet been opened...

echo $stream->read(10); // The file is opened and read from only when needed.

## LimitStream

GuzzleHttp\Psr7\LimitStream

Used to read a subset or slice of an existing stream object. This can be useful for breaking a large file into smaller pieces to be sent in chunks (e.g., the Amazon S3 Multipart Upload API).

use GuzzleHttp\Psr7;

$original = Psr7\stream_for(fopen('/tmp/test.txt', 'r+'));
echo $original->getSize(); // >>> 1048576

// Limit the size of the body to 1024 bytes and start reading from byte 2048
$stream = new Psr7\LimitStream($original, 1024, 2048);
echo $stream->getSize(); // >>> 1024
echo $stream->tell(); // >>> 0

## NoSeekStream

GuzzleHttp\Psr7\NoSeekStream

Wraps a stream and does not allow seeking.

use GuzzleHttp\Psr7;

$original = Psr7\stream_for('foo');
$noSeek = new Psr7\NoSeekStream($original);

echo $noSeek->read(3); // foo
var_export($noSeek->isSeekable()); // false
$noSeek->seek(0);
var_export($noSeek->read(3)); // NULL

## PumpStream

GuzzleHttp\Psr7\PumpStream

Provides a read-only stream that pumps data from a PHP callable.

When invoking the provided callable, the PumpStream passes the amount of data requested to read to the callable. The callable can choose to ignore this value and return fewer or more bytes than requested. Any extra data returned by the provided callable is buffered internally until drained using the read() function of the PumpStream. The provided callable MUST return false when there is no more data to read.
Implementing Stream Decorators

Creating a stream decorator is very easy thanks to the GuzzleHttp\Psr7\StreamDecoratorTrait. This trait provides methods that implement Psr\Http\Message\StreamInterface by proxying to an underlying stream. Just use the StreamDecoratorTrait and implement your custom methods.

For example, let’s say we wanted to call a specific function each time the last byte is read from a stream. This could be implemented by overriding the read() method.

```
use Psr\Http\Message\StreamInterface;
use GuzzleHttp\Psr7\StreamDecoratorTrait;

class EofCallbackStream implements StreamInterface
{
    use StreamDecoratorTrait;

    private $callback;

    public function __construct(StreamInterface $stream, callable $cb)
    {
        $this->stream = $stream;
        $this->callback = $cb;
    }

    public function read($length)
    {
        $result = $this->stream->read($length);

        // Invoke the callback when EOF is hit
        if ($this->eof()) {
            call_user_func($this->callback);
        }

        return $result;
    }
}
```

This decorator could be added to any existing stream and used like this.

```
use GuzzleHttp\Psr7;

$original = Psr7\stream_for('foo');

$eofStream = new EofCallbackStream($original, function () {
    echo 'EOF!';
});

$eofStream->read(2);
$eofStream->read(1);
// echoes "EOF!"
$eofStream->seek(0);
$eofStream->read(3);
// echoes "EOF!"
```

Paginators in the AWS SDK for PHP Version 3

Some AWS service operations are paginated and respond with truncated results. For example, the Amazon S3ListObjects operation only returns up to 1,000 objects at a time. Operations like these (typically prefixed with “list” or “describe”) require making subsequent requests with token (or marker) parameters to retrieve the entire set of results.
**Paginators** are a feature of the AWS SDK for PHP that act as an abstraction over this process to make it easier for developers to use paginated APIs. A paginator is essentially an iterator of results. They are created via the `getPaginator()` method of the client. When you call `getPaginator()`, you must provide the name of the operation and the operation's arguments (in the same way you do when you execute an operation). You can iterate over a paginator object using `foreach` to get individual `Aws\Result` objects.

```php
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'my-bucket'
]);
foreach ($results as $result) {
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
}
```

**Paginator Objects**

The object returned by `getPaginator()` method is an instance of the `Aws\ResultPaginator` class. This class implements PHP's native `iterator` interface, which is why it works with `foreach`. It can also be used with iterator functions, like `iterator_to_array`, and integrates well with SPL iterators like the `LimitIterator` object.

Paginator objects hold only one “page” of results at a time and are executed lazily. This means that they make only as many requests as they need to yield the current page of results. For example, the Amazon S3 `ListObjects` operation only returns up to 1,000 objects at a time, so if your bucket has ~10,000 objects, the paginator would need to do 10 requests total. When you iterate through the results, the first request is executed when you start iterating, the second in the second iteration of the loop, and so on.

**Enumerating Data from Results**

Paginator objects have a method named `search()`, which allows you to create iterators for data within a set of results. When you call `search()`, provide a JMESPath expression (p. 83) to specify what data to extract. Calling `search()` returns an iterator that yields the results of the expression on each page of results. This is evaluated lazily, as you iterate through the returned iterator.

The following example is equivalent to the preceding code example, but uses the `ResultPaginator::search()` method to be more concise.

```php
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'my-bucket'
]);
foreach ($results->search('Contents[].Key') as $key) {
    echo $key . "\n";
}
```

JMESPath expressions enable you to do fairly complex things. For example, if you wanted to print all of the object keys and common prefixes (i.e., do an `ls` of a bucket), you could do the following.

```php
// List all prefixes ("directories") and objects ("files") in the bucket
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'my-bucket',
    'Delimiter' => '/'
]);
```
Asynchronous Pagination

You can iterate over the results of a paginator asynchronously by providing a callback for the `each()` method of an `Aws\ResultPaginator`. The callback is invoked for each value that is yielded by the paginator.

```php
$expression = '[$CommonPrefixes[].Prefix, Contents[].Key][]';
foreach ($results->search($expression) as $item) {
    echo $item . "\n";
}
```

### Note

Using the `each()` method allows you to paginate over the results of an API operation while concurrently sending other requests asynchronously.

A non-null return value from the callback will be yielded by the underlying coroutine-based promise. This means that you can return promises from the callback that must be resolved before continuing iteration over the remaining items, essentially merging in other promises to the iteration. The last non-null value returned by the callback is the result that fulfills the promise to any downstream promises. If the last return value is a promise, the resolution of that promise is the result that fulfills or rejects downstream promises.

```php
// Delete all keys that end with "Foo"
$promise = $results->each(function ($result) use ($s3Client) {
    if (substr($result['Key'], -3) === 'Foo') {
        // Merge this promise into the iterator
        return $s3Client->deleteAsync(['Bucket' => 'my-bucket', 'Key' => 'Foo']);
    }
});
$promise->then(function ($result) {
    // Result would be the last result to the deleteAsync operation
}
->otherwise(function ($reason) {
    // Reason would be an exception that was encountered either in the
    // call to deleteAsync or calls performed while iterating
});

// Forcing a synchronous wait will also wait on all of the deleteAsync calls
$promise->wait();
```

Waiters in the AWS SDK for PHP Version 3

Waiters help make it easier to work with eventually consistent systems by providing an abstracted way to wait until a resource enters into a particular state by polling the resource. You can find a list of the
waitees supported by a client by viewing the API documentation for a single version of a service client. To navigate there, go to the client's page in the API documentation and navigate to the specifie version number (represented by a date) and scroll down to the 'Waiters' section. This link will bring you to the waiters section of S3.

In the following example, the Amazon S3 client is used to create a bucket. Then the waiter is used to wait until the bucket exists.

```php
// Create a bucket
$s3Client->createBucket(['Bucket' => 'my-bucket']);

// Wait until the created bucket is available
$s3Client->waitUntil('BucketExists', ['Bucket' => 'my-bucket']);
```

If the waiter has to poll the bucket too many times, it will throw a `RuntimeException` exception.

## Waiter Configuration

Waiters are driven by an associative array of configuration options. All of the options used by a particular waiter have default values, but they can be overridden to support different waiting strategies.

You can modify waiter configuration options by passing an associative array of `@waiter` options to the `$args` argument of a client's `waitUntil()` and `getWaiter()` methods.

```php
// Providing custom waiter configuration options to a waiter
$s3Client->waitUntil('BucketExists', [
    'Bucket' => 'my-bucket',
    '@waiter' => [
        'delay' => 3,
        'maxAttempts' => 10
    ]
]);
```

### delay (int)

Number of seconds to delay between polling attempts. Each waiter has a default delay configuration value, but you might need to modify this setting for specific use cases.

### maxAttempts (int)

Maximum number of polling attempts to issue before failing the waiter. This option ensures that you do not wait on a resource indefinitely. Each waiter has a default maxAttempts configuration value, but you might need to modify this setting for specific use cases.

### initDelay (int)

Amount of time in seconds to wait before the first polling attempt. This might be useful when waiting on a resource that you know will take awhile to enter into the desired state.

### before (callable)

A PHP callable function that is invoked before each attempt. The callable is invoked with the `Aws\CommandInterface` command that is about to be executed and the number of attempts that have been executed so far. Uses of the `before` callable might be to modify commands before they are executed or provide progress information.

```php
use Aws\CommandInterface;

$s3Client->waitUntil('BucketExists', [
```
Waiting Asynchronously

In addition to waiting synchronously, you can invoke a waiter to wait asynchronously while sending other requests or waiting on multiple resources at once.

You can access a waiter promise by retrieving a waiter from a client using the client's \getWaiter\($\text{name}$, array $\text{args} = []$) method. Use the $\text{promise()}$ method of a waiter to initiate the waiter. A waiter promise is fulfilled with the last \Aws\CommandInterface that was executed in the waiter, and rejected with a \RuntimeException on error.

```php
use \Aws\CommandInterface;

$waiterName = 'BucketExists';
$waiterOptions = ['Bucket' => 'my-bucket'];

// Create a waiter promise
$waiter = $s3Client->getWaiter($waiterName, $waiterOptions);

// Initiate the waiter and retrieve a promise
$promise = $waiter->promise();

// Call methods when the promise is resolved.
$promise
    ->then(function () {
        echo "Waiter completed\n";
    })
    ->otherwise(function (\Exception $e) {
        echo "Waiter failed: \" . $e . "\n";
    });

// Block until the waiter completes or fails. Note that this might throw
// a \RuntimeException if the waiter fails.
$promise->wait();
```

Exposing a promise-based waiters API allows for some powerful and relatively low overhead use cases. For example, what if you wanted to wait on multiple resources, and do something with the first waiter that successfully resolved?

```php
use \Aws\CommandInterface;

// Create an array of waiter promises
$promises = [
    $s3Client->getWaiter('BucketExists', ['Bucket' => 'a'])->promise(),
    $s3Client->getWaiter('BucketExists', ['Bucket' => 'b'])->promise(),
    $s3Client->getWaiter('BucketExists', ['Bucket' => 'c'])->promise()
];

// Initiate a race between the waiters, fulfilling the promise with the
```
JMESPath Expressions in the AWS SDK for PHP Version 3

JMESPath enables you to declaratively specify how to extract elements from a JSON document. The AWS SDK for PHP has a dependency on jmespath.php to power some of the high-level abstractions like Paginators in the AWS SDK for PHP Version 3 (p. 78) and Waiters in the AWS SDK for PHP Version 3 (p. 80), but also exposes JMESPath searching on Aws\ResultInterface and Aws\ResultPaginator.

You can play around with JMESPath in your browser by trying the online JMESPath examples. You can learn more about the language, including the available expressions and functions, in the JMESPath specification.

The AWS CLI supports JMESPath. Expressions you write for CLI output are 100 percent compatible with expressions written for the AWS SDK for PHP.

Extracting Data from Results

The Aws\ResultInterface interface has a search($expression) method that extracts data from a result model based on a JMESPath expression. Using JMESPath expressions to query the data from a result object can help to remove boilerplate conditional code, and more concisely express the data that is being extracted.

To demonstrate how it works, we'll start with the default JSON output below, which describes two Amazon Elastic Block Store (Amazon EBS) volumes attached to separate Amazon EC2 instances.

```php
$result = $ec2Client->describeVolumes();
// Output the result data as JSON (just so we can clearly visualize it)
echo json_encode($result->toArray(), JSON_PRETTY_PRINT);
```

```json
{
    "Volumes": [
        {
            "AvailabilityZone": "us-west-2a",
            "Attachments": [
                {
                    "AttachTime": "2013-09-17T00:55:03.000Z",
                    "InstanceId": "i-a071c394",
                    "VolumeId": "vol-e11a5288",
                    "State": "attached",
                    "DeleteOnTermination": true,
                    "Device": "/dev/sda1"
                }
            ]
        }
    }
}
```
First, we can retrieve only the first volume from the Volumes list with the following command.

```php
$firstVolume = $result->search('Volumes[0]');
```

Now, we use the wildcard-index expression `[*]` to iterate over the entire list and also extract and rename three elements: VolumeId is renamed to ID, AvailabilityZone is renamed to AZ, and Size remains Size. We can extract and rename these elements using a multi-hash expression placed after the wildcard-index expression.

```php
$data = $result->search('Volumes[*].{ID: VolumeId, AZ: AvailabilityZone, Size: Size}');
```

This gives us an array of PHP data like the following:

```php
array(2) {
 [0] =>
  array(3) {
   'AZ' =>
    string(10) "us-west-2a"
   'ID' =>
    string(12) "vol-e11a5288"
   'Size' =>
    int(30)
  }
}
```
Extracting Data from Results

In the multi-hash notation, you can also use chained keys such as key1.key2[0].key3 to extract elements deeply nested within the structure. The following example demonstrates this with the Attachments[0].InstanceId key, aliased to simply InstanceId. (In most cases, JMESPath expressions will ignore whitespace.)

```php
$expr = 'Volumes[*].{ID: VolumeId,
    InstanceId: Attachments[0].InstanceId,
    AZ: AvailabilityZone,
    Size: Size}';
$data = $result->search($expr);
var_dump($data);
```

The previous expression will output the following data:

```php
array(2) {
    [0] =>
        array(4) {
            'ID' =>
                string(12) "vol-e11a5288"
            'InstanceId' =>
                string(10) "i-a071c394"
            'AZ' =>
                string(10) "us-west-2a"
            'Size' =>
                int(30)
        }
    [1] =>
        array(4) {
            'ID' =>
                string(12) "vol-2e410a47"
            'InstanceId' =>
                string(10) "i-4b41a37c"
            'AZ' =>
                string(10) "us-west-2a"
            'Size' =>
                int(8)
        }
}
```

You can also filter multiple elements with the multi-list expression: [key1, key2]. This formats all filtered attributes into a single ordered list per object, regardless of type.

```php
$expr = 'Volumes[*].[VolumeId, Attachments[0].InstanceId, AvailabilityZone, Size]';
$data = $result->search($expr);
var_dump($data);
```

Running the previous search produces the following data:
Use a filter expression to filter results by the value of a specific field. The following example query outputs only volumes in the us-west-2a Availability Zone.

```
$data = $result->search("Volumes[?AvailabilityZone ## 'us-west-2a']");
```

JMESPath also supports function expressions. Let's say you want to run the same query as above, but instead retrieve all volumes in which the volume is in an AWS Region that starts with "us-". The following expression uses the `starts_with` function, passing in a string literal of `us-`. This function's result is then compared against the JSON literal value of `true`, passing only results of the filter predicate that returned `true` through the filter projection.

```
$data = $result->search('Volumes[?starts_with(AvailabilityZone, 'us-') ## `true`]');
```

### Extracting Data from paginators

As you know from the [Paginators in the AWS SDK for PHP Version 3 (p. 78)](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/paginators.html) guide, `Aws\ResultPaginator` objects are used to yield results from a pageable API operation. The AWS SDK for PHP enables you to extract and iterate over filtered data from `Aws\ResultPaginator` objects, essentially implementing a flat-map over the iterator in which the result of a JMESPath expression is the map function.

Let's say you want to create an iterator that yields only objects from a bucket that are larger than 1 MB. This can be achieved by first creating a `ListObjects` paginator and then applying a `search()` function to the paginator, creating a flat-mapped iterator over the paginated data.

```
$result = $s3Client->getPaginator('ListObjects', ['Bucket' => 't1234']);
.filtered = $result->search('Contents[?Size > `1048576`]');

// The result yielded as $data will be each individual match from
// Contents in which the Size attribute is > 1048576
foreach ($filtered as $data) {
    var_dump($data);
}
```
SDK Metrics in the AWS SDK for PHP Version 3

AWS SDK Metrics for Enterprise Support (SDK Metrics) enables enterprise customers to collect metrics from AWS SDKs on their hosts and clients shared with AWS Enterprise Support. SDK Metrics provides information that helps speed up detection and diagnosis of issues occurring in connections to AWS services for AWS Enterprise Support customers.

As telemetry is collected on each host, it is relayed via UDP to localhost, where the CloudWatch agent aggregates the data and sends it to the SDK Metrics service. Therefore, to receive metrics, you must add the CloudWatch agent to your instance.

Learn more about SDK Metrics in the Amazon CloudWatch User Guide.

Each of the following topics describes how to set up, configure, and manage SDK Metrics for the AWS SDK for PHP.

Topics

- Authorize SDK Metrics to Collect and Send Metrics in the AWS SDK for PHP Version 3 (p. 87)
- Set Up SDK Metrics for the AWS SDK for PHP Version 3 (p. 89)
- Definitions for SDK Metrics (p. 93)

Authorize SDK Metrics to Collect and Send Metrics in the AWS SDK for PHP Version 3

To collect metrics from AWS SDKs using AWS SDK Metrics for Enterprise Support Enterprise customers must create an IAM Role that gives CloudWatch agent permission to gather data from their Amazon EC2 instance or production environment.

Use the following PHP code sample or the AWS Console to create an IAM Policy and Role for an CloudWatch agent to access SDK Metrics in your environment.

Learn more about using SDK Metrics with AWS SDK for PHP in Set Up SDK Metrics for the AWS SDK for PHP Version 3 (p. 89). For more information about SDK Metrics see IAM Permissions for SDK Metrics in the Amazon CloudWatch User Guide.

Set Up Access Permissions Using the AWS SDK for PHP

Create an IAM role for the instance that has permission for Amazon EC2 Systems Manager and SDK Metrics.

First, create a policy using CreatePolicy. Then create a role using CreateRole. Finally, attach the policy you created to your new role with AttachRolePolicy.

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient([  
    'profile' => 'default',
];
```
'region' => 'us-west-2',
'version' => '2010-05-08'
]);

$roleName = 'AmazonCSM';

$description = 'An Instance role that has permission for Amazon EC2 Systems Manager and SDK Metric Monitoring.';

$AmazonCSMPolicy = '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["sdkmetrics-beta:*"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["ssm:GetParameter"],
            "Resource": "arn:aws:ssm:*:*:parameter/AmazonCSM*"
        }
    ]
};

$rolePolicy = '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
};
	ry {
    $iamPolicy = $client->createPolicy([ 'PolicyName' => $roleName . 'policy', 'PolicyDocument' => $AmazonCSMPolicy ]);
    if ($iamPolicy['@metadata']['statusCode'] == 200) {
        $policyArn = $iamPolicy['Policy']['Arn'];
        echo('<p> Your IAM Policy has been created. Arn - '); echo($policyArn);
        echo('<p>');
        $role = $client->createRole([ 'RoleName' => $roleName, 'Description' => $description, 'AssumeRolePolicyDocument' => $rolePolicy, ]);
        echo('<p> Your IAM User Role has been created. Arn: '); echo($role['Role']['Arn']);
        echo('<p>');
        if ($role['@metadata']['statusCode'] == 200) {
            $result = $client->attachRolePolicy([ 'PolicyArn' => $policyArn, 'RoleName' => $roleName, ]);
        };
    };

}
Set Up Access Permissions by Using the IAM Console

Alternatively, you can use the IAM console to create a role.

1. Go to the IAM console, and create a role to use Amazon EC2.
2. In the navigation pane, choose Roles.
3. Choose Create Role.
4. Choose AWS Service, and then choose EC2.
5. Choose Next: Permissions.
6. Under Attach permissions policies, choose create policy.
7. For Service, choose Systems Manager. For Actions, expand Read, and choose GetParameters. For resources, specify your CloudWatch agent.
8. Add additional permission.
9. Select Choose a service, and then Enter service manually. For Service, enter sdkmetrics. Select all sdkmetrics actions and all resources, and then choose Review Policy.
10. Name the Role AmazonSDKMetrics, and add a description.
11. Choose Create Role.

Set Up SDK Metrics for the AWS SDK for PHP Version 3

The following steps demonstrate how to set up SDK Metrics for the AWS SDK for PHP. These steps pertain to an Amazon EC2 instance running Amazon Linux for a client application that is using the AWS SDK for PHP. SDK Metrics is also available for your production environments if you enable it while configuring the AWS SDK for PHP.

To use SDK Metrics, run the latest version of the CloudWatch agent. Learn how to Configure the CloudWatch Agent for SDK Metrics in the Amazon CloudWatch User Guide.

For more details about IAM Permissions for SDK Metrics, check out the IAM Permissions for SDK Metrics for AWS SDK for PHP (p. 87) article.

To set up SDK Metrics with the AWS SDK for PHP, follow these instructions:
1. Create an application with an AWS SDK for PHP client to use an AWS service.
2. Host your project on an Amazon EC2 instance or in your local environment.
3. Install and use the latest version of the AWS SDK for PHP.
4. Install and configure a CloudWatch agent on an EC2 instance or in your local environment.
5. Authorize SDK Metrics to collect and send metrics.
6. Enable SDK Metrics for the AWS SDK for PHP (p. 90).

For more information, see the following:

- Update a CloudWatch Agent (p. 92)
- Disable SDK Metrics (p. 93)

### Enable SDK Metrics for the AWS SDK for PHP

By default, SDK Metrics is turned off, host is set to ‘127.0.0.1’ and the port is set to 31000. The following are the default parameters.

```php
//default values
[
    'enabled' => false,
    'host' => '127.0.0.1',
    'port' => 31000,
]
```

Enabling SDK Metrics is independent of configuring your credentials to use an AWS service.

You can enable SDK Metrics by passing in a client configuration option, setting environment variables, or by using the AWS Shared config file.

#### Order of Precedence

The order of precedence is as follows (1 overrides 2-3, etc.):

1. Client configuration option
2. Environment variables
3. AWS Shared config file

#### Option 1: Client Configuration Option

You can pass in a `csm` option to your client constructor to enable and set the configuration options. This option can be an associative array, an instance of `\Aws\ClientSideMonitoring\ConfigurationInterface`, a callable that provides an instance of `\Aws\ClientSideMonitoring\ConfigurationInterface`, or the boolean value of `false`.

**Associative array:**

```php
$client = new \Aws\S3\S3Client([  
    'region' => 'your-region',
    'version' => 'latest',
    'csm' => [
        'enabled' => true,
        'host' => 'my.host',
        'port' => 1234,
        'client_id' => 'My Application'
    ]
]);
```
Instance of `Aws\ClientSideMonitoring\ConfigurationInterface`:

```php
$client = new \Aws\S3\S3Client(
    ['region' => 'your-region',
     'version' => 'latest',
     'csm' => new \Aws\ClientSideMonitoring\ConfigurationInterface
     (true,
      'my.host',
      1234,
      'My Application'
     )
);
```

Callable:

```php
$client = new \Aws\S3\S3Client(
    ['region' => 'your-region',
     'version' => 'latest',
     'csm' => function() {
       return new \Aws\ClientSideMonitoring\ConfigurationInterface
       (true,
        '127.0.0.1',
        1234,
        'My Application'
       );
     }
);
```

Boolean `false`:

```php
$client = new \Aws\S3\S3Client(
    ['region' => 'your-region',
     'version' => 'latest',
     'csm' => false
);
```

Option 2: Set Environment Variables

The SDK first checks the profile specified in the environment variable under AWS_PROFILE to determine if SDK Metrics is enabled.

To turn on SDK Metrics, add the following to your environment variables.

```bash
export AWS_CSM_ENABLED=true
```

Other configuration settings (p. 92) are available. For more information about using shared files, see Using Credentials from Environment Variables (p. 44).

**Note**

Enabling SDK Metrics does not configure your credentials to use an AWS service. To do that, see Credentials for the AWS SDK for PHP Version 3 (p. 42).

Option 3: AWS Shared Config File

If no CSM configuration is found in the environment variables, the SDK looks for your default AWS profile field in a shared configuration file. Typically this is in `~/.aws/config`, but you can change which file is used by setting the AWS_CONFIG_FILE environment variable. If AWS_DEFAULT_PROFILE is set
to something other than default, update that profile. To enable SDK Metrics, add `csm_enabled` to the shared config file.

```ini
[default]
csm_enabled = true

[profile aws_csm]
csm_enabled = true
```

Other configuration settings (p. 92) are available. For more information about using AWS Shared files, see Using the AWS Credentials File and Credential Profiles (p. 44).

**Note**
Enabling SDK Metrics does not configure your credentials to use an AWS service. To do that, see Credentials for the AWS SDK for PHP Version 3 (p. 42).

### Update a CloudWatch Agent

To make changes to the host or port ID, you need to set the values and then restart any AWS jobs that are currently active.

**Option 1: Client Configuration Option**

See Option 1 above under “Enable SDK Metrics for the AWS SDK for PHP” for examples of how to set the client configuration `csm` option to modify CSM settings.

**Option 2: Set Environment Variables**

Most AWS services use the default port. But if the service you want SDK Metrics to monitor uses a unique port, add `AWS_CSM_PORT=[port_number]`, to the host's environment variables. Additionally, a different host can be specified using the `AWS_CSM_HOST` environment variable.

```bash
export AWS_CSM_ENABLED=true
export AWS_CSM_PORT=1234
export AWS_CSM_HOST=192.168.0.1
```

**Option 3: AWS Shared Config File**

Most services use the default port. But if your service requires a unique port ID, add `csm_port = [port_number]` to `~/.aws/config` (or the file specified by the environment variable `AWS_CONFIG_FILE`). A non-default host can be configured using `csm_host`.

```ini
[default]
csm_enabled = false
csm_host = 123.4.5.6
csm_port = 1234

[profile aws_csm]
csm_enabled = false
csm_host = 123.4.5.6
csm_port = 1234
```

### Restart SDK Metrics

To restart a job, run the following commands.

```bash
amazon-cloudwatch-agent-ctl -a stop;
```
Disable SDK Metrics

To turn off SDK Metrics, set `csm_enabled` to `false` in your environment variables, or in your AWS Shared config file located at `~/.aws/config` (or the file specified by the environment variable `AWS_CONFIG_FILE`). Then restart your CloudWatch agent so that the changes can take effect.

Set `csm_enabled` to `false`

**Note**

Note the order of precedence listed above. For example, if SDK Metrics is enabled in the environment variables but disabled in the config file, the SDK Metrics remains enabled.

**Option 1: Client configuration**

```php
$client = new \Aws\S3\S3Client([  
    'region' => 'your-region',  
    'version' => 'latest',  
    'csm' => false  
]);
```

**Option 2: Environment Variables**

```bash
export AWS_CSM_ENABLED=false
```

**Option 3: AWS Shared Config File**

```ini
[default]
csm_enabled = false
[profile aws_csm]
csm_enabled = false
```

Stop SDK Metrics and Restart CloudWatch agent

To disable SDK Metrics, use the following command.

```bash
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a stop && echo "Done"
```

If you are using other CloudWatch features, restart CloudWatch with the following command.

```bash
amazon-cloudwatch-agent-ctl -a start;
```

Definitions for SDK Metrics

You can use the following descriptions of SDK Metrics to interpret your results. In general, these metrics are available for review with your Technical Account Manager during regular business reviews. AWS Support resources and your Technical Account Manager should have access to SDK Metrics data to help you resolve cases, but if you discover data that is confusing or unexpected, but doesn't seem to be negatively impacting your applications' performance, it is best to review that data during scheduled business reviews.
<table>
<thead>
<tr>
<th>Metric:</th>
<th>CallCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Total number of successful or failed API calls from your code to AWS services</td>
</tr>
<tr>
<td>How to use it</td>
<td>Use it as a baseline to correlate with other metrics like errors or throttling.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric:</th>
<th>ClientErrorCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Number of API calls that fail with client errors (4xx HTTP response codes). Examples: Throttling, Access denied, S3 bucket does not exist, and invalid parameter value.</td>
</tr>
<tr>
<td>How to use it</td>
<td>Except in certain cases related to throttling (ex. when throttling occurs due to a limit that needs to be increased) this metric can indicate something in your application that needs to be fixed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric:</th>
<th>ConnectionErrorCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Number of API calls that fail because of errors connecting to the service. These can be caused by network issues between the customer application and AWS services including load balancers, DNS failures, transit providers. In some cases, AWS issues may result in this error.</td>
</tr>
<tr>
<td>How to use it</td>
<td>Use this metric to determine whether issues are specific to your application or are caused by your infrastructure and/or network. High ConnectionErrorCount could also indicate short timeout values for API calls.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric:</th>
<th>ThrottleCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Number of API calls that fail due to throttling by AWS services.</td>
</tr>
<tr>
<td>How to use it</td>
<td>Use this metric to assess if your application has reached throttle limits, as well as to determine the cause of retries and application latency. Consider distributing calls over a window instead of batching your calls.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric:</th>
<th>ServerErrorCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Number of API calls that fail due to server errors (5xx HTTP response codes) from AWS Services. These are typically caused by AWS services.</td>
</tr>
</tbody>
</table>

94
### Metric: ServerErrorCount

**How to use it**
Determine cause of SDK retries or latency. This metric will not always indicate that AWS services are at fault, as some AWS teams classify latency as an HTTP 503 response.

---

### Metric: EndToEndLatency

**Definition**
Total time for your application to make a call using the AWS SDK, inclusive of retries. In other words, regardless of whether it is successful after several attempts, or as soon as a call fails due to an unretriable error.

**How to use it**
Determine how AWS API calls contribute to your application's overall latency. Higher than expected latency may be caused by issues with network, firewall, or other configuration settings, or by latency that occurs as a result of SDK retries.
Using AWS Services with AWS SDK for PHP Version 3

Some of the AWS services supported by the AWS SDK for PHP Version 3 include additional features beyond executing operations on an API. This guide documents these higher-level features per service.

Topics
- Using AWS Cloud9 with the AWS SDK for PHP Version 3 (p. 96)
- Using the DynamoDB Session Handler with AWS SDK for PHP Version 3 (p. 97)
- Amazon S3 Multi-Region Client with AWS SDK for PHP Version 3 (p. 102)
- Amazon S3 Stream Wrapper with AWS SDK for PHP Version 3 (p. 103)
- Amazon S3 Transfer Manager with AWS SDK for PHP Version 3 (p. 108)
- Amazon S3 Client-Side Encryption with the AWS SDK for PHP Version 3 (p. 111)

Using AWS Cloud9 with the AWS SDK for PHP Version 3

You can use AWS Cloud9 with the AWS SDK for PHP Version 3 to write, run, and debug your PHP code using just a browser. AWS Cloud9 includes tools such as a code editor, debugger, and terminal. Because the AWS Cloud9 IDE is cloud based, you can work on your projects from your office, home, or anywhere using an internet-connected machine. For general information about AWS Cloud9, see the AWS Cloud9 User Guide.

Follow these instructions to set up AWS Cloud9 with the AWS SDK for PHP:

- Step 1: Set up Your AWS Account to Use AWS Cloud9 (p. 96)
- Step 2: Set up Your AWS Cloud9 Development Environment (p. 96)
- Step 3: Set up the AWS SDK for PHP (p. 97)
- Step 4: Download Example Code (p. 97)
- Step 5: Run and Debug Example Code (p. 97)

Step 1: Set up Your AWS Account to Use AWS Cloud9

Start to use AWS Cloud9 by signing in to the AWS Cloud9 console as an AWS Identity and Access Management (IAM) entity (for example, an IAM user) in your AWS account who has access permissions for AWS Cloud9.

To set up an IAM entity in your AWS account to access AWS Cloud9, and to sign in to the AWS Cloud9 console, see Team Setup for AWS Cloud9 in the AWS Cloud9 User Guide.

Step 2: Set up Your AWS Cloud9 Development Environment

After you sign in to the AWS Cloud9 console, use the console to create an AWS Cloud9 development environment. After you create the environment, AWS Cloud9 opens the IDE for that environment.
Step 3: Set up the AWS SDK for PHP

After AWS Cloud9 opens the IDE for your development environment, use the IDE to set up the AWS SDK for PHP in your environment, as follows.

1. If the terminal isn't already open in the IDE, open it. On the menu bar in the IDE, choose Window, New Terminal.
2. Run the following commands to install the AWS SDK for PHP.

```
curl -sS https://getcomposer.org/installer | php
php composer.phar require aws/aws-sdk-php
```

If the IDE can't find PHP, run the following command to install it. (This command assumes you chose the option to Create a new instance for environment (EC2), earlier in this topic.)

```
sudo yum -y install php56
```

Step 4: Download Example Code

Use the terminal you opened in the previous step to download example code for the AWS SDK for PHP into the AWS Cloud9 development environment.

To do this, run the following command. This command downloads a copy of all of the code examples used in the official AWS SDK documentation into your environment's root directory.

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

To find code examples for the AWS SDK for PHP, use the Environment window to open the `ENVIRONMENT_NAME/aws-doc-sdk-examples/php/example_code` directory, where `ENVIRONMENT_NAME` is the name of your development environment.

To learn how to work with these and other code examples, see Code Examples (p. 116).

Step 5: Run and Debug Example Code

To run code in your AWS Cloud9 development environment, see Run Your Code in the AWS Cloud9 User Guide.

To debug code, see Debug Your Code in the AWS Cloud9 User Guide.

Using the DynamoDB Session Handler with AWS SDK for PHP Version 3

The DynamoDB Session Handler is a custom session handler for PHP that enables developers to use Amazon DynamoDB as a session store. Using DynamoDB for session storage alleviates issues that occur
with session handling in a distributed web application by moving sessions off of the local file system and into a shared location. DynamoDB is fast, scalable, easy to set up, and handles replication of your data automatically.

The DynamoDB Session Handler uses the `session_set_save_handler()` function to hook DynamoDB operations into PHP's native session functions to allow for a true drop in replacement. This includes support for features such as session locking and garbage collection, which are a part of PHP's default session handler.

For more information about the DynamoDB service, see the Amazon DynamoDB homepage.

## Basic Usage

### Step 1: Register the Handler

First, instantiate and register the session handler.

```php
use Aws\DynamoDb\SessionHandler;

$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name' => 'sessions'
]);

$sessionHandler->register();
```

### Step 2. Create a Table to Store Your Sessions

Before you can actually use the session handler, you need to create a table in which to store the sessions. You can do this ahead of time by using the AWS Console for Amazon DynamoDB, or by using the AWS SDK for PHP.

When creating this table use 'id' as the name of the primary key. Also it is recommended to setup a Time To Live attribute using the 'expires' attribute to benefit from automatic garbage collection of sessions.

### Step 3. Use PHP Sessions as You Normally Would

Once the session handler is registered and the table exists, you can write to and read from the session using the `$_SESSION` superglobal, just like you normally do with PHP's default session handler. The DynamoDB Session Handler encapsulates and abstracts the interactions with DynamoDB and enables you to simply use PHP's native session functions and interface.

```php
// Start the session
session_start();

// Alter the session data
$_SESSION['user.name'] = 'jeremy';
$_SESSION['user.role'] = 'admin';

// Close the session (optional, but recommended)
session_write_close();
```

## Configuration

You can configure the behavior of the session handler using the following options. All options are optional, but be sure to understand what the defaults are.
table_name

The name of the DynamoDB table in which to store the sessions. This defaults to 'sessions'.

hash_key

The name of the hash key in the DynamoDB sessions table. This defaults to 'id'.

data_attribute

The name of the attribute in the DynamoDB sessions table in which the session data is stored. This defaults to 'data'.

data_attribute_type

The type of the attribute in the DynamoDB sessions table in which the session data is stored. This defaults to 'string', but can optionally be set to 'binary'.

session_lifetime

The lifetime of an inactive session before it should be garbage collected. If it isn't provided, the actual lifetime value that will be used is \ini_get('session.gc_maxlifetime').

session_lifetime_attribute

The name of the attribute in the DynamoDB sessions table in which the session expiration time is stored. This defaults to 'expires'.

consistent_read

Whether the session handler should use consistent reads for the GetItem operation. The default is true.

locking

Whether to use session locking. The default is false.

batch_config

Configuration used to batch deletes during garbage collection. These options are passed directly into DynamoDB WriteRequestBatch objects. Manually trigger garbage collection via SessionHandler::garbageCollect().

max_lock_wait_time

Maximum time (in seconds) that the session handler should wait to acquire a lock before giving up. The default to is 10 and is only used with session locking.

min_lock_retry_microtime

Minimum time (in microseconds) that the session handler should wait between attempts to acquire a lock. The default is 10000 and is only used with session locking.

max_lock_retry_microtime

Maximum time (in microseconds) that the session handler should wait between attempts to acquire a lock. The default is 50000 and is only used with session locking.

To configure the Session Handler, specify the configuration options when you instantiate the handler. The following code is an example with all of the configuration options specified.

```php
$sessionHandler = SessionHandler::fromClient($dynamodb, [
    'table_name' => 'sessions',
    'hash_key'   => 'id',
]);
```
Pricing

Aside from data storage and data transfer fees, the costs associated with using DynamoDB are calculated based on the provisioned throughput capacity of your table (see the Amazon DynamoDB pricing details). Throughput is measured in units of write capacity and read capacity. The Amazon DynamoDB homepage says:

A unit of read capacity represents one strongly consistent read per second (or two eventually consistent reads per second) for items as large as 4 KB. A unit of write capacity represents one write per second for items as large as 1 KB.

Ultimately, the throughput and the costs required for your sessions table will correlate with your expected traffic and session size. The following table explains the amount of read and write operations that are performed on your DynamoDB table for each of the session functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read via <code>session_start()</code></td>
<td>• 1 read operation (only 0.5 if <code>consistent_read</code> is false).</td>
</tr>
<tr>
<td></td>
<td>• (Conditional) 1 write operation to delete the session if it is expired.</td>
</tr>
<tr>
<td>Read via <code>session_start()</code> (Using session locking)</td>
<td>• A minimum of 1 write operation.</td>
</tr>
<tr>
<td></td>
<td>• (Conditional) Additional write operations for each attempt at acquiring a lock on the session. Based on configured lock wait time and retry options.</td>
</tr>
<tr>
<td></td>
<td>• (Conditional) 1 write operation to delete the session if it is expired.</td>
</tr>
<tr>
<td>Write via <code>session_write_close()</code></td>
<td>• 1 write operation.</td>
</tr>
<tr>
<td>Delete via <code>session_destroy()</code></td>
<td>• 1 write operation.</td>
</tr>
<tr>
<td>Garbage Collection</td>
<td>• 0.5 read operations per 4 KB of data in the table to scan for expired sessions.</td>
</tr>
<tr>
<td></td>
<td>• 1 write operation per expired item to delete it.</td>
</tr>
</tbody>
</table>

Session Locking

The DynamoDB Session Handler supports pessimistic session locking to mimic the behavior of PHP’s default session handler. By default, the DynamoDB Session Handler has this feature turned off because it can become a performance bottleneck and drive up costs, especially when an application accesses the session when using Ajax requests or iframes. Carefully consider whether your application requires session locking before enabling it.
To enable session locking, set the 'locking' option to true when you instantiate the SessionHandler.

```php
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name' => 'sessions',
    'locking' => true,
]);
```

## Garbage Collection

Setup a TTL attribute in your DynamoDB table, using the attribute ‘expires’. This will automatically garbage collect your sessions and avoid the need to garbage collect them yourself.

Alternatively, the DynamoDB Session Handler supports session garbage collection by using a series of `Scan` and `BatchWriteItem` operations. Due to the nature of how the `Scan` operation works, and to find all of the expired sessions and delete them, the garbage collection process can require a lot of provisioned throughput.

For this reason, we do not support automated garbage collection. A better practice is to schedule the garbage collection to occur during an off-peak time when a burst of consumed throughput will not disrupt the rest of the application. For example, you could have a nightly cron job trigger a script to run the garbage collection. This script would need to do something like the following.

```php
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name' => 'sessions',
    'batch_config' => [
        'batch_size' => 25,
        'before' => function ($command) {
            echo "About to delete a batch of expired sessions.\n";
        }
    ]
]);

$sessionHandler->garbageCollect();
```

You can also use the 'before' option within 'batch_config' to introduce delays on the `BatchWriteItem` operations that are performed by the garbage collection process. This will increase the amount of time it takes the garbage collection to complete, but it can help you spread out the requests made by the DynamoDB Session Handler to help you stay close to or within your provisioned throughput capacity during garbage collection.

```php
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name' => 'sessions',
    'batch_config' => [
        'batch_size' => 25,
        'before' => function ($command) {
            $command['@http']['delay'] = 5000;
        }
    ]
]);

$sessionHandler->garbageCollect();
```

## Best Practices

1. Create your sessions table in an AWS Region that is geographically closest to or in the same Region as your application servers. This ensures the lowest latency between your application and DynamoDB database.
2. Choose the provisioned throughput capacity of your sessions table carefully. Take into account the expected traffic to your application and the expected size of your sessions. Alternatively use the 'On Demand' Read/Write capacity mode for your table.

3. Monitor your consumed throughput through the AWS Management Console or with Amazon CloudWatch, and adjust your throughput settings as needed to meet the demands of your application.

4. Keep the size of your sessions small (ideally less than 1 KB). Small sessions perform better and require less provisioned throughput capacity.

5. Do not use session locking unless your application requires it.

6. Instead of using PHP's built-in session garbage collection triggers, schedule your garbage collection via a cron job, or another scheduling mechanism, to run during off-peak hours. Use the 'batch_config' option to your advantage.

### Required IAM Permissions

To use the DynamoDB SessionHandler, your configured credentials (p. 42) must have permission to use the DynamoDB table that you created in a previous step (p. 98). The following IAM policy contains the minimum permissions that you need. To use this policy, replace the Resource value with the Amazon Resource Name (ARN) of the table that you created previously. For more information about creating and attaching IAM policies, see Managing IAM Policies in the IAM User Guide.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:dynamodb:<region>:<account-id>:table/<table-name>"
        }
    ]
}
```

### Amazon S3 Multi-Region Client with AWS SDK for PHP Version 3

The AWS SDK for PHP Version 3 provides a generic multi-region client that can be used with any service. This enables users to specify which AWS Region to send a command to by providing an `@region` input parameter to any command. In addition, the SDK provides a multi-region client for Amazon S3 that responds intelligently to specific Amazon S3 errors and reroutes commands accordingly. This enables users to use the same client to talk to multiple Regions. This is a particularly useful feature for users of the Amazon S3 Stream Wrapper with AWS SDK for PHP Version 3 (p. 103), whose buckets reside in multiple Regions.

### Basic Usage

The basic usage pattern of an Amazon S3 client is the same whether using a standard S3 client or its multi-region counterpart. The only usage difference at the command level is that an AWS Region can be specified using the `@region` input parameter.
// Create a multi-region S3 client
$s3Client = (new \Aws\Sdk)->createMultiRegionS3(['version' => 'latest']);

// You can also use the client constructor
$s3Client = new \Aws\S3\S3MultiRegionClient([  'version' => 'latest',  // Any Region specified while creating the client will be used as the  // default Region  'region' => 'us-west-2',]);

// Get the contents of a bucket
$objects = $s3Client->listObjects(['Bucket' => $bucketName]);

// If you would like to specify the Region to which to send a command, do so  // by providing an @region parameter
$objects = $s3Client->listObjects([  'Bucket' => $bucketName,  '@region' => 'eu-west-1',]);

Important
When using the multi-region Amazon S3 client, you will not encounter any permanent redirect exceptions. A standard Amazon S3 client will throw an instance of Aws\S3\Exception\PermanentRedirectException when a command is sent to the wrong Region. A multi-region client will instead redispacth the command to the correct Region.

Bucket Region Cache
Amazon S3 multi-region clients maintain an internal cache of the AWS Regions in which given buckets reside. By default, each client has its own in-memory cache. To share a cache between clients or processes, supply an instance of Aws\CacheInterface as the bucket_region_cache option to your multi-region client.

use Aws\DoctrineCacheAdapter;
use Aws\Sdk;
use Doctrine\Common\Cache\ApcuCache;

$-sdk = new Aws\Sdk([  'version' => 'latest',  'region' => 'us-west-2',  'S3' => [  'bucket_region_cache' => new DoctrineCacheAdapter(new ApcuCache),],]);

Amazon S3 Stream Wrapper with AWS SDK for PHP Version 3
The Amazon S3 stream wrapper enables you to store and retrieve data from Amazon S3 using built-in PHP functions, such as file_get_contents, fopen, copy, rename, unlink, mkdir, and rmdir.

You need to register the Amazon S3 stream wrapper to use it.

$client = new Aws\S3\S3Client([/** options **/]);
// Register the stream wrapper from an S3Client object
$client->registerStreamWrapper();

This enables you to access buckets and objects stored in Amazon S3 using the s3:// protocol. The Amazon S3 stream wrapper accepts strings that contain a bucket name followed by a forward slash and an optional object key or prefix: s3://<bucket>[/<key-or-prefix>].

**Note**  
The stream wrapper is designed for working with objects and buckets on which you have at least read permission. This means that your user should have permission to execute **ListBucket** on any buckets and **GetObject** on any object with which the user needs to interact. For use cases where you don’t have this permission level, we recommended that you use Amazon S3 client operations directly.

## Downloading Data

You can grab the contents of an object by using `file_get_contents`. However, be careful with this function; it loads the entire contents of the object into memory.

```php
// Download the body of the "key" object in the "bucket" bucket
$data = file_get_contents('s3://bucket/key');
```

Use `fopen()` when working with larger files or if you need to stream data from Amazon S3.

```php
// Open a stream in read-only mode
if ($stream = fopen('s3://bucket/key', 'r')) {
    // While the stream is still open
    while (!feof($stream)) {
        // Read 1,024 bytes from the stream
        echo fread($stream, 1024);
    }
    // Be sure to close the stream resource when you're done with it
    fclose($stream);
}
```

**Note**  
File write errors are only returned when a call to `fflush` is made. These errors are not returned when an unflushed `fclose` is called. The return value for `fclose` will be `true` if it closes the stream, regardless of any errors in response to its internal `fflush`. These errors are also not returned when calling `file_put_contents` because of how PHP implements it.

## Opening Seekable Streams

Streams opened in "r" mode only allow data to be read from the stream, and are not seekable by default. This is so that data can be downloaded from Amazon S3 in a truly streaming manner, where previously read bytes do not need to be buffered into memory. If you need a stream to be seekable, you can pass `seekable` into the stream context options of a function.

```php
$context = stream_context_create([    's3' => ['seekable' => true]]);
if ($stream = fopen('s3://bucket/key', 'r', false, $context)) {
    // Read bytes from the stream
    fread($stream, 1024);
}
```
Opening seekable streams enables you to seek bytes that were previously read. You can't skip ahead to
bytes that have not yet been read from the remote server. To allow previously read data to recalled, data
is buffered in a PHP temp stream using a stream decorator. When the amount of cached data exceeds 2
MB, the data in the temp stream transfers from memory to disk. Keep this in mind when downloading
large files from Amazon S3 using the seekable stream context setting.

Uploading Data

You can upload data to Amazon S3 using `file_put_contents()`.

```php
file_put_contents('s3://bucket/key', 'Hello!');
```

You can upload larger files by streaming data using `fopen()` and a "w", "x", or "a" stream access mode.
The Amazon S3 stream wrapper does not support simultaneous read and write streams (e.g. "r+", "w+",
etc). This is because the HTTP protocol doesn't allow simultaneous reading and writing.

```php
$stream = fopen('s3://bucket/key', 'w');
fwrite($stream, 'Hello!');
fclose($stream);
```

**Note**

Amazon S3 requires a Content-Length header to be specified before the payload of a request is
sent. Therefore, the data to be uploaded in a `PutObject` operation is internally buffered using a
PHP temp stream until the stream is flushed or closed.

**Note**

File write errors are returned only when a call to `fflush` is made. These errors are not returned
when an unflushed `fclose` is called. The return value for `fclose` will be true if it closes the
stream, regardless of any errors in response to its internal `fflush`. These errors are also not
returned when calling `file_put_contents` because of how PHP implements it.

**fopen Modes**

PHP's `fopen()` function requires that you specify a `mode` option. The mode option specifies whether data
can be read or written to a stream, and whether the file must exist when opening a stream. The Amazon
S3 stream wrapper supports the following modes.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>A read-only stream where the file must already exist.</td>
</tr>
<tr>
<td>w</td>
<td>A write-only stream. If the file already exists, it is overwritten.</td>
</tr>
<tr>
<td>a</td>
<td>A write-only stream. If the file already exists, it is downloaded to a temporary stream and any writes to the stream is appended to any previously uploaded data.</td>
</tr>
</tbody>
</table>
Other Object Functions

Stream wrappers allow many different built-in PHP functions to work with a custom system such as Amazon S3. Here are some of the functions that the Amazon S3 stream wrapper enables you to perform with objects stored in Amazon S3.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unlink()</td>
<td>Delete an object from a bucket.</td>
</tr>
<tr>
<td>filesize()</td>
<td>Get the size of an object.</td>
</tr>
<tr>
<td>is_file()</td>
<td>Checks if a URL is a file.</td>
</tr>
<tr>
<td>file_exists()</td>
<td>Checks if an object exists.</td>
</tr>
<tr>
<td>filetype()</td>
<td>Checks if a URL maps to a file or bucket (dir).</td>
</tr>
<tr>
<td>file()</td>
<td>Load the contents of an object in an array of lines. You can pass in any options available to the GetObject operation to modify how the file is downloaded.</td>
</tr>
<tr>
<td>filemtime()</td>
<td>Get the last modified date of an object.</td>
</tr>
<tr>
<td>rename()</td>
<td>Rename an object by copying the object then deleting the original. You can pass in options</td>
</tr>
</tbody>
</table>
Working with Buckets

You can modify and browse Amazon S3 buckets similarly to how PHP allows the modification and traversal of directories on your file system.

Here's an example of creating a bucket.

```php
mkdir('s3://bucket');
```

You can pass in stream context options to the `mkdir()` method to modify how the bucket is created using the parameters available to the `CreateBucket` operation.

```php
// Create a bucket in the EU (Ireland) Region
mkdir('s3://bucket', stream_context_create(['s3' => ['LocationConstraint' => 'eu-west-1']]));
```

You can delete buckets using the `rmdir()` function.

```php
// Delete a bucket
rmdir('s3://bucket');
```

**Note**

A bucket can only be deleted if it is empty.

Listing the Contents of a Bucket

You can use the `opendir()`, `readdir()`, `rewinddir()`, and `closedir()` PHP functions with the Amazon S3 stream wrapper to traverse the contents of a bucket. You can pass in parameters available to the `ListObjects` operation as custom stream context options to the `opendir()` function to modify how objects are listed.

```php
$dir = "s3://bucket/";
if (is_dir($dir) && ($dh = opendir($dir))) {
    while (($file = readdir($dh)) !== false) {
        echo "filename: \$file \n";
    }
    closedir($dh);
}
```

You can recursively list each object and prefix in a bucket using PHP’s `RecursiveDirectoryIterator`.

```php
$dir = 's3://bucket';
$iterator = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($dir));
foreach ($iterator as $file) {
    echo $file->getType() . ': ' . $file . "\n";
}

Another way to list the contents of a bucket recursively that incurs fewer HTTP requests is to use the \nAws\recursive_dir_iterator($path, $context = null) function.

<?php
require 'vendor/autoload.php';

$iter = Aws\recursive_dir_iterator('s3://bucket/key');
foreach ($iter as $filename) {
    echo $filename . "\n";
}

Stream Context Options

You can customize the client used by the stream wrapper, or the cache used to cache previously loaded information about buckets and keys, by passing in custom stream context options.

The stream wrapper supports the following stream context options on every operation.

client

The Aws\AwsClientInterface object to use to execute commands.

cache

An instance of Aws\CacheInterface to use to cache previously obtained file stats. By default, the stream wrapper uses an in-memory LRU cache.

Amazon S3 Transfer Manager with AWS SDK for PHP Version 3

The Amazon S3 transfer manager in the AWS SDK for PHP is used to upload entire directories to an Amazon S3 bucket and download entire buckets to a local directory.

Uploading a Local Directory to Amazon S3

The Aws\S3\Transfer object is used to perform transfers. The following example shows how to recursively upload a local directory of files to an Amazon S3 bucket.

// Create an S3 client
$client = new \Aws\S3\S3Client([  
    'region' => 'us-west-2',
    'version' => '2006-03-01',
]);

// Where the files will be source from
$source = '/path/to/source/files';

// Where the files will be transferred to
$dest = 's3://bucket';

// Create a transfer object
$manager = new \Aws\S3\Transfer($client, $source, $dest);
// Perform the transfer synchronously
$manager->transfer();

In this example, we created an Amazon S3 client, created a Transfer object, and performed transfer synchronously. Using the previous example demonstrates the bare minimum amount of code needed to perform a transfer. The transfer object can perform transfers asynchronously and has various configuration options you can use to customize the transfers.

You can upload the local files to a "subfolder" of an Amazon S3 bucket by providing a key prefix in the s3:// URI. The following example uploads the local files on disk to the bucket bucket and stores the files under the foo key prefix.

```php
$source = '/path/to/source/files';
$dest = 's3://bucket/foo';
$manager = new Aws\S3\Transfer($client, $source, $dest);
$manager->transfer();
```

**Downloading an Amazon S3 Bucket**

You can recursively download an Amazon S3 bucket to a local directory on disk by specifying the $source argument as an Amazon S3 URI (e.g., s3://bucket) and the $dest argument as the path to a local directory.

```php
// Where the files will be sourced from
$source = 's3://bucket';

// Where the files will be transferred to
$dest = '/path/to/destination/dir';

$manager = new Aws\S3\Transfer($client, $source, $dest);
$manager->transfer();
```

**Note**
The SDK will automatically create any necessary directories when downloading the objects in the bucket.

You can include a key prefix in the Amazon S3 URI after the bucket to download only objects stored under a "pseudo-folder". The following example downloads only files stored under the "/foo" key prefix of the given bucket.

```php
$source = 's3://bucket/foo';
$dest = '/path/to/destination/dir';
$manager = new Aws\S3\Transfer($client, $source, $dest);
$manager->transfer();
```

**Configuration**

The Transfer object constructor accepts the following arguments.

* `$client` The Aws\ClientInterface object to use to perform the transfers.

* `$source (string|`\``Iterator``)` The source data being transferred. This can point to a local path on disk (e.g., /path/to/files) or an Amazon S3 bucket (e.g., s3://bucket). The s3:// URI may also contain a key prefix that can be used to only transfer objects under a common prefix.
If the $source argument is an Amazon S3 URI, the $dest argument must be a local directory (and vice versa).

In addition to providing a string value, you can also provide an \Iterator object that yields absolute file names. If you provide an iterator, you **must** provide a base_dir option in the $options associative array.

**$dest**

The destination where the files will be transferred. If the $source argument is a local path on disk, $dest must be an Amazon S3 bucket URI (e.g., s3://bucket). If the $source argument is an Amazon S3 bucket URI, the $dest argument must be a local path on disk.

**$options**

An associative array of **transfer options** (p. 110).

### Transfer Options

**base_dir (string)**

Base directory of the source, if $source is an iterator. If the $source option is not an array, then this option is ignored.

**before (callable)**

A callback to invoke before each transfer. The callback should have a function signature like function (Aws\Command $command) {...}. The provided command will be a GetObject, PutObject, CreateMultipartUpload, UploadPart, or CompleteMultipartUpload command.

**mup_threshold (int)**

Size in bytes in which a multipart upload should be used instead of PutObject. Defaults to 16777216 (16 MB).

**concurrency (int, default=5)**

Number of files to upload concurrently. The ideal concurrency value will vary based on the number of files being uploaded and the average size of each file. Generally, smaller files benefit from a higher concurrency while larger files do not.

**debug (bool)**

Set to true to print out debug information for transfers. Set to an fopen() resource to write to a specific stream instead of writing to STDOUT.

### Async Transfers

The Transfer object is an instance of GuzzleHttp\Promise\PromisorInterface. This means that the transfer can occur asynchronously and is initiated by calling the promise method of the object.

```php
$source = '/path/to/source/files';
$dest = 's3://bucket';
$manager = new \Aws\S3\Transfer($client, $source, $dest);

// Initiate the transfer and get a promise
$promise = $manager->promise();

// Do something when the transfer is complete using the then() method
$promise->then(function () {
  echo 'Done!';
});
```
The promise will be rejected if any of the files fail to transfer. You can handle the failed transfer asynchronously using the otherwise method of the promise. The otherwise function accepts a callback to invoke when an error occurs. The callback accepts the $reason for the rejection, which will typically be an instance of Aws\Exception\AwsException (although a value of any type can be delivered to the callback).

```php
$promise->otherwise(function ($reason) {
    echo 'Transfer failed: ';
    var_dump($reason);
});
```

Because the Transfer object returns a promise, these transfers can occur concurrently with other asynchronous promises.

### Customizing the Transfer Manager’s Commands

Custom options can be set on the operations executed by the transfer manager via a callback passed to its constructor.

```php
$uploader = new Transfer($s3Client, $source, $dest, [
    'before' => function ($command) {
        // Commands can vary for multipart uploads, so check which command
        // is being processed
        if (in_array($command->getName(), ['PutObject', 'CreateMultipartUpload'])) {
            // Set custom cache-control metadata
            $command['CacheControl'] = 'max-age=3600';
            // Apply a canned ACL
            $command['ACL'] = strpos($command['Key'], 'CONFIDENTIAL') === false ? 'public-read' : 'private';
        }
    },
]);
```

### Amazon S3 Client-Side Encryption with the AWS SDK for PHP Version 3

With client-side encryption, data is encrypted and decrypted directly in your environment. This means that this data is encrypted before it's transferred to Amazon S3, and you don't rely on an external service to handle encryption for you. For new implementations, we suggest the use of S3EncryptionClientV2 and S3EncryptionMultipartUploaderV2 over the deprecated S3EncryptionClient and S3EncryptionMultipartUploader. It is recommended that older implementations still using the deprecated versions attempt to migrate. S3EncryptionClientV2 maintains support for decrypting data that was encrypted using the legacy S3EncryptionClient.

The AWS SDK for PHP implements envelope encryption and uses OpenSSL for its encrypting and decrypting. The implementation is interoperable with other SDKs that match its feature support. It's also compatible with the SDK's promise-based asynchronous workflow (p. 63).

### Migration Guide

For those who are trying to migrate to from the deprecated clients to the new clients, there is a migration guide which can be found here.
Setup

To get started with client-side encryption, you need the following:

- An AWS KMS encryption key
- An S3 bucket

Before running any example code, configure your AWS credentials. See Credentials for the AWS SDK for PHP Version 3 (p. 42).

Encryption

Uploading an encrypted object in S3EncryptionClientV2 takes three additional parameters on top of the standard PutObject parameters:

- `@KmsEncryptionContext` is a key-value pair which can be used to add an extra layer of security to your encrypted object. The encryption client must pass in the same key, which it will automatically do on a get call. If no additional context is desired, pass in an empty array.
- `@CipherOptions` are additional configurations for the encryption including which cipher to use and keysize.
- `@MaterialsProvider` is a provider which handles generating a cipher key and initialization vector, as well as encrypting your cipher key.

```php
use Aws\S3\S3Client;
use Aws\S3\Crypto\S3EncryptionClientV2;
use Aws\Kms\KmsClient;
use Aws\Crypto\KmsMaterialsProviderV2;

// Let's construct our S3EncryptionClient using an S3Client
$encryptionClient = new S3EncryptionClientV2(
    new S3Client(
        [ 'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ]
);

$kmsKeyId = 'kms-key-id';
(materialsProvider = new KmsMaterialsProviderV2(
    new KmsClient(
        [ 'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ],
    $kmsKeyId
));

$bucket = 'the-bucket-name';
$key = 'the-file-name';
$cipherOptions = [
    'Cipher' => 'gcm',
    'KeySize' => 256,
    // Additional configuration options
];

$result = $encryptionClient->putObject(
    [ '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
];
```
Decryption

Downloading and decrypting an object has four additional parameters, two of which are required, on top of the standard GetObject parameters. The client will detect the basic cipher options for you.

- `@SecurityProfile`: If set to 'V2', only objects that are encrypted in V2-compatible format can be decrypted. Setting this parameter to 'V2_AND_LEGACY' also allows objects encrypted in V1-compatible format to be decrypted. To support migration, set `@SecurityProfile` to 'V2_AND_LEGACY'. Use 'V2' only for new application development.
- `@MaterialsProvider` is a provider which handles generating a cipher key and initialization vector, as well as encrypting your cipher key.
- `@KmsAllowDecryptWithAnyCmk`: (optional) Setting this parameter to true enables decryption without supplying a KMS key id to the constructor of the MaterialsProvider. The default value is false.
- `@CipherOptions` (optional) are additional configurations for the encryption including which cipher to use and keysize.

```php
$result = $encryptionClient->getObject(
    ['@KmsAllowDecryptWithAnyCmk' => true,
     '@SecurityProfile' => 'V2_AND_LEGACY',
     '@MaterialsProvider' => $materialsProvider,
     '@CipherOptions' => $cipherOptions,
     'Bucket' => $bucket,
     'Key' => $key,
    ]);```

Note

In addition to the Amazon S3 and AWS KMS-based service errors, you might receive thrown InvalidArgumentException objects if your '@CipherOptions' are not correctly configured.

Cipher Configuration

`Cipher` (string)

Cipher method that the encryption client uses while encrypting. Only 'gcm' is supported at this time.

Important

PHP is updated in version 7.1 to include the extra parameters necessary to encrypt and decrypt using OpenSSL for GCM encryption. For PHP versions 7.0 and earlier, a polyfill for
GCM support is provided and used by the encryption clients \texttt{S3EncryptionClientV2} and \texttt{S3EncryptionMultipartUploaderV2}. However, the performance for large inputs will be much slower using the polyfill than using the native implementation for PHP 7.1+, so upgrading older PHP version environments may be necessary to use them effectively.

\texttt{'KeySize'} (int)

The length of the content encryption key to generate for encrypting. Defaults to 256 bits. Valid configuration options are 256 and 128 bits.

\texttt{'Aad'} (string)

Optional `Additional authentication data` to include with your encrypted payload. This information is validated on decryption. \texttt{Aad} is available only when using the `gcm` cipher.

\textbf{Important}

Additional authentication data is not supported by all AWS SDKs and as such other SDKs may not be able to decrypt files encrypted using this parameter.

## Metadata Strategies

You also have the option of providing an instance of a class that implements the \texttt{Aws\Crypto\MetadataStrategyInterface}. This simple interface handles saving and loading the \texttt{Aws\Crypto\MetadataEnvelope} that contains your envelope encryption materials. The SDK provides two classes that implement this: \texttt{Aws\S3\Crypto\HeadersMetadataStrategy} and \texttt{Aws\S3\Crypto\InstructionFileMetadataStrategy}. \texttt{HeadersMetadataStrategy} is used by default.

```php
(strategy = new InstructionFileMetadataStrategy(
    $s3Client
));

$encryptionClient->putObject([  
    '@MaterialsProvider' => $materialsProvider,
    '@MetadataStrategy' => $strategy,
    '@KmsEncryptionContext' => [],
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

$result = $encryptionClient->getObject([  
    '@KmsAllowDecryptWithAnyCmk' => false,
    '@MaterialsProvider' => $materialsProvider,
    '@SecurityProfile' => 'V2',
    '@MetadataStrategy' => $strategy,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

\textbf{Class name constants for the HeadersMetadataStrategy and InstructionFileMetadataStrategy} can also be supplied by invoking `::class`.

```php
$result = $encryptionClient->putObject([  
    '@MaterialsProvider' => $materialsProvider,
    '@MetadataStrategy' => HeadersMetadataStrategy::class,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
];
```
'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

Note

If there is a failure after an instruction file is uploaded, it will not be automatically deleted.

### Multipart Uploads

Performing a multipart upload with client-side encryption is also possible. The `Aws\S3\Crypto\S3EncryptionMultipartUploaderV2` prepares the source stream for encryption before uploading. Creating one takes on a similar experience to using the `Aws\S3\MultipartUploader` and the `Aws\S3\Crypto\S3EncryptionClientV2`. The `S3EncryptionMultipartUploaderV2` can handle the same 'MetadataStrategy' option as the `S3EncryptionClientV2`, as well as all available 'CipherOptions' configurations.

```php
$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV2(
    new KmsClient(
        ['region' => 'us-east-1',
        'version' => 'latest',
        'profile' => 'default',
    ],
    $kmsKeyId
),

$bucket = 'the-bucket-name';
$key = 'the-upload-key';
$cipherOptions = [
    'Cipher' => 'gcm',
    'KeySize' => 256,
    // Additional configuration options
];

$multipartUploader = new S3EncryptionMultipartUploaderV2(
    new S3Client(
        ['region' => 'us-east-1',
        'version' => 'latest',
        'profile' => 'default',
    ],
    fopen('large-file-to-encrypt.txt', 'r'),
    [
        '@MaterialsProvider' => $materialsProvider,
        '@CipherOptions' => $cipherOptions,
        'bucket' => $bucket,
        'key' => $key,
    ]
); $multipartUploader->upload();

Note

In addition to the Amazon S3 and AWS KMS-based service errors, you might receive thrown `InvalidArgumentException` objects if your 'CipherOptions' are not correctly configured.
AWS SDK for PHP Version 3 Code Examples

Topics

- Credentials (p. 24)
- Amazon CloudFront Examples Using the AWS SDK for PHP Version 3 (p. 116)
- Signing Custom Amazon CloudSearch Domain Requests with AWS SDK for PHP Version 3 (p. 136)
- Amazon CloudWatch Examples Using the AWS SDK for PHP Version 3 (p. 137)
- Amazon EC2 Examples Using the AWS SDK for PHP Version 3 (p. 154)
- Signing an Amazon OpenSearch Service Search Request with AWS SDK for PHP Version 3 (p. 164)
- AWS Identity and Access Management Examples Using the AWS SDK for PHP Version 3 (p. 165)
- AWS Key Management Service Examples Using the AWS SDK for PHP Version 3 (p. 183)
- Amazon Kinesis Examples Using the AWS SDK for PHP Version 3 (p. 199)
- AWS Elemental MediaConvert Examples Using the AWS SDK for PHP Version 3 (p. 211)
- Amazon S3 Examples Using the AWS SDK for PHP Version 3 (p. 218)
- Managing Secrets Using the Secrets Manager API and the AWS SDK for PHP Version 3 (p. 239)
- Amazon SES Examples Using the AWS SDK for PHP Version 3 (p. 245)
- Amazon SNS Examples Using the AWS SDK for PHP Version 3 (p. 268)
- Amazon SQS Examples Using the AWS SDK for PHP Version 3 (p. 281)

The AWS SDK for PHP Version 3 includes code examples that demonstrate common Amazon Web Services scenarios that use the SDK.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Amazon CloudFront Examples Using the AWS SDK for PHP Version 3

Amazon CloudFront is an AWS web service that speeds up serving static and dynamic web content from your own web server or an AWS server, such as Amazon S3. CloudFront delivers content through
a worldwide network of data centers called edge locations. When a user requests content that you're distributing with CloudFront, they're routed to the edge location that provides the lowest latency. If the content isn't cached there already, CloudFront retrieves a copy from the origin server, serves it, and then caches it for future requests.

For more information about CloudFront, see the Amazon CloudFront Developer Guide.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Managing Amazon CloudFront Distributions Using the CloudFront API and the AWS SDK for PHP Version 3

Amazon CloudFront caches content in worldwide edge locations to speed up distribution of static and dynamic files that you store on your own server, or on an Amazon service like Amazon S3 and Amazon EC2. When users request content from your website, CloudFront serves it from the closest edge location, if the file is cached there. Otherwise CloudFront retrieves a copy of the file, serves it, and then caches it for the next request. Caching content at an edge location reduces the latency of similar user requests in that area.

For each CloudFront distribution that you create, you specify where the content is located and how to distribute it when users make requests. This topic focuses on distributions for static and dynamic files such as HTML, CSS, JSON, and image files. For information about using CloudFront with video on demand, see On-Demand and Live Streaming Video with CloudFront.

The following examples show how to:

- Create a distribution using CreateDistribution.
- Get a distribution using GetDistribution.
- List distributions using ListDistributions.
- Update distributions using UpdateDistributions.
- Disable distributions using DisableDistribution.
- Delete distributions using DeleteDistributions.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon CloudFront, see the Amazon CloudFront Developer Guide.

Create a CloudFront Distribution

Create a distribution from an Amazon S3 bucket. In the following example, optional parameters are commented out, but default values are displayed. To add customizations to your distribution, uncomment both the value and the parameter inside $distribution.

To create a CloudFront distribution, use the CreateDistribution operation.
Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function createS3Distribution($cloudFrontClient, $distribution)
{
    try {
        $result = $cloudFrontClient->createDistribution(
            'DistributionConfig' => $distribution
        );
        $message = '';
        if (isset($result['Distribution']['Id'])) {
            $message = 'Distribution created with the ID of ' . $result['Distribution']['Id'];
        }
        $message .= ' and an effective URI of ' . $result['@metadata']['effectiveUri'] . '.
        return $message;
    } catch (AwsException $e) {
        return 'Error: ' . $e['message'];
    }
}

function createsTheS3Distribution()
{
    $originName = 'my-unique-origin-name';
    $s3BucketURL = 'my-bucket-name.s3.amazonaws.com';
    $callerReference = 'my-unique-caller-reference';
    $comment = 'my-comment-about-this-distribution';
    $defaultCacheBehavior = [
        'AllowedMethods' => [
            'CachedMethods' => [
                'Items' => ['HEAD', 'GET'],
                'Quantity' => 2
            ],
            'Items' => ['HEAD', 'GET'],
            'Quantity' => 2
        ],
        'Compress' => false,
        'DefaultTTL' => 0,
        'FieldLevelEncryptionId' => '',
        'ForwardedValues' => [
            'Cookies' => [],
            'Forward' => 'none',
            'Headers' => [
                'Quantity' => 0
            ],
            'QueryString' => false,
            'QueryStringCacheKeys' => [
                'Quantity' => 0
            ],
            'LambdaFunctionAssociations' => [
                'Quantity' => 0
            ],
        ],
    ];
```
retrieve a cloudfront distribution

to retrieve the status and details of a specified cloudfront distribution, use the getdistribution operation.

import

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

sample code

```php
function getDistribution($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution([
            'Id' => $distributionId
        ]);  
    } catch (AwsException $e) {
        echo 'Error: ' . $e->getMessage();
    }
}
```

// uncomment the following line to run this code in an aws account.
// createsTheS3Distribution();

119
$message = '';  
if (isset($result['Distribution']['Status']))  
{
    $message = 'The status of the distribution with the ID of ' . 
    $result['Distribution']['Id'] . ' is currently ' . 
    $result['Distribution']['Status'];
}  

if (isset($result['@metadata']['effectiveUri']))  
{
    $message .= ', and the effective URI is ' . 
    $result['@metadata']['effectiveUri'] . '.';
}  
else {
    $message = 'Error: Could not get the specified distribution. ' . 
    'The distribution\'s status is not available.';
}

return $message;
}

function getsADistribution()  
{
    $distributionId = 'E1BTGP2EXAMPLE';
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient([  
        'profile' => 'default',  
        'version' => '2018-06-18',  
        'region' => 'us-east-1'
    ]);  
    echo getDistribution($cloudFrontClient, $distributionId);  
}

// Uncomment the following line to run this code in an AWS account.  
// getsADistribution();

### List CloudFront Distributions

Get a list of existing CloudFront distributions in the specified AWS Region from your current account using the **ListDistributions** operation.

**Imports**

```php
require 'vendor/autoload.php';

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
function listDistributions($cloudFrontClient)  
{
    try {
        $result = $cloudFrontClient->listDistributions([]);
        return $result;
```
function listTheDistributions()
{
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient(["profile" => 'default',
                                                             "version" => '2018-06-18',
                                                             "region" => 'us-east-2']);

    $distributions = listDistributions($cloudFrontClient);

    if (count($distributions) == 0)
    {
        echo 'Could not find any distributions.';
    } else {
        foreach ($distributions['DistributionList']['Items'] as $distribution)
        {
            echo 'The distribution with the ID of ' . $distribution['Id'] . ' has the status of ' . $distribution['Status'] . '.
        }
    }
}

// Uncomment the following line to run this code in an AWS account.
// listTheDistributions();

Update a CloudFront Distribution

Updating a CloudFront distribution is similar to creating a distribution. However, when you update a
distribution, more fields are required and all values must be included. To make changes to an existing
distribution, we recommend that you first retrieve the existing distribution, and update the values you
want to change in the $distribution array.

To update a specified CloudFront distribution, use the UpdateDistribution operation.

Imports

require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;

Sample Code

function updateDistribution($cloudFrontClient, $distributionId, $distributionConfig, $eTag)
{
    try {
        $result = $cloudFrontClient->updateDistribution(["DistributionConfig" => $distributionConfig,
                                                             "Id" => $distributionId,
                                                             "IfMatch" => $eTag]);

        return 'The distribution with the following effective URI has been updated: ' . $result["@metadata"]["effectiveUri"];
```php
function getDistributionConfig($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution(
            'Id' => $distributionId,
        );
        if (isset($result['Distribution']['DistributionConfig']))
        {
            return [
                'DistributionConfig' => $result['Distribution']['DistributionConfig'],
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        } else {
            return [
                'Error' => 'Error: Cannot find distribution configuration details.',
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        }
    } catch (AwsException $e) {
        return [
            'Error' => 'Error: ' . $e->getAwsErrorMessage()
        ];
    }
}

function getDistributionETag($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution(
            'Id' => $distributionId,
        );
        if (isset($result['ETag']))
        {
            return [
                'ETag' => $result['ETag'],
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        } else {
            return [
                'Error' => 'Error: Cannot find distribution ETag header value.',
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        }
    } catch (AwsException $e) {
        return [
            'Error' => 'Error: ' . $e->getAwsErrorMessage()
        ];
    }
}

function updateADistribution()
{
    // $distributionId = 'E1BTGP2EXAMPLE';
    $distributionId = 'E1X3BKQ569KEMH';
    $cloudFrontClient = new CloudFrontClient([}
'profile' => 'default',
'version' => '2018-06-18',
'region' => 'us-east-1' });

// To change a distribution, you must first get the distribution's
// ETag header value.
$eTag = getDistributionETag($cloudFrontClient, $distributionId);

if (array_key_exists('Error', $eTag)) {
    exit($eTag['Error']);
}

// To change a distribution, you must also first get information about
// the distribution's current configuration. Then you must use that
// information to build a new configuration.
$currentConfig = getDistributionConfig($cloudFrontClient, $distributionId);

if (array_key_exists('Error', $currentConfig)) {
    exit($currentConfig['Error']);
}

// To change a distribution's configuration, you can set the
// distribution's related configuration value as part of a change request,
// for example:
//   'Enabled' => true
// Some configuration values are required to be specified as part of a change
// request, even if you don't plan to change their values. For ones you
// don't want to change but are required to be specified, you can just reuse
// their current values, as follows.
$distributionConfig = [
    'CallerReference' => $currentConfig['DistributionConfig']['CallerReference'],
    'Comment' => $currentConfig['DistributionConfig']['Comment'],
    'DefaultCacheBehavior' => $currentConfig['DistributionConfig']['DefaultCacheBehavior'],
    'DefaultRootObject' => $currentConfig['DistributionConfig']['DefaultRootObject'],
    'Enabled' => $currentConfig['DistributionConfig']['Enabled'],
    'Origins' => $currentConfig['DistributionConfig']['Origins'],
    'Aliases' => $currentConfig['DistributionConfig']['Aliases'],
    'CustomErrorResponses' => $currentConfig['DistributionConfig']['CustomErrorResponses'],
    'HttpVersion' => $currentConfig['DistributionConfig']['HttpVersion'],
    'CacheBehaviors' => $currentConfig['DistributionConfig']['CacheBehaviors'],
    'Logging' => $currentConfig['DistributionConfig']['Logging'],
    'PriceClass' => $currentConfig['DistributionConfig']['PriceClass'],
    'Restrictions' => $currentConfig['DistributionConfig']['Restrictions'],
    'ViewerCertificate' => $currentConfig['DistributionConfig']['ViewerCertificate'],
    'WebACLId' => $currentConfig['DistributionConfig']['WebACLId']
];

echo updateDistribution($cloudFrontClient, $distributionId,
    $distributionConfig, $eTag['ETag']);
}

// Uncomment the following line to run this code in an AWS account.
// updateADistribution();

---

**Disable a CloudFront Distribution**

To deactivate or remove a distribution, change its status from deployed to disabled.

To disable the specified CloudFront distribution, use the DisableDistribution operation.

**Imports**
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;

Sample Code

```php
function disableDistribution($cloudFrontClient, $distributionId, $distributionConfig, $eTag)
{
    try {
        $result = $cloudFrontClient->updateDistribution(["DistributionConfig" => $distributionConfig,
            'Id' => $distributionId,
            'IfMatch' => $eTag
        ]);;
        return 'The distribution with the following effective URI has been disabled: '.
            $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function getDistributionConfig($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution(["Id" => $distributionId,
            ]);;
        if (isset($result['Distribution']['DistributionConfig']))
        { return [
            'DistributionConfig' => $result['Distribution']['DistributionConfig'],
            'effectiveUri' => $result['@metadata']['effectiveUri']
        ];
        } else {
            return [
                'Error' => 'Error: Cannot find distribution configuration details.',
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        }
    } catch (AwsException $e) {
        return [
            'Error' => 'Error: ' . $e->getAwsErrorMessage()];
    }
}

function getDistributionETag($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution(["Id" => $distributionId,
            ]);;
        if (isset($result['ETag']))
        { return [
            'ETag' => $result['ETag'],
            'effectiveUri' => $result['@metadata']['effectiveUri']
        ];
        }
    }
```
} else {
    return [
        'Error' => 'Error: Cannot find distribution ETag header value.',
        'effectiveUri' => $result['@metadata']['effectiveUri']
    ];
}
}

} catch (AwsException $e) {
    return [
        'Error' => 'Error: ' . $e->getAwsErrorMessage()
    ];
}
}

function disableADistribution() {
    $distributionId = 'E1BTGP2EXAMPLE';
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient(
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    );
    // To disable a distribution, you must first get the distribution's
    // ETag header value.
    $eTag = getDistributionETag($cloudFrontClient, $distributionId);
    if (array_key_exists('Error', $eTag)) {
        exit($eTag['Error']);
    }
    // To delete a distribution, you must also first get information about
    // the distribution's current configuration. Then you must use that
    // information to build a new configuration, including setting the new
    // configuration to disabled.
    $currentConfig = getDistributionConfig($cloudFrontClient, $distributionId);
    if (array_key_exists('Error', $currentConfig)) {
        exit($currentConfig['Error']);
    }
    $distributionConfig = [
        'CacheBehaviors' => $currentConfig['DistributionConfig']['CacheBehaviors'],
        'CallerReference' => $currentConfig['DistributionConfig']['CallerReference'],
        'Comment' => $currentConfig['DistributionConfig']['Comment'],
        'DefaultCacheBehavior' => $currentConfig['DistributionConfig'],'
        'DefaultRootObject' => $currentConfig['DistributionConfig']['DefaultRootObject'],
        'Enabled' => false,
        'Origins' => $currentConfig['DistributionConfig']['Origins'],
        'Aliases' => $currentConfig['DistributionConfig']['Aliases'],
        'CustomErrorResponses' => $currentConfig['DistributionConfig']['CustomErrorResponses'],
        'HttpVersion' => $currentConfig['DistributionConfig']['HttpVersion'],
        'Logging' => $currentConfig['DistributionConfig']['Logging'],
        'PriceClass' => $currentConfig['DistributionConfig']['PriceClass'],
        'Restrictions' => $currentConfig['DistributionConfig']['Restrictions'],
        'ViewerCertificate' => $currentConfig['DistributionConfig']['ViewerCertificate'],
        'WebACLId' => $currentConfig['DistributionConfig']['WebACLId']
    ];
    echo disableDistribution($cloudFrontClient, $distributionId,
        $distributionConfig, $eTag['ETag']);
}
Delete a CloudFront Distribution

Once a distribution is in a disabled status, you can delete the distribution.

To remove a specified CloudFront distribution, use the DeleteDistribution operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function deleteDistribution($cloudFrontClient, $distributionId, $eTag)
{
    try {
        $result = $cloudFrontClient->deleteDistribution(["Id" => $distributionId,
            "IfMatch" => $eTag");
        return 'The distribution at the following effective URI has been deleted: ' . $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function getDistributionETag($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution(['Id' => $distributionId,]);
        if (isset($result['ETag']))
        {
            return [
                'ETag' => $result['ETag'],
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        } else {
            return [
                'Error' => 'Error: Cannot find distribution ETag header value.',
                'effectiveUri' => $result['@metadata']['effectiveUri']
            ];
        }
    } catch (AwsException $e) {
        return ['Error' => 'Error: ' . $e->getAwsErrorMessage()];
    }
}

function deleteADistribution()
{
    $distributionId = 'E17G7YNEXAMPLE';
}
Managing Amazon CloudFront Invalidations Using the CloudFront API and the AWS SDK for PHP Version 3

Amazon CloudFront caches copies of static and dynamic files in worldwide edge locations. To remove or update a file on all edge locations, create an invalidation for each file or for a group of files.

Each calendar month, your first 1,000 invalidations are free. To learn more about removing content from a CloudFront edge location, see Invalidating Files.

The following examples show how to:

- Create a distribution invalidation using CreateInvalidation.
- Get a distribution invalidation using GetInvalidation.
- List distributions using ListInvalidations.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon CloudFront, see the Amazon CloudFront Developer Guide.

**Create a Distribution Invalidation**

Create a CloudFront distribution invalidation by specifying the path location for the files you need to remove. This example invalidates all files in the distribution, but you can identify specific files under Items.

To create a CloudFront distribution invalidation, use the CreateInvalidation operation.
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;

Sample Code

function createInvalidation($cloudFrontClient, $distributionId, $callerReference, $paths, $quantity)
{
    try {
        $result = $cloudFrontClient->createInvalidation([
            'DistributionId' => $distributionId,
            'InvalidationBatch' => [
                'CallerReference' => $callerReference,
                'Paths' => [
                    'Items' => $paths,
                    'Quantity' => $quantity,
                ],
            ],
        ]);
        $message = ''; 
        if (isset($result['Location']))
        {
            $message = 'The invalidation location is: ' . $result['Location'];
        }
        $message .= ' and the effective URI is ' . $result['@metadata']['effectiveUri']. '.';
        return $message;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function createTheInvalidation()
{
    $distributionId = 'E1WICG14DUW2AP'; // 'E17G7YNEXAMPLE';
    $callerReference = 'my-unique-value';
    $paths = ['//*'];
    $quantity = 1;

    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient([ 
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]); 
    echo createInvalidation($cloudFrontClient, $distributionId, $callerReference, $paths, $quantity);
}

// Uncomment the following line to run this code in an AWS account.
// createTheInvalidation();

Get a Distribution Invalidation

To retrieve the status and details about a CloudFront distribution invalidation, use the GetInvalidation operation.
Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function getInvalidation($cloudFrontClient, $distributionId, $invalidationId)
{
    try {
        $result = $cloudFrontClient->getInvalidation(['
            DistributionId' => $distributionId,
            'Id' => $invalidationId,
        ]); 
        $message = ''; 
        if (isset($result['Invalidation']['Status'])) {
            $message = 'The status for the invalidation with the ID of ' . $result['Invalidation']['Id'] . ' is ' . $result['Invalidation']['Status'];
        }
        if (isset($result['@metadata']['effectiveUri'])) {
            $message .= ', and the effective URI is ' . $result['@metadata']['effectiveUri'] . ' .';
        } else {
            $message = 'Error: Could not get information about the invalidation. The invalidation\'s status was not available.';
        }
        return $message;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function getsAnInvalidation()
{
    $distributionId = 'E1BTGP2EXAMPLE'; 
    $invalidationId = 'I1CDEZZEXAMPLE'; 
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient(['
        profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]); 
    echo getInvalidation($cloudFrontClient, $distributionId, $invalidationId);
}

// Uncomment the following line to run this code in an AWS account.
// getsAnInvalidation();
```

List Distribution Invalidations

To list all current CloudFront distribution invalidations, use the `ListInvalidations` operation.
Imports

```php
require 'vendor/autoload.php';

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function listInvalidations($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->listInvalidations([  
            'DistributionId' => $distributionId
        ]);  
        return $result;
    } catch (AwsException $e) {
        exit('Error: ' . $e->getAwsErrorMessage());
    }
}

function listTheInvalidations()
{
    $distributionId = 'E1WICG1EXAMPLE';
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient([  
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);  
    $invalidations = listInvalidations($cloudFrontClient, $distributionId);
    if (isset($invalidations['InvalidationList']))
    {
        if ($invalidations['InvalidationList']['Quantity'] > 0)
        {
            foreach ($invalidations['InvalidationList']['Items'] as $invalidation)
            {
                echo 'The invalidation with the ID of ' . $invalidation['Id'] .  
                    ' has the status of ' . $invalidation['Status'] . ' .  
                    ' . $invalidation['Status'] . ' . '
                    . "\n";
            }
        } else {
            echo 'Could not find any invalidations for the specified distribution.';
        }
    } else {
        echo 'Error: Could not get invalidation information. Could not get ' .  
            'information about the specified distribution.';
    }
}

// Uncomment the following line to run this code in an AWS account.
// listTheInvalidations();
```

Signing Amazon CloudFront URLs with AWS SDK for PHP Version 3

Signed URLs enable you to provide users access to your private content. A signed URL includes additional information (for example, expiration time) that gives you more control over access to your content. This additional information appears in a policy statement, which is based on either a canned policy or a
custom policy. For information about how to set up private distributions and why you need to sign URLs, see Serving Private Content through Amazon CloudFront in the Amazon CloudFront Developer Guide.

- Create a signed Amazon CloudFront URL using getSignedURL.
- Create a signed Amazon CloudFront cookie using getSignedCookie.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

## Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon CloudFront, see the Amazon CloudFront Developer Guide.

### Signing CloudFront URLs for Private Distributions

You can sign a URL using the CloudFront client in the SDK. First, you must create a CloudFrontClient object. You can sign a CloudFront URL for a video resource using either a canned or custom policy.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
function signPrivateDistribution($cloudFrontClient, $resourceKey, $expires,
    $privateKey, $keyPairId)
{
    try {
        $result = $cloudFrontClient->getSignedUrl(
            ['url' => $resourceKey,
             'expires' => $expires,
             'private_key' => $privateKey,
             'key_pair_id' => $keyPairId
            ]);
        return $result;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function signAPrivateDistribution()
{
    $resourceKey = 'https://d13l49jEXAMPLE.cloudfront.net/my-file.txt';
    $expires = time() + 300; // 5 minutes (5 * 60 seconds) from now.
    $privateKey = dirname(__DIR__) . '/cloudfront/my-private-key.pem';
    $keyPairId = 'AAPKAJIKZATYYEXAMPLE';

    $cloudFrontClient = new CloudFrontClient(
        ['profile' => 'default',
         'version' => '2014-11-06',
         'region' => 'us-east-1'
        ]);
}
```
Use a Custom Policy When Creating CloudFront URLs

To use a custom policy, provide the policy key instead of expires.

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function signPrivateDistributionPolicy($cloudFrontClient, $resourceKey, $customPolicy, $privateKey, $keyPairId)
{
    try {
        $result = $cloudFrontClient->getSignedUrl(
            'url' => $resourceKey,
            'policy' => $customPolicy,
            'private_key' => $privateKey,
            'key_pair_id' => $keyPairId
        );
        return $result;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function signAPrivateDistributionPolicy()
{
    $resourceKey = 'https://d13l49jEXAMPLE.cloudfront.net/my-file.txt';
    $expires = time() + 300; // 5 minutes (5 * 60 seconds) from now.
    $customPolicy = <<<POLICY
    {
        "Statement": [
            {
                "Resource": "{$resourceKey}"
            }
        ]
    }
POLICY
    $privateKey = dirname(__DIR__) . '/cloudfront/my-private-key.pem';
    $keyPairId = 'AAPKAJIKZATYYYEXAMPLE';
    $cloudFrontClient = new CloudFrontClient(
        'profile' => 'default',
        'version' => '2014-11-06',
    )
```
Use a CloudFront Signed URL

The form of the signed URL differs, depending on whether the URL you are signing is using the “HTTP” or “RTMP” scheme. In the case of “HTTP”, the full, absolute URL is returned. For “RTMP”, only the relative URL is returned for your convenience. This is because some players require the host and path to be provided as separate parameters.

The following example shows how you could use the signed URL to construct a webpage that displays a video using JWPlayer. The same type of technique would apply to other players such as FlowPlayer, but require different client-side code.

```html
<html>
<head>
  <title>CFlong Streaming Example</title>
  <script type="text/javascript" src="https://example.com/jwplayer.js"></script>
</head>
<body>
  <div id="video">The canned policy video will be here.</div>
  <script type="text/javascript">
    jwplayer('video').setup({
      width: "720",
      height: "480"
    });
  </script>
</body>
</html>
```

Signing CloudFront Cookies for Private Distributions

As an alternative to signed URLs, you can also grant clients access to a private distribution via signed cookies. Signed cookies enable you to provide access to multiple restricted files, such as all of the files for a video in HLS format or all of the files in the subscribers' area of a website. For more information on why you might want to use signed cookies instead of signed URLs (or vice versa), see Choosing Between Signed URLs and Signed Cookies in the Amazon CloudFront Developer Guide.

Creating a signed cookie is similar to creating a signed URL. The only difference is the method that is called (getSignedCookie instead of getSignedUrl).

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function signCookie($cloudFrontClient, $resourceKey, $expires,
```
Use a Custom Policy When Creating CloudFront Cookies

As with `getSignedUrl`, you can provide a 'policy' parameter instead of an `expires` parameter and a `url` parameter to sign a cookie with a custom policy. A custom policy can contain wildcards in the resource key. This enables you to create a single signed cookie for multiple files.

`getSignedCookie` returns an array of key-value pairs, all of which must be set as cookies to grant access to a private distribution.

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```
Sample Code

```php
function signCookiePolicy($cloudFrontClient, $customPolicy, $privateKey, $keyPairId)
{
    try {
        $result = $cloudFrontClient->getSignedCookie([  
            'policy' => $customPolicy,
            'private_key' => $privateKey,
            'key_pair_id' => $keyPairId
        ]);  
        return $result;
    } catch (AwsException $e) {
        return [ 'Error' => $e->getAwsErrorMessage() ];
    }
}

function signACookiePolicy()
{
    $resourceKey = 'https://d13l49jEXAMPLE.cloudfront.net/my-file.txt';  
    $expires = time() + 300; // 5 minutes (5 * 60 seconds) from now.
    $customPolicy = <<<POLICY
    {
        "Statement": [
            {  
                "Resource": "{$resourceKey}",
                "Condition": {
                    "IpAddress": {"AWS:SourceIp": "$_SERVER['REMOTE_ADDR']"}/32",
                    "DateLessThan": {"AWS:EpochTime": {$expires}}
                }
            }
        ]
    }
    POLICY;
    $privateKey = dirname(__DIR__) . '/cloudfront/my-private-key.pem';
    $keyPairId = 'AAPKAJIKZATYYEXAMPLE';
    $cloudFrontClient = new CloudFrontClient([  
        'profile' => 'default',
        'version' => '2014-11-06',
        'region' => 'us-east-1'
    ]);  
    $result = signCookiePolicy($cloudFrontClient, $customPolicy, $privateKey, $keyPairId);
    /* If successful, returns something like:
    CloudFront-Policy = eyJTdGF0...fX19XX0_
    CloudFront-Signature = RdwEQWZ...N8vetw_
    CloudFront-Key-Pair-Id = AAPKAJIKZATYYEXAMPLE
    */
    foreach ($result as $key => $value) {
        echo $key . ' = ' . $value . "\n";
    }
}

// Uncomment the following line to run this code in an AWS account.
// signACookiePolicy();
```

Send CloudFront Cookies to Guzzle Client

You can also pass these cookies to a GuzzleHttp\Cookie\CookieJar for use with a Guzzle client.
use GuzzleHttp\Client;
use GuzzleHttp\Cookie\CookieJar;

$distribution = "example-distribution.cloudfront.net";
$client = new \GuzzleHttp\Client(
    'base_uri' => "https://$distribution",
    'cookies' => CookieJar::fromArray($signedCookieCustomPolicy, $distribution),
);
$client->get('video.mp4');

For more information, see Using Signed Cookies in the Amazon CloudFront Developer Guide.

Signing Custom Amazon CloudSearch Domain Requests with AWS SDK for PHP Version 3

Amazon CloudSearch domain requests can be customized beyond what is supported by the AWS SDK for PHP. In cases where you need to make custom requests to domains protected by IAM authentication, you can use the SDK’s credential providers and signers to sign any PSR-7 request.

For example, if you’re following Cloud Search’s Getting Started guide and want to use an IAM-protected domain for Step 3, you would need to sign and execute your request as follows.

The following examples show how to:

- Sign a request with the AWS signing protocol using SignatureV4.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Sign CSlong Domain Request

Imports

require './vendor/autoload.php';
use Aws\Credentials\CredentialProvider;
use Aws\Signature\SignatureV4;
use GuzzleHttp\Client;
use GuzzleHttp\Psr7\Request;

Sample Code

function searchDomain($client, $domainName, $domainId, $domainRegion, $searchString)
{
    $domainPrefix = 'search-';
    $cloudSearchDomain = 'cloudsearch.amazonaws.com';
}
#cloudSearchVersion = '2013-01-01';
$searchPrefix = 'search?';

// Specify the search to send.
$request = new Request(
    'GET',
    "https://{$domainPrefix}{$domainName}-{$domainId}.{$domainRegion}." .
    "[{$cloudSearchDomain}]/{$cloudSearchVersion}/" .
    "{$searchPrefix}{$searchString}"
);

// Get default AWS account access credentials.
$credentials = call_user_func(CredentialProvider::defaultProvider())->wait();

// Sign the search request with the credentials.
$signer = new SignatureV4('cloudsearch', $domainRegion);
$request = $signer->signRequest($request, $credentials);

// Send the signed search request.
$response = $client->send($request);

// Report the search results, if any.
$results = json_decode($response->getBody());
$message = '';
if ($results->hits->found &gt; 0) {
    $message .= 'Search results:' . "\n";
    foreach($results->hits->hit as $hit)
    {
        $message .= $hit->fields->title . "\n";
    }
} else
{
    $message .= 'No search results.';
}

return $message;

function searchADomain()
{
    $domainName = 'my-search-domain';
    $domainId = '7kbitd6nyiglhtmsxEXAMPLE';
    $domainRegion = 'us-east-1';
    $searchString = 'q=star+wars&return=title';
    $client = new Client();

    echo searchDomain($client, $domainName, $domainId, $domainRegion, $searchString);
}

// Uncomment the following line to run this code in an AWS account.
// searchADomain();

Amazon CloudWatch Examples Using the AWS SDK for PHP Version 3

Amazon CloudWatch (CloudWatch) is a web service that monitors your Amazon Web Services resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics,
which are variables you can measure for your resources and applications. CloudWatch alarms send
notifications or automatically make changes to the resources you are monitoring based on rules that you
define.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the
AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage
Patterns of the AWS SDK for PHP Version 3 (p. 7).

Topics
  • Working with Amazon CloudWatch Alarms with AWS SDK for PHP Version 3 (p. 138)
  • Getting Metrics from Amazon CloudWatch with AWS SDK for PHP Version 3 (p. 142)
  • Publishing Custom Metrics in Amazon CloudWatch with AWS SDK for PHP Version 3 (p. 147)
  • Sending Events to Amazon CloudWatch Events with AWS SDK for PHP Version 3 (p. 150)
  • Using Alarm Actions with Amazon CloudWatch Alarms with AWS SDK for PHP Version 3 (p. 152)

Working with Amazon CloudWatch Alarms with AWS SDK for PHP Version 3

An Amazon CloudWatch alarm watches a single metric over a time period you specify. It performs one
or more actions based on the value of the metric relative to a given threshold over a number of time
periods.

The following examples show how to:

  • Describe an alarm using DescribeAlarms.
  • Create an alarm using PutMetricAlarm.
  • Delete an alarm using DeleteAlarms.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the
AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage
Patterns of the AWS SDK for PHP Version 3 (p. 7).

Describe Alarms

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code
function describeAlarms($cloudWatchClient)
{
    try {
        $result = $cloudWatchClient->describeAlarms();
        $message = '';

        if (isset($result['@metadata']['effectiveUri']))
        {
            $message .= 'Alarms at the effective URI of ' .
                         $result['@metadata']['effectiveUri'] . "\n\n";
            if (isset($result['CompositeAlarms']))
            {
                $message .= "Composite alarms:\n";
                foreach ($result['CompositeAlarms'] as $alarm) {
                    $message .= $alarm['AlarmName'] . "\n";
                }
            } else {
                $message .= 'No composite alarms found.\n';
            }
            if (isset($result['MetricAlarms']))
            {
                $message .= "Metric alarms:\n";
                foreach ($result['MetricAlarms'] as $alarm) {
                    $message .= $alarm['AlarmName'] . "\n";
                }
            } else {
                $message .= 'No metric alarms found.';
            }
            } else {
                $message .= 'No alarms found.';
            }

            return $message;
        } catch (AwsException $e) {
            return 'Error: ' . $e->getAwsErrorMessage();
        }
    }

    function describeTheAlarms()
    {
        $cloudWatchClient = new CloudWatchClient(["profile" => 'default',
                                                   'region' => 'us-east-1',
                                                   'version' => '2010-08-01']);
        echo describeAlarms($cloudWatchClient);
    }

    // Uncomment the following line to run this code in an AWS account.
    // describeTheAlarms();

Create an Alarm

Imports

require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;

Sample Code

```php
function putMetricAlarm($cloudWatchClient, $cloudWatchRegion,
                         $alarmName, $namespace, $metricName,
                         $dimensions, $statistic, $period, $comparison, $threshold,
                         $evaluationPeriods)
{
    try {
        $result = $cloudWatchClient->putMetricAlarm(
            ['AlarmName' => $alarmName,
             'Namespace' => $namespace,
             'MetricName' => $metricName,
             'Dimensions' => $dimensions,
             'Statistic' => $statistic,
             'Period' => $period,
             'ComparisonOperator' => $comparison,
             'Threshold' => $threshold,
             'EvaluationPeriods' => $evaluationPeriods
        ]);

        if (isset($result['@metadata']['effectiveUri']))
        {
            if ($result['@metadata']['effectiveUri'] ==
                'https://monitoring.' . $cloudWatchRegion . '.amazonaws.com')
                return 'Successfully created or updated specified alarm.';
            else {
                return 'Could not create or update specified alarm.';
            } // else
        } else {
            return 'Could not create or update specified alarm.';
        } // if
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    } // catch
}
```

```php
function putTheMetricAlarm()
{
    $alarmName = 'my-ec2-resources';
    $namespace = 'AWS/Usage';
    $metricName = 'ResourceCount';
    $dimensions = [
        ['Name' => 'Type', 'Value' => 'Resource'],
        ['Name' => 'Resource', 'Value' => 'vCPU'],
        ['Name' => 'Service', 'Value' => 'EC2'],
        ['Name' => 'Class', 'Value' => 'Standard/OnDemand']
    ];
    $statistic = 'Average';
```
$period = 300;
$comparison = 'GreaterThanThreshold';
$threshold = 1;
$evaluationPeriods = 1;

$cloudWatchRegion = 'us-east-1';
$cloudWatchClient = new CloudWatchClient(
    ['profile' => 'default',
     'region' => $cloudWatchRegion,
     'version' => '2010-08-01']);

echo putMetricAlarm($cloudWatchClient, $cloudWatchRegion,
                    $alarmName, $namespace, $metricName,
                    $dimensions, $statistic, $period, $comparison, $threshold,
                    $evaluationPeriods);
}

// Uncomment the following line to run this code in an AWS account.
// putTheMetricAlarm();

### Delete Alarms

**Imports**

```php
require 'vendor/autoload.php';

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
function deleteAlarms($cloudWatchClient, $alarmNames)
{
    try {
        $result = $cloudWatchClient->deleteAlarms(
            ['AlarmNames' => $alarmNames]);

        return 'The specified alarms at the following effective URI have been deleted or do not currently exist: ' . $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function deleteTheAlarms()
{
    $alarmNames = array('my-alarm');

    $cloudWatchClient = new CloudWatchClient(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => '2010-08-01']);

    echo deleteAlarms($cloudWatchClient, $alarmNames);
}

// Uncomment the following line to run this code in an AWS account.
```
Getting Metrics from Amazon CloudWatch with AWS SDK for PHP Version 3

Metrics are data about the performance of your systems. You can enable detailed monitoring of some resources, such as your Amazon EC2 instances, or of your own application metrics.

The following examples show how to:

- List metrics using `ListMetrics`.
- Retrieve alarms for a metric using `DescribeAlarmsForMetric`.
- Get statistics for a specified metric using `GetMetricStatistics`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

**List Metrics**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
function listMetrics($cloudWatchClient)
{
    try {
        $result = $cloudWatchClient->listMetrics();
        $message = '';

        if (isset($result['@metadata']['effectiveUri']))
        {
            $message .= 'For the effective URI at ' . $result['@metadata']['effectiveUri'] . ':

        if ((isset($result['Metrics']))) and (count($result['Metrics']) > 0))
        {
            $message .= "Metrics found:\n\n"

            foreach($result['Metrics'] as $metric)
            {
                $message .= 'For metric ' . $metric['MetricName'] . ' in namespace ' . $metric['Namespace'] . ':

                if ((isset($metric['Dimensions']))) and (count($metric['Dimensions']) > 0))
                {
```
$message .= "Dimensions:\n";

foreach ($metric['Dimensions'] as $dimension)
{
    $message .= 'Name: ' . $dimension['Name'] . 
        ', Value: ' . $dimension['Value'] . "\n";
}

$message .= "\n";
} else {
    $message .= "No dimensions.\n\n";
}
} else {
    $message .= 'No metrics found.';
}
} else {
    $message .= 'No metrics found.';
}

return $message;
} catch (AwsException $e) {
    return 'Error: ' . $e->getAwsErrorMessage();
}
}

function listTheMetrics()
{
    $cloudWatchClient = new CloudWatchClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => '2010-08-01'
    ]);;
    echo listMetrics($cloudWatchClient);
}

// Uncomment the following line to run this code in an AWS account.
// listTheMetrics();

Retrieve Alarms for a Metric

Imports

require 'vendor/autoload.php';

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;

Sample Code

function describeAlarmsForMetric($cloudWatchClient, $metricName, $namespace, $dimensions)
{
    try {
        $result = $cloudWatchClient->describeAlarmsForMetric([
            'MetricName' => $metricName,
            'Namespace' => $namespace,
            'Dimensions' => $dimensions
        ]);;
        $message = ";";
if (isset($result['@metadata']['effectiveUri']))
{
    $message .= 'At the effective URI of ' .
                $result['@metadata']['effectiveUri'] . ':

    if ((isset($result['MetricAlarms'])) and
        (count($result['MetricAlarms']) > 0))
    {
        $message .= 'Matching alarms for ' . $metricName . ':

            foreach ($result['MetricAlarms'] as $alarm)
            {
                $message .= $alarm['AlarmName'] . ':
            }
        } else {
            $message .= 'No matching alarms found for ' . $metricName . ':
        }
    } else {
        $message .= 'No matching alarms found for ' . $metricName . ':
    }
}
return $message;
}

function describeTheAlarmsForMetric()
{
    $metricName = 'BucketSizeBytes';
    $namespace = 'AWS/S3';
    $dimensions = [
        ['Name' => 'StorageType', 'Value' => 'StandardStorage'],
        ['Name' => 'BucketName', 'Value' => 'my-bucket']
    ];

    $cloudWatchClient = new CloudWatchClient(
        ['profile' => 'default',
        'region' => 'us-east-1',
        'version' => '2010-08-01']);

    echo describeAlarmsForMetric($cloudWatchClient, $metricName, $namespace, $dimensions);
}

// Uncomment the following line to run this code in an AWS account.
// describeTheAlarmsForMetric();

Get Metric Statistics

Imports

require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
Getting Metrics from Amazon CloudWatch

Sample Code

```php
function getMetricStatistics($cloudWatchClient, $namespace, $metricName, $dimensions, $startTime, $endTime, $period, $statistics, $unit) {
    try {
        $result = $cloudWatchClient->getMetricStatistics(["Namespace" => $namespace, 'MetricName' => $metricName, 'Dimensions' => $dimensions, 'StartTime' => $startTime, 'EndTime' => $endTime, 'Period' => $period, 'Statistics' => $statistics, 'Unit' => $unit]);
        $message = '';
        if (isset($result['@metadata']['effectiveUri'])) {
            $message .= 'For the effective URI at ' . $result['@metadata']['effectiveUri'] . "\n"
            if ((isset($result['Datapoints'])) and (count($result['Datapoints']) > 0)) {
                $message .= "Datapoints found:\n"
                foreach($result['Datapoints'] as $datapoint) {
                    foreach ($datapoint as $key => $value) {
                        $message .= $key . ' = ' . $value . "\n"
                    }
                    $message .= "\n";
                } else {
                    $message .= 'No datapoints found.';
                } else {
                    $message .= 'No datapoints found.';
                }
            } catch (AwsException $e) {
                return 'Error: ' . $e->getAwsErrorMessage();
            }
        }
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function getTheMetricStatistics() {
    // Average number of Amazon EC2 vCPUs every 5 minutes within
    // the past 3 hours.
    $namespace = 'AWS/Usage';
    $metricName = 'ResourceCount';
    $dimensions = [
        ['Name' => 'Service', 'Value' => 'EC2'],
        ['Name' => 'Resource', 'Value' => 'vCPU']
    ];
    $result = $cloudWatchClient->getMetricStatistics(["Namespace" => $namespace, 'MetricName' => $metricName, 'Dimensions' => $dimensions, 'StartTime' => $startTime, 'EndTime' => $endTime, 'Period' => $period, 'Statistics' => $statistics, 'Unit' => $unit]);
    $message = '';
    if (isset($result['@metadata']['effectiveUri'])) {
        $message .= 'For the effective URI at ' . $result['@metadata']['effectiveUri'] . "\n"
        if ((isset($result['Datapoints'])) and (count($result['Datapoints']) > 0)) {
            $message .= "Datapoints found:\n"
            foreach($result['Datapoints'] as $datapoint) {
                foreach ($datapoint as $key => $value) {
                    $message .= $key . ' = ' . $value . "\n"
                }
                $message .= "\n";
            } else {
                $message .= 'No datapoints found.';
            } else {
                $message .= 'No datapoints found.';
            }
        } catch (AwsException $e) {
            return 'Error: ' . $e->getAwsErrorMessage();
        }
    }
    return $message;
}
```


```
$startTime = strtotime('-3 hours');
$endTime = strtotime('now');
$period = 300; // Seconds. (5 minutes = 300 seconds.)
$statistics = array('Average');
$unit = 'None';

$cloudWatchClient = new CloudWatchClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-08-01'  
]);

echo getMetricStatistics($cloudWatchClient, $namespace, $metricName,  
    $dimensions, $startTime, $endTime, $period, $statistics, $unit);

// Another example: average number of bytes of standard storage in the  
// specified Amazon S3 bucket each day for the past 3 days.

/*
    $namespace = 'AWS/S3';
    $metricName = 'BucketSizeBytes';
    $dimensions = [
        [  
            'Name' => 'StorageType',  
            'Value'=> 'StandardStorage'
        ],
        [  
            'Name' => 'BucketName',  
            'Value' => 'my-bucket'
        ]
    ];
    $startTime = strtotime('-3 days');
    $endTime = strtotime('now');
    $period = 86400; // Seconds. (1 day = 86400 seconds.)
    $statistics = array('Average');
    $unit = 'Bytes';

    $cloudWatchClient = new CloudWatchClient([  
        'profile' => 'default',  
        'region' => 'us-east-1',  
        'version' => '2010-08-01'  
    ]);  

echo getMetricStatistics($cloudWatchClient, $namespace, $metricName,  
    $dimensions, $startTime, $endTime, $period, $statistics, $unit);
*/

// Uncomment the following line to run this code in an AWS account.
// getTheMetricStatistics();
```
Publishing Custom Metrics in Amazon CloudWatch with AWS SDK for PHP Version 3

Metrics are data about the performance of your systems. An alarm watches a single metric over a time period you specify. It performs one or more actions based on the value of the metric, relative to a given threshold over a number of time periods.

The following examples show how to:

- Publish metric data using PutMetricData.
- Create an alarm using PutMetricAlarm.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Publish Metric Data

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
function putMetricData($cloudWatchClient, $cloudWatchRegion, $namespace, $metricData)
{
    try {
        $result = $cloudWatchClient->putMetricData([ 
            'Namespace' => $namespace,
            'MetricData' => $metricData
        ]); 
        
        if (isset($result['@metadata']['effectiveUri']))
        {
            if ($result['@metadata']['effectiveUri'] == 'https://monitoring.' . $cloudWatchRegion . '.amazonaws.com')
            {
                return 'Successfully published datapoint(s).';
            } else {
                return 'Could not publish datapoint(s).';
            }
        } else {
            return 'Error: Could not publish datapoint(s).';
        }
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}
```
function putTheMetricData()
{
    $namespace = 'MyNamespace';
    $metricData = [
        'MetricName' => 'MyMetric',
        'Timestamp' => 1589228818, // 11 May 2020, 20:26:58 UTC.
        'Dimensions' => [
            'Name' => 'MyDimension1',
            'Value' => 'MyValue1'
        ],
        'Name' => 'MyDimension2',
        'Value' => 'MyValue2'
    ],
    'Unit' => 'Count',
    'Value' => 1
];

    $cloudWatchRegion = 'us-east-1';
    $cloudWatchClient = new CloudWatchClient([%
        'profile' => 'default',
        'region' => $cloudWatchRegion,
        'version' => '2010-08-01'
    ]);%

    echo putMetricData($cloudWatchClient, $cloudWatchRegion, $namespace, $metricData);
}

// Uncomment the following line to run this code in an AWS account.
// putTheMetricData();

Create an Alarm

Imports

require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;

Sample Code

function putMetricAlarm($cloudWatchClient, $cloudWatchRegion, $alarmName, $namespace, $metricName, $dimensions, $statistic, $period, $comparison, $threshold, $evaluationPeriods)
{
    try {
        $result = $cloudWatchClient->putMetricAlarm([%
            'AlarmName' => $alarmName,
            'Namespace' => $namespace,
            'MetricName' => $metricName,
            'Dimensions' => $dimensions,
            'Statistic' => $statistic,
            'Offset' => 0,
            'Period' => $period,
            'ComparisonOperator' => $comparison,
            'Threshold' => $threshold,
            'EvaluationPeriods' => $evaluationPeriods
        ]);%
    } catch (AwsException $e) {
        // Handle exception
    }
'Period' => $period,
'ComparisonOperator' => $comparison,
'Threshold' => $threshold,
'EvaluationPeriods' => $evaluationPeriods
]);

if (isset($result['@metadata']['effectiveUri']))
{
    if ($result['@metadata']['effectiveUri'] == 'https://monitoring.' . $cloudWatchRegion . '.amazonaws.com')
    {
        return 'Successfully created or updated specified alarm.;
    } else {
        return 'Could not create or update specified alarm.';
    }
} else {
    return 'Could not create or update specified alarm.';
}
} catch (AwsException $e) {
    return 'Error: ' . $e->getAwsErrorMessage();
}
}

function putTheMetricAlarm()
{
    $alarmName = 'my-ec2-resources';
    $namespace = 'AWS/Usage';
    $metricName = 'ResourceCount';
    $dimensions = [
        ['Name' => 'Type', 'Value' => 'Resource'],
        ['Name' => 'Resource', 'Value' => 'vCPU'],
        ['Name' => 'Service', 'Value' => 'EC2'],
        ['Name' => 'Class', 'Value' => 'Standard/OnDemand'],
    ];
    $statistic = 'Average';
    $period = 300;
    $comparison = 'GreaterThanThreshold';
    $threshold = 1;
    $evaluationPeriods = 1;

    $cloudWatchRegion = 'us-east-1';
    $cloudWatchClient = new CloudWatchClient(['profile' => 'default', 'region' => $cloudWatchRegion, 'version' => '2010-08-01']);

    echo putMetricAlarm($cloudWatchClient, $cloudWatchRegion, $alarmName, $namespace, $metricName, $dimensions, $statistic, $period, $comparison, $threshold, $evaluationPeriods);
}

// Uncomment the following line to run this code in an AWS account.
Sending Events to Amazon CloudWatch Events with AWS SDK for PHP Version 3

CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources to any of various targets. Using simple rules, you can match events and route them to one or more target functions or streams.

The following examples show how to:

• Create a rule using `PutRule`.
• Add targets to a rule using `PutTargets`.
• Send custom events to CloudWatch Events using `PutEvents`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Create a Rule

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudWatchEvents\CloudWatchEventsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#client = new Aws\cloudwatchevents\cloudwatcheventsClient([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2015-10-07']);
try {
    $result = $client->putRule(array(  'Name' => 'DEMO_EVENT', // REQUIRED  'RoleArn' => 'IAM_ROLE_ARN',  'ScheduleExpression' => 'rate(5 minutes)',  'State' => 'ENABLED',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
Add Targets to a Rule

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudWatchEvents\CloudWatchEventsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new Aws\cloudwatchevents\CloudWatchEventsClient([ 'profile' => 'default', 'region' => 'us-west-2', 'version' => '2015-10-07']);
try {
    $result = $client->putTargets([ 'Rule' => 'DEMO_EVENT', // REQUIRED 'Targets' => [ // REQUIRED ['Arn' => 'LAMBDA_FUNCTION_ARN', // REQUIRED 'Id' => 'myCloudWatchEventsTarget' // REQUIRED ], ],
    ],
); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Send Custom Events

Imports

```php
require 'vendor/autoload.php';
use Aws\CloudWatchEvents\CloudWatchEventsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new Aws\cloudwatchevents\CloudWatchEventsClient([ 'profile' => 'default', 'region' => 'us-west-2', 'version' => '2015-10-07']);
try {
    $result = $client->putEvents([ 'Entries' => [ // REQUIRED ['Detail' => '<string>', 'DetailType' => '<string>',
```
Using Alarm Actions with Amazon CloudWatch Alarms with AWS SDK for PHP Version 3

Use alarm actions to create alarms that automatically stop, terminate, reboot, or recover your Amazon EC2 instances. You can use the stop or terminate actions when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances.

The following examples show how to:

- Enable actions for specified alarms using `EnableAlarmActions`.
- Disable actions for specified alarms using `DisableAlarmActions`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP (p. 7).

**Enable Alarm Actions**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
function enableAlarmActions($cloudWatchClient, $alarmNames) {
    try {
        $result = $cloudWatchClient->enableAlarmActions(["AlarmNames" => $alarmNames]);
        if (isset($result["@metadata"]['effectiveUri'])) {
            return 'At the effective URI of ' . $result["@metadata"]['effectiveUri'] .
        }
    } catch (AwsException $e) {
        // output error message if fails
        error_log($e->getMessage());
    }
}
```
Using Alarm Actions with Amazon CloudWatch Alarms

Disable Alarm Actions

Imports

require 'vendor/autoload.php';

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;

Sample Code

function disableAlarmActions($cloudWatchClient, $alarmNames)
{
    try {
        $result = $cloudWatchClient->disableAlarmActions([
            'AlarmNames' => $alarmNames
        ]);  
        if (isset($result['@metadata']['effectiveUri']))
        {
            return 'At the effective URI of '.
                    $result['@metadata']['effectiveUri'] .
                    ', actions for any matching alarms have been disabled.';
        } else {
            return 'Actions for some matching alarms ' .
                    'might not have been disabled.';
        }
    }
    catch (AwsException $e) {
        return 'Error: '. $e->getAwsErrorMessage();
    }
}

function disableTheAlarmActions()
{
    $alarmNames = array('my-alarm');
    $cloudWatchClient = new CloudWatchClient([  'profile' => 'default',
            'region' => 'us-east-1',
            'version' => '2010-08-01'
    ]);  
    echo enableAlarmActions($cloudWatchClient, $alarmNames);
}

// Uncomment the following line to run this code in an AWS account.
// enableTheAlarmActions();
Amazon EC2 Examples Using the AWS SDK for PHP Version 3

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides virtual server hosting in the cloud. It's designed to make web-scale cloud computing easier for developers by providing resizeable compute capacity.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Topics

- Managing Amazon EC2 Instances Using the AWS SDK for PHP Version 3 (p. 154)
- Using Elastic IP Addresses with Amazon EC2 with AWS SDK for PHP Version 3 (p. 157)
- Using Regions and Availability Zones for Amazon EC2 with AWS SDK for PHP Version 3 (p. 159)
- Working with Amazon EC2 Key Pairs with AWS SDK for PHP Version 3 (p. 160)
- Working with Security Groups in Amazon EC2 with AWS SDK for PHP Version 3 (p. 162)

Managing Amazon EC2 Instances Using the AWS SDK for PHP Version 3

The following examples show how to:

- Describe Amazon EC2 instances using DescribeInstances.
- Enable detailed monitoring for a running instance using MonitorInstances.
- Disable monitoring for a running instance using UnmonitorInstances.
- Start an Amazon EBS-backed AMI that you've previously stopped, using StartInstances.
- Stop an Amazon EBS-backed instance using StopInstances.
- Request a reboot of one or more instances using RebootInstances.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.
**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

**Describe Instances**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

**Sample Code**

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
     'version' => '2016-11-15',
     'profile' => 'default'
    ]);,
$result = $ec2Client->describeInstances();
var_dump($result);
```

**Enable and Disable Monitoring**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

**Sample Code**

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
     'version' => '2016-11-15',
     'profile' => 'default'
    ]);,
$instanceIds = array('InstanceID1', 'InstanceID2');
$monitorInstance = 'ON';
if ($monitorInstance == 'ON') {
    $result = $ec2Client->monitorInstances(array(
        'InstanceIds' => $instanceIds
    ));,
} else {
    $result = $ec2Client->unmonitorInstances(array(
        'InstanceIds' => $instanceIds
    ));
```
Start and Stop an Instance

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([  
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);
$action = 'START';
$instanceIds = array('InstanceID1', 'InstanceID2');
if ($action == 'START') {
    $result = $ec2Client->startInstances(array(  
        'InstanceIds' => $instanceIds,
    ));
} else {
    $result = $ec2Client->stopInstances(array(  
        'InstanceIds' => $instanceIds,
    ));
}
var_dump($result);
```

Reboot an Instance

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([  
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);
$instanceIds = array('InstanceID1', 'InstanceID2');
```
Using Elastic IP Addresses with Amazon EC2 with AWS SDK for PHP Version 3

An Elastic IP address is a static IP address designed for dynamic cloud computing. An Elastic IP address is associated with your AWS account. It’s a public IP address, which is reachable from the internet. If your instance does not have a public IP address, you can associate an Elastic IP address with your instance to enable communication with the internet.

The following examples show how to:

- Describe one or more of your instances using DescribeInstances.
- Acquire an Elastic IP address using AllocateAddress.
- Associate an Elastic IP address with an instance using AssociateAddress.
- Release an Elastic IP address using ReleaseAddress.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Describe an Instance

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);
$result = $ec2Client->describeInstances();
var_dump($result);
```
Allocate and Associate an Address

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
#ec2Client = new Aws\Ec2\Ec2Client([  'region' => 'us-west-2',  'version' => '2016-11-15',  'profile' => 'default'
]);
$instanceId = 'InstanceID';
$allocation = $ec2Client->allocateAddress(array(  'DryRun' => false,  'Domain' => 'vpc',
));
$result = $ec2Client->associateAddress(array(  'DryRun' => false,  'InstanceId' => $instanceId,  'AllocationId' => $allocation->get('AllocationId')
));
var_dump($result);
```

Release an Address

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
#ec2Client = new Aws\Ec2\Ec2Client([  'region' => 'us-west-2',  'version' => '2016-11-15',  'profile' => 'default'
]);
$associationID = 'AssociationID';
$allocationID = 'AllocationID';
$result = $ec2Client->disassociateAddress(array(  'AssociationId' => $associationID,  'AllocationId' => $allocationID,
));
```
$result = $ec2Client->releaseAddress(array(
    'AllocationId' => $allocationID,
));
var_dump($result);

Using Regions and Availability Zones for Amazon EC2 with AWS SDK for PHP Version 3

Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of AWS Regions and Availability Zones. Each Region is a separate geographic area, with multiple isolated locations known as Availability Zones. Amazon EC2 provides the ability to place instances and data in multiple locations.

The following examples show how to:

- Describe the Availability Zones that are available to you using DescribeAvailabilityZones.
- Describe AWS Regions that are currently available to you using DescribeRegions.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Describe Availability Zones

Imports

require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;

Sample Code

$ec2Client = new Aws\Ec2\Ec2Client([  'region' => 'us-west-2',  'version' => '2016-11-15',  'profile' => 'default'  ]);  
$result = $ec2Client->describeAvailabilityZones();
var_dump($result);

Describe Regions

Imports
Working with Amazon EC2 Key Pairs with AWS SDK for PHP Version 3

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt data. Then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair.

The following examples show how to:

- Create a 2048-bit RSA key pair using `CreateKeyPair`.
- Delete a specified key pair using `DeleteKeyPair`.
- Describe one or more of your key pairs using `DescribeKeyPairs`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

## Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

### Create a Key Pair

#### Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

#### Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
     'version' => '2016-11-15',
     'profile' => 'default']
);
$result = $ec2Client->describeRegions();
var_dump($result);
```
Working with Key Pairs

Delete a Key Pair

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([  
    'region' => 'us-west-2',  
    'version' => '2016-11-15',  
    'profile' => 'default'  
]);

$keyPairName = 'my-keypair';

$result = $ec2Client->deleteKeyPair(array(  
    'KeyName' => $keyPairName  
));

var_dump($result);
```

Describe Key Pairs

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([  
    'region' => 'us-west-2',  
    'version' => '2016-11-15',  
    'profile' => 'default'  
]);
```

```
```
Working with Security Groups in Amazon EC2 with AWS SDK for PHP Version 3

An Amazon EC2 security group acts as a virtual firewall that controls the traffic for one or more instances. You add rules to each security group to allow traffic to or from its associated instances. You can modify the rules for a security group at any time. The new rules are automatically applied to all instances that are associated with the security group.

The following examples show how to:

- Describe one or more of your security groups using DescribeSecurityGroups.
- Add an ingress rule to a security group using AuthorizeSecurityGroupIngress.
- Create a security group using CreateSecurityGroup.
- Delete a security group using DeleteSecurityGroup.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

Credentials

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

Describe Security Groups

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
     'version' => '2016-11-15',
     'profile' => 'default'
   ]);
$result = $ec2Client->describeSecurityGroups();
var_dump($result);
```
Add an Ingress Rule

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([ 
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$result = $ec2Client->authorizeSecurityGroupIngress(array(
    'GroupName' => 'string',
    'SourceSecurityGroupName' => 'string'
));

var_dump($result);```

Create a Security Group

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client([ 
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

// Create the security group
$securityGroupName = 'my-security-group';
$result = $ec2Client->createSecurityGroup(array(
    'GroupId' => $securityGroupName,
));

// Get the security group ID (optional)
$securityGroupId = $result->get('GroupId');

echo "Security Group ID: " . $securityGroupId . "\n";```
Delete a Security Group

Imports

```php
require 'vendor/autoload.php';
use Aws\Ec2\Ec2Client;
```

Sample Code

```php
$ec2Client = new Aws\Ec2\Ec2Client(
    ['region' => 'us-west-2',
     'version' => '2016-11-15',
     'profile' => 'default']
);

$securityGroupId = 'my-security-group-id';

$result = $ec2Client->deleteSecurityGroup(array(
    'GroupId' => $securityGroupId
));

var_dump($result);
```

Signing an Amazon OpenSearch Service Search Request with AWS SDK for PHP Version 3

Amazon OpenSearch Service (OpenSearch Service) is a managed service that makes it easy to deploy, operate, and scale Amazon OpenSearch Service, a popular open-source search, and analytics engine. OpenSearch Service offers direct access to the Amazon OpenSearch Service API. This means that developers can use the tools with which they’re familiar, as well as robust security options, such as using IAM users and roles for access control. Many Amazon OpenSearch Service clients support request signing, but if you’re using a client that doesn’t, you can sign arbitrary PSR-7 requests with the built-in credential providers and signers of the AWS SDK for PHP.

The following examples show how to:

- Sign a request with the AWS signing protocol using SignatureV4.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

Credentials

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

Signing an OpenSearch Service Request

OpenSearch Service uses Signature Version 4. This means that you need to sign requests against the service’s signing name (es, in this case) and the AWS Region of your OpenSearch Service domain. A full
AWS Identity and Access Management (IAM) is a web service that enables Amazon Web Services customers to manage users and user permissions in AWS. The service is targeted at organizations with multiple users or systems in the cloud that use AWS products. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users can access.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

**Topics**

- Managing IAM Access Keys with AWS SDK for PHP Version 3 (p. 165)
- Managing IAM Users with AWS SDK for PHP Version 3 (p. 168)
- Using IAM Account Aliases with AWS SDK for PHP Version 3 (p. 171)
- Working with IAM Policies with AWS SDK for PHP Version 3 (p. 173)
- Working with IAM Server Certificates with AWS SDK for PHP Version 3 (p. 180)

**Managing IAM Access Keys with AWS SDK for PHP Version 3**

Users need their own access keys to make programmatic calls to AWS. To fill this need, you can create, modify, view, or rotate access keys (access key IDs and secret access keys) for IAM users. By default, when you create an access key, its status is Active. This means the user can use the access key for API calls.

The following examples show how to:
• Create a secret access key and corresponding access key ID using `CreateAccessKey`.
• Return information about the access key IDs associated with an IAM user using `ListAccessKeys`.
• Retrieve information about when an access key was last used using `GetAccessKeyLastUsed`.
• Change the status of an access key from Active to Inactive, or vice versa, using `UpdateAccessKey`.
• Delete an access key pair associated with an IAM user using `DeleteAccessKey`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/awslabs/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

**Create an Access Key**

### Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

### Sample Code

```php
$client = new IamClient(['profile' => 'default',
                        'region' => 'us-west-2',
                        'version' => '2010-05-08']);
try {
    $result = $client->createAccessKey(['UserName' => 'IAM_USER_NAME',
                                         'Profile' => 'default',
                                         'Region' => 'us-west-2',
                                         'Version' => '2010-05-08']);
    $keyID = $result['AccessKey']['AccessKeyId'];
    $createDate = $result['AccessKey']['CreateDate'];
    $userName = $result['AccessKey']['UserName'];
    $status = $result['AccessKey']['Status'];
    // $secretKey = $result['AccessKey']['SecretAccessKey']
    echo '<p>AccessKey ' . $keyID . ' created on ' . $createDate . '</p>
    echo '<p>Username: ' . $userName . '</p>
    echo '<p>Status: ' . $status . '</p>';
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

**List Access Keys**

### Imports

```php
require 'vendor/autoload.php';
```
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2010-05-08'  ]);  
try {  
    $result = $client->listAccessKeys();  
    var_dump($result);  
} catch (AwsException $e) { 
    // output error message if fails  
    error_log($e->getMessage()); 
}

Get Information about an Access Key’s Last Use

Imports

require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient([   'profile' => 'default',   'region' => 'us-west-2',   'version' => '2010-05-08'  ]);   
try { 
    $result = $client->getAccessKeyLastUsed([   'AccessKeyId' => 'ACCESS_KEY_ID', // REQUIRED  ]);   
    var_dump($result); 
} catch (AwsException $e) { 
    // output error message if fails  
    error_log($e->getMessage()); 
}

Update an Access Key

Imports

require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2010-05-08',
]);

try {
    $result = $client->updateAccessKey(
        ['AccessKeyId' => 'ACCESS_KEY_ID', // REQUIRED
         'Status' => 'Inactive', // REQUIRED
         'UserName' => 'IAM_USER_NAME',
     ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

Delete an Access Key

Imports

require 'vendor/autoload.php';

use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2010-05-08',
]);

try {
    $result = $client->deleteAccessKey(
        ['AccessKeyId' => 'ACCESS_KEY_ID', // REQUIRED
         'UserName' => 'IAM_USER_NAME',
     ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

Managing IAM Users with AWS SDK for PHP Version 3

An IAM user is an entity that you create in AWS to represent the person or service that uses it to interact with AWS. A user in AWS consists of a name and credentials.
The following examples show how to:

- Create a new IAM user using `CreateUser`.
- List IAM users using `ListUsers`.
- Update an IAM user using `UpdateUser`.
- Retrieve information about an IAM user using `GetUser`.
- Delete an IAM user using `DeleteUser`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

### Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

### Create an IAM User

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);

try {
    $result = $client->createUser(array(  
        // UserName is required  
        'UserName' => 'string',  
    ));
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

### List IAM Users

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
```
use Aws\Exception\AwsException;

Sample Code

```php
$client = new IamClient([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2010-05-08']);
try {
    $result = $client->listUsers();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Update an IAM User

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2010-05-08']);
try {
    $result = $client->updateUser(array(      // UserName is required       'UserName' => 'string1',       'NewUserName' => 'string'));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Get Information about an IAM User

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
```
use Aws\Exception\AwsException;

Sample Code

```php
$client = new IamClient([  
    'profile' => 'default', 
    'region' => 'us-west-2', 
    'version' => '2010-05-08' 
]);

try {  
    $result = $client->getUser(array(  
        'UserName' => 'string', 
    ));  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

Delete an IAM User

Imports

```php
require 'vendor/autoload.php';

use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient([  
    'profile' => 'default', 
    'region' => 'us-west-2', 
    'version' => '2010-05-08' 
]);

try {  
    $result = $client->deleteUser(array(  
        'UserName' => 'string'  
    ));  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

Using IAM Account Aliases with AWS SDK for PHP Version 3

If you want the URL for your sign-in page to contain your company name or other friendly identifier instead of your AWS account ID, you can create an alias for your AWS account ID. If you create an AWS account alias, your sign-in page URL changes to incorporate the alias.
The following examples show how to:

- Create an alias using `CreateAccountAlias`.
- List the alias associated with the AWS account using `ListAccountAliases`.
- Delete an alias using `DeleteAccountAlias`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

## Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

### Create an Alias

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);
try {
    $result = $client->createAccountAlias(array(  
        // AccountAlias is required  
        'AccountAlias' => 'string',  
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

### List Account Aliases

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
```
Delete an Alias

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2010-05-08'
]);

try {
    $result = $client->deleteAccountAlias(array(
        // AccountAlias is required
        'AccountAlias' => 'string',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Working with IAM Policies with AWS SDK for PHP Version 3

You grant permissions to a user by creating a policy. A policy is a document that lists the actions that a user can perform and the resources those actions can affect. By default, any actions or resources that are not explicitly allowed are denied. Policies can be created and attached to users, groups of users, roles assumed by users, and resources.

The following examples show how to:

- Create a managed policy using `CreatePolicy`.  

• Attach a policy to a role using `AttachRolePolicy`.
• Attach a policy to a user using `AttachUserPolicy`.
• Attach a policy to a group using `AttachGroupPolicy`.
• Remove a role policy using `DetachRolePolicy`.
• Remove a user policy using `DetachUserPolicy`.
• Remove a group policy using `DetachGroupPolicy`.
• Delete a managed policy using `DeletePolicy`.
• Delete a role policy using `DeleteRolePolicy`.
• Delete a user policy using `DeleteUserPolicy`.
• Delete a group policy using `DeleteGroupPolicy`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/awslabs/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

**Create a Policy**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2010-05-08']);

```
**Attach a Policy to a Role**

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);

$roleName = 'ROLE_NAME';

$policyName = 'AmazonDynamoDBFullAccess';

$policyArn = 'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess';

try {
    $attachedRolePolicies = $client->getIterator('ListAttachedRolePolicies', [
        'RoleName' => $roleName,
    ]);)
    if (count($attachedRolePolicies) > 0) {
        foreach ($attachedRolePolicies as $attachedRolePolicy) {
            if ($attachedRolePolicy['PolicyName'] == $policyName) {
                echo $policyName . " is already attached to this role. \n";
                exit();
            }
        }
    }
    $result = $client->attachRolePolicy(array(
        'RoleName' => $roleName,
        'PolicyArn' => $policyArn
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
error_log($e->getMessage());
}

## Attach a Policy to a User

### Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

### Sample Code

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);
$userName = 'USER_NAME';
$policyName = 'AmazonDynamoDBFullAccess';
$policyArn = 'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess';
try {
    $attachedUserPolicies = $client->getIterator('ListAttachedUserPolicies', [
        'UserName' => $userName,
    ]);
    if (count($attachedUserPolicies) > 0) {
        foreach ($attachedUserPolicies as $attachedUserPolicy) {
            if ($attachedUserPolicy['PolicyName'] == $policyName) {
                echo $policyName . " is already attached to this role. \n";
                exit();
            }
        }
    }
    $result = $client->attachUserPolicy(array(
        'UserName' => $userName,
        'PolicyArn' => $policyArn,
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

## Attach a Policy to a Group

### Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```
require 'vendor/autoload.php';

use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

**Sample Code**

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);

try {
    $result = $client->attachGroupPolicy(array(
        // GroupName is required
        'GroupName' => 'string',
        // PolicyArn is required
        'PolicyArn' => 'string',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

**Detach a User Policy**

**Imports**

```php
require 'vendor/autoload.php';

use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

**Sample Code**

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);

try {
    $result = $client->detachUserPolicy(array(
        // UserName is required
        'UserName' => 'string',
        // PolicyArn is required
        'PolicyArn' => 'string',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
Detach a Group Policy

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\Exception;
```

Sample Code

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);

try {  
    $result = $client->detachGroupPolicy(array(  
        // GroupName is required  
        'GroupName' => 'string',  
        // PolicyArn is required  
        'PolicyArn' => 'string',  
    ));  
    var_dump($result);
} catch (Exception $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

Delete a Policy

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\Exception;
```

Sample Code

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);

try {  
    $result = $client->deletePolicy(array(  
        // PolicyArn is required  
        'PolicyArn' => 'string'
    ));  
    var_dump($result);
} catch (Exception $e) {  
    // output error message if fails
    error_log($e->getMessage());
}
Delete a Role Policy

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);
try {
    $result = $client->deleteRolePolicy(array(
        // RoleName is required
        'RoleName' => 'string',
        // PolicyName is required
        'PolicyName' => 'string'
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a User Policy

Imports

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new IamClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2010-05-08']
);
try {
    $result = $client->deleteUserPolicy(array(
        // UserName is required
        'UserName' => 'string'
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
Delete a Group Policy

Imports

```
require 'vendor/autoload.php';

use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);

try {
    $result = $client->deleteGroupPolicy(array(  
        // GroupName is required  
        'GroupName' => 'string',  
        // PolicyName is required  
        'PolicyName' => 'string',  
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails  
    error_log($e->getMessage());
}
```

Working with IAM Server Certificates with AWS SDK for PHP Version 3

To enable HTTPS connections to your website or application on AWS, you need an SSL/TLS server certificate. To use a certificate that you obtained from an external provider with your website or application on AWS, you must upload the certificate to IAM or import it into AWS Certificate Manager.

The following examples show how to:

- List the certificates stored in IAM using `ListServerCertificates`.
- Retrieve information about a certificate using `GetServerCertificate`.
- Update a certificate using `UpdateServerCertificate`.
• Delete a certificate using `DeleteServerCertificate`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php/).  

**Credentials**

Before running the example code, configure your AWS credentials, as described in *Credentials for the AWS SDK for PHP Version 3* (p. 42). Then import the AWS SDK for PHP, as described in *Basic Usage Patterns of the AWS SDK for PHP Version 3* (p. 7).

### List Server Certificates

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);
try {
    $result = $client->listServerCertificates();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

### Retrieve a Server Certificate

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new IamClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2010-05-08'  
]);
try {
    // ...
```
$result = $client->getServerCertificate(array(
    // ServerCertificateName is required
    'ServerCertificateName' => 'string',
));
var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

Update a Server Certificate

Imports

require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient([    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2010-05-08'
]);
try {
    $result = $client->updateServerCertificate(array(        // ServerCertificateName is required
        'ServerCertificateName' => 'string',
        'NewServerCertificateName' => 'string',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

Delete a Server Certificate

Imports

require 'vendor/autoload.php';
use Aws\Iam\IamClient;
use Aws\Exception\AwsException;

Sample Code

$client = new IamClient([    'profile' => 'default',
    'region' => 'us-west-2',
]
try {
    $result = $client->deleteServerCertificate(array(
        // ServerCertificateName is required
        'ServerCertificateName' => 'string',
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

AWS Key Management Service Examples Using the AWS SDK for PHP Version 3

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. For more information about AWS KMS, see the Amazon KMS documentation. Whether you are writing secure PHP applications or sending data to other AWS services, AWS KMS helps you maintain control over who can use your master keys and gain access to your encrypted data.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Topics

- Working with Keys Using the AWS KMS API and the AWS SDK for PHP Version 3 (p. 183)
- Encrypting and Decrypting AWS KMS Data Keys Using the AWS SDK for PHP Version 3 (p. 187)
- Working with AWS KMS Key Policies Using the AWS SDK for PHP Version 3 (p. 190)
- Working with Grants Using the AWS KMS API and the AWS SDK for PHP Version 3 (p. 193)
- Working with Aliases Using the AWS KMS API and the AWS SDK for PHP Version 3 (p. 196)

Working with Keys Using the AWS KMS API and the AWS SDK for PHP Version 3

The primary resources in AWS Key Management Service (AWS KMS) are customer master keys (CMKs). You can use a CMK to encrypt your data.

The following examples show how to:

- Create a customer CMK using CreateKey.
- Generate a data key using GenerateDataKey.
- View a CMK using DescribeKey.
- Get key IDs and key ARNS of CMKs using ListKeys.
- Enable CMKs using EnableKey.
- Disable CMKs using DisableKey.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.
Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

Create a CMK

To create a CMK, use the CreateKey operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient(['profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2']);

// Creates a customer master key (CMK) in the caller's AWS account.
#desc = "Key for protecting critical data";

try {
    $result = $KmsClient->createKey(['
        Description' => $desc,
    ]);}
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Generate a Data Key

To generate a data encryption key, use the GenerateDataKey operation. This operation returns plaintext and encrypted copies of the data key that it creates. Specify the customer master key (CMK) under which to generate the data key.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```
Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([  'profile' => 'default',  'version' => '2014-11-01',  'region' => 'us-east-2']);

$KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$keySpec = 'AES_256';

try {  
    $result = $KmsClient->generateDataKey([  'KeyId' => $KeyId,  'KeySpec' => $keySpec,  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

View a CMK

To get detailed information about a CMK, including the CMK's Amazon Resource Name (ARN) and key state, use the DescribeKey operation.

DescribeKey doesn't get aliases. To get aliases, use the ListAliases operation.

Imports

```php
require 'vendor/autoload.php';

use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([  'profile' => 'default',  'version' => '2014-11-01',  'region' => 'us-east-2']);

$KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

try {  
    $result = $KmsClient->describeKey([  'KeyId' => $KeyId,  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```
Get the Key ID and Key ARNs of a CMK

To get the ID and ARN of the CMK, use the ListAliases operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);
$limit = 10;
try {
    $result = $KmsClient->listKeys(
        ['Limit' => $limit,
        ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Enable a CMK

To enable a disabled CMK, use the EnableKey operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);
#keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
```
Encrypting and Decrypting AWS KMS Data Keys
Using the AWS SDK for PHP Version 3

Data keys are encryption keys that you can use to encrypt data, including large amounts of data and other data encryption keys.

You can use a AWS Key Management Service (AWS KMS) customer master key (CMK) to generate, encrypt, and decrypt data keys. However, AWS KMS does not store, manage, or track your data keys, or perform cryptographic operations with data keys. Use and manage data keys outside of AWS KMS.

The following examples show how to:
• Encrypt a data key using Encrypt.
• Decrypt a data key using Decrypt.
• Re-encrypt a data key with a new CMK using ReEncrypt.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

**Encrypt**

The Encrypt operation is designed to encrypt data keys, but it's not frequently used. The GenerateDataKey and GenerateDataKeyWithoutPlaintext operations return encrypted data keys. You might use the Encrypt method when you're moving encrypted data to a new AWS Region and want to encrypt its data key by using a CMK in the new Region.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
#KmsClient = new Aws\Kms\KmsClient([  
    'profile' => 'default',  
    'version' => '2014-11-01',  
    'region' => 'us-east-2'  
]);
#keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
#message = pack('c*', 1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
try {  
    $result = $KmsClient->encrypt([  
        'KeyId' => $keyId,  
        'Plaintext' => $message,  
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

**Decrypt**

To decrypt a data key, use the Decrypt operation.
The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a `GenerateDataKey`, `GenerateDataKeyWithoutPlaintext`, or `Encrypt` response.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
     'version' => '2014-11-01',
     'region' => 'us-east-2']
);
$ciphertext = 'Place your cipher text blob here';
try {
    $result = $KmsClient->decrypt(
        ['CiphertextBlob' => $ciphertext,
         ]);
    $plaintext = $result['Plaintext'];
    var_dump($plaintext);
} catch (AwsException $e) {
    // Output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

**Reencrypt**

To decrypt an encrypted data key, and then immediately reencrypt the data key under a different CMK, use the ReEncrypt operation. The operations are performed entirely on the server side within AWS KMS, so they never expose your plaintext outside of AWS KMS.

The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a `GenerateDataKey`, `GenerateDataKeyWithoutPlaintext`, or `Encrypt` response.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
#KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
     'version' => '2014-11-01',
     'region' => 'us-east-2']
);
#ciphertext = 'Place your cipher text blob here';
try {
    $result = #KmsClient->decrypt(
        ['CiphertextBlob' => #ciphertext,
         ]);
    #plaintext = $result['Plaintext'];
    var_dump(#plaintext);
} catch (AwsException $e) {
    // Output error message if fails
    echo $e->getMessage();
    echo "\n";
}
Working with AWS KMS Key Policies Using the AWS SDK for PHP Version 3

When you create an AWS Key Management Service (AWS KMS) customer master key (CMK), you determine who can use and manage that CMK. These permissions are contained in a document called the key policy. You can use the key policy to add, remove, or modify permissions at any time for a customer-managed CMK, but you cannot edit the key policy for a CMK that’s managed by AWS. For more information, see Authentication and Access Control for AWS KMS.

The following examples show how to:

- List the names of key policies using ListKeyPolicies.
- Get a key policy using GetKeyPolicy.
- Set a key policy using PutKeyPolicy.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

List All Key Policies

To get the names of key policies for a CMK, use the ListKeyPolicies operation. The only key policy name it returns is the default name.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```
Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([ 'profile' => 'default', 'version' => '2014-11-01', 'region' => 'us-east-2']);
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$limit = 10;
try {
    $result = $KmsClient->listKeyPolicies([ 'KeyId' => $keyId, 'Limit' => $limit, ]);    
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Retrieve a Key Policy

To get the key policy for a CMK, use the `GetKeyPolicy` operation.

`GetKeyPolicy` requires a policy name. Unless you created a key policy when you created the CMK, the only valid policy name is the default. Learn more about the default key policy.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([ 'profile' => 'default', 'version' => '2014-11-01', 'region' => 'us-east-2']);
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$policyName = "default";
try {
    $result = $KmsClient->getKeyPolicy([ 'KeyId' => $keyId, 'PolicyName' => $policyName ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```
Set a Key Policy

To establish or change a key policy for a CMK, use the `PutKeyPolicy` operation.

PutKeyPolicy requires a policy name. Unless you created a Key Policy when you created the CMK, the only valid policy name is the default. Learn more about the Default Key Policy.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient(
    [
    'profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
    ]);

$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$policyName = "default";

try {
    $result = $KmsClient->putKeyPolicy(
        [
        'KeyId' => $keyId,
        'PolicyName' => $policyName,
        'Policy' => '{
            "Version": "2012-10-17",
            "Id": "custom-policy-2016-12-07",
            "Statement": [
                {
                    "Sid": "Enable IAM User Permissions",
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::111122223333:user/root" },
                    "Action": [ "kms:*" ],
                    "Resource": "*" },
                {
                    "Sid": "Enable IAM User Permissions",
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::111122223333:user/ExampleUser" },
                    "Action": [ "kms:Encrypt*", "kms:GenerateDataKey*", "kms:Decrypt*", "kms:DescribeKey*", "kms:ReEncrypt*" ],
                    "Resource": "*" } ]
        } );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    
    var_dump($e);
}
```
Working with Grants Using the AWS KMS API and the AWS SDK for PHP Version 3

A grant is another mechanism for providing permissions, an alternative to the key policy. You can use grants to give long-term access that allows AWS principals to use your AWS Key Management Service (AWS KMS) customer-managed CMKs. For more information, see Using Grants.

The following examples show how to:

- Create a grant for a customer master key (CMK) using CreateGrant.
- View a grant for a CMK using ListGrants.
- Retire a grant for a CMK using RetireGrant.
- Revoke a grant for a CMK using RevokeGrant.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

Create a Grant

To create a grant for an AWS KMS CMK, use the CreateGrant operation.

Imports

```php
require 'vendor/autoload.php';

use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([  'profile' => 'default',  'version' => '2014-11-01',  'region' => 'us-east-2']);

$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$granteePrincipal = "arn:aws:iam::111122223333:user/Alice";
$operation = ['Encrypt', 'Decrypt']; // A list of operations that the grant allows.

try {
```
$result = $KmsClient->createGrant(
    ['GranteePrincipal' => $granteePrincipal,
     'KeyId' => $keyId,
     'Operations' => $operation
    ]); 
var_dump($result); 
} catch (AwsException $e) { 
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

View a Grant

To get detailed information about the grants on an AWS KMS CMK, use the ListGrants operation.

Imports

```
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
     'version' => '2014-11-01',
     'region' => 'us-east-2'
    ]); 
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$limit = 10;
try { 
    $result = $KmsClient->listGrants(
        ['KeyId' => $keyId,
         'Limit' => $limit,
        ]); 
    var_dump($result); 
} catch (AwsException $e) { 
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Retire a Grant

To retire a grant for an AWS KMS CMK, use the RetireGrant operation. Retire a grant to clean up after you finish using it.

Imports

```
require 'vendor/autoload.php';
```
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient(
    ['profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);
$grantToken = 'Place your grant token here';

try {
    $result = $KmsClient->retireGrant(
        ['GrantToken' => $grantToken,
    ]); 
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

//Can also identify grant to retire by a combination of the grant ID and the Amazon
Resource Name (ARN) of the customer master key (CMK)
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$grantId = 'Unique identifier of the grant returned during CreateGrant operation'

try {
    $result = $KmsClient->retireGrant(
        ['GrantId' => $grantToken,
        'KeyId' => $keyId,
    ]); 
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Revoke a Grant

To revoke a grant to an AWS KMS CMK, use the RevokeGrant operation. You can revoke a grant to explicitly deny operations that depend on it.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([}
Working with Aliases Using the AWS KMS API and the AWS SDK for PHP Version 3

An alias is an optional display name for an AWS Key Management Service (AWS KMS) customer master key (CMK).

The following examples show how to:

- Create an alias using `CreateAlias`.
- View an alias using `ListAliases`.
- Update an alias using `UpdateAlias`.
- Delete an alias using `DeleteAlias`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/get-started.html#credentials). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/basicsdk.html).

For more information about using AWS Key Management Service (AWS KMS), see the [AWS KMS Developer Guide](https://docs.aws.amazon.com/kms/latest/developerguide/).
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([
    'profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$aliasName = "alias/projectKey1";
try {
    $result = $KmsClient->createAlias(["AliasName" => $aliasName, 
                                         'TargetKeyId' => $keyId, 
                          ]); 
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

View an Alias

To list all aliases, use the ListAliases operation. The response includes aliases that are defined by AWS services, but are not associated with a CMK.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([
    'profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);
$limit = 10;
try {
    $result = $KmsClient->listAliases(["Limit" => $limit, 
                                         ]); 
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
```
Update an Alias

To associate an existing alias with a different CMK, use the UpdateAlias operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$KmsClient = new Aws\Kms\KmsClient([  'profile' => 'default',  'version' => '2014-11-01',  'region' => 'us-east-2']);
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$aliasName = "alias/projectKey1";

try {  
    $result = $KmsClient->updateAlias([  'AliasName' => $aliasName,  'TargetKeyId' => $keyId,  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

Delete an Alias

To delete an alias, use the DeleteAlias operation. Deleting an alias has no effect on the underlying CMK.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#KmsClient = new Aws\Kms\KmsClient([  'profile' => 'default',  'version' => '2014-11-01',  ]);
#keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
#aliasName = "alias/projectKey1";

try {  
    $result = $KmsClient->deleteAlias([  'AliasName' => #aliasName,  'TargetKeyId' => #keyId,  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}  
```
Amazon Kinesis Examples Using the AWS SDK for PHP Version 3

Amazon Kinesis is an AWS service that collects, processes, and analyzes data in real time. Configure your data streams with Amazon Kinesis Data Streams or use Amazon Kinesis Data Firehose to send data to Amazon S3, OpenSearch Service, Amazon Redshift, or Splunk.

For more information about Kinesis, see the Amazon Kinesis documentation.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Topics
- Creating Data Streams Using the Kinesis Data Streams API and the AWS SDK for PHP Version 3 (p. 199)
- Manage Data Shards Using the Kinesis Data Streams API and the AWS SDK for PHP Version 3 (p. 203)
- Creating Delivery Streams Using the Kinesis Data Firehose API and the AWS SDK for PHP Version 3 (p. 205)

Creating Data Streams Using the Kinesis Data Streams API and the AWS SDK for PHP Version 3

Amazon Kinesis Data Streams allows you to send real-time data. Create a data producer with Kinesis Data Streams that delivers data to the configured destination every time you add data.

For more information, see Creating and Managing Streams in the Amazon Kinesis Developer Guide.

The following examples show how to:
- Create a data stream using CreateAlias.
- Get details about a single data stream using DescribeStream.
- List existing data streams using ListStreams.
- Send data to an existing data stream using PutRecord.
- Delete a data stream using DeleteStream.
All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

## Credentials

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

For more information about using Amazon Kinesis Developer Guide, see the Amazon Kinesis Data Streams Developer Guide.

## Create a Data Stream Using a Kinesis Data Stream

Establish a Kinesis data stream where you can send information to be processed by Kinesis using the following code example. Learn more about [Creating and Updating Data Streams](p. 42) in the Amazon Kinesis Developer Guide.

To create a Kinesis data stream, use the `CreateStream` operation.

### Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

### Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient(
    ['profile' => 'default',
     'version' => '2013-12-02',
     'region' => 'us-east-2'
    ]);

$shardCount = 2;
$name = "my_stream_name"

try {
    $result = $kinesisClient->createStream(
        ['ShardCount' => $shardCount,
         'StreamName' => $name,]
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

## Retrieve a Data Stream

Get details about an existing data stream using the following code example. By default, this returns information about the first 10 shards connected to the specified Kinesis data stream. Remember to check `StreamStatus` from the response before writing data to a Kinesis data stream.

To retrieve details about a specified Kinesis data stream, use the `DescribeStream` operation.
Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient([  'profile' => 'default',  'version' => '2013-12-02',  'region' => 'us-east-2']);
$name = "my_stream_name";
try {  
$kinesisClient->describeStream(['StreamName' => $name,  
]);  
var_dump($result);
} catch (AwsException $e) {  
// output error message if fails  
echo $e->getMessage();  
echo "\n";
}
```

List Existing Data Streams That Are Connected to Kinesis

List the first 10 data streams from your AWS account in the selected AWS Region. Use the returned `HasMoreStreams` to determine if there are more streams associated with your account.

To list your Kinesis data streams, use the ListStreams operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient([  'profile' => 'default',  'version' => '2013-12-02',  'region' => 'us-east-2']);
try {  
$kinesisClient->listStreams([  
]);  
var_dump($result);
} catch (AwsException $e) {  
// output error message if fails  
echo $e->getMessage();  
echo "\n";
}
Send Data to an Existing Data Stream

Once you create a data stream, use the following example to send data. Before sending data to it, use DescribeStream to check whether the data StreamStatus is active.

To write a single data record to a Kinesis data stream, use the PutRecord operation. To write up to 500 records into a Kinesis data stream, use the PutRecords operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient(
    'profile' => 'default',
    'version' => '2013-12-02',
    'region' => 'us-east-1'
);
$name = "my_stream_name";
$content = '{"ticker_symbol":"QXZ", "sector":"HEALTHCARE", "change":-0.05, "price":84.51}';
$groupID = "input to a hash function that maps the partition key (and associated data) to a specific shard";
try {
    $result = $kinesisClient->PutRecord(
        'DataStream' => $name,
        'PartitionKey' => $groupID
    );
    print("ShardID = " . $result["ShardId"] . "\n");
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Delete a Data Stream

This example demonstrates how to delete a data stream. Deleting a data stream also deletes any data you sent to the data stream. Active Kinesis data streams switch to the DELETING state until the stream deletion is complete. While in the DELETING state, the stream continues to process data.

To delete a Kinesis data stream, use the DeleteStream operation.

Imports

```php
require 'vendor/autoload.php';
```
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;

Sample Code

```php
#kinesisClient = new Aws\Kinesis\KinesisClient([  'profile' => 'default',  'version' => '2013-12-02',  'region' => 'us-east-2' ]);  
$name = "my_stream_name";

try {  
    $result = $kinesisClient->deleteStream([  'StreamName' => $name,  ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";  
}
```

Manage Data Shards Using the Kinesis Data Streams API and the AWS SDK for PHP Version 3

Amazon Kinesis Data Streams enables you to send real-time data to an endpoint. The rate of data flow depends on the number of shards in your stream.

You can write 1,000 records per second to a single shard. Each shard also has an upload limit of 1 MiB per second. Usage is calculated and charged on a per-shard basis, so use these examples to manage the data capacity and cost of your stream.

The following examples show how to:

- List shards in a stream using `ListShards`.
- Add or reduce the number of shards in a stream using `UpdateShardCount`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

Credentials

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3 (p. 42)](#). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7)](#).

For more information about using Amazon Kinesis Data Streams, see the [Amazon Kinesis Data Streams Developer Guide](https://docs.aws.amazon.com/kinesis/latest/dev/)

List Data Stream Shards

List the details of up to 100 shards in a specific stream.
To list the shards in a Kinesis data stream, use the `ListShards` operation.

### Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

### Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient([  'profile' => 'default',  'version' => '2013-12-02',  'region' => 'us-east-2'  ]);  $name = "my_stream_name";
try {
    $result = $kinesisClient->ListShards([  'StreamName' => $name,  ]);  var_dump($result);
} catch (AwsException $e) {
    // output error message if fails  echo $e->getMessage();  echo "\n";
}
```

### Add More Data Stream Shards

If you need more data stream shards, you can increase your current number of shards. We recommend that you double your shard count when increasing. This makes a copy of each shard currently available to increase your capacity. You can double the number of your shards only twice in one 24-hour period.

Remember that billing for Kinesis Data Streams usage is calculated per shard, so when demand decreases, we recommend that you reduce your shard count by half. When you remove shards, you can only scale down the amount of shards to half of your current shard count.

To update the shard count of a Kinesis data stream, use the `UpdateShardCount` operation.

### Imports

```php
require 'vendor/autoload.php';
use Aws\Kinesis\KinesisClient;
use Aws\Exception\AwsException;
```

### Sample Code

```php
$kinesisClient = new Aws\Kinesis\KinesisClient([  'profile' => 'default',  'version' => '2013-12-02',  'region' => 'us-east-2'  ]);
$name = "my_stream_name";
$totalshards = 4;

try {
    $result = $kinesisClient->UpdateShardCount([  
        'ScalingType' => 'UNIFORM_SCALING',  
        'StreamName' => $name,  
        'TargetShardCount' => $totalshards  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}

Creating Delivery Streams Using the Kinesis Data Firehose API and the AWS SDK for PHP Version 3

Amazon Kinesis Data Firehose enables you to send real-time data to other AWS services including Amazon Kinesis Data Streams, Amazon S3, Amazon OpenSearch Service (OpenSearch Service), and Amazon Redshift, or to Splunk. Create a data producer with delivery streams to deliver data to the configured destination every time you add data.

The following examples show how to:

- Create a delivery stream using CreateDeliveryStream.
- Get details about a single delivery stream using DescribeDeliveryStream.
- List your delivery streams using ListDeliveryStreams.
- Send data to a delivery stream using PutRecord.
- Delete a delivery stream using DeleteDeliveryStream.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon Kinesis Data Firehose, see the Amazon Kinesis Data Firehose Developer Guide.

Create a Delivery Stream Using a Kinesis Data Stream

To establish a delivery stream that puts data into an existing Kinesis data stream, use the CreateDeliveryStream operation.

This enables developers to migrate existing Kinesis services to Kinesis Data Firehose.

Imports
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;

Sample Code

```php
#firehoseClient = new Aws\Firehose\FirehoseClient([  
    'profile' => 'default',  
    'version' => '2015-08-04',  
    'region' => 'us-east-2'  
]);

$name = "my_stream_name";
$stream_type = "KinesisStreamAsSource";
$role = "arn:aws:iam::0123456789:policy/Role";

try {  
    $result = $firehoseClient->createDeliveryStream([  
        'DeliveryStreamName' => $name,  
        'DeliveryStreamType' => $stream_type,  
        'KinesisStreamSourceConfiguration' => [  
            'KinesisStreamARN' => $kinesis_stream,  
            'RoleARN' => $role,  
        ],  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";  
}
```

Create a Delivery Stream Using an Amazon S3 Bucket

To establish a delivery stream that puts data into an existing Amazon S3 bucket, use the `CreateDeliveryStream` operation.

Provide the destination parameters, as described in Destination Parameters. Then ensure that you grant Kinesis Data Firehose access to your Amazon S3 bucket, as described in Grant Kinesis Data Firehose Access to an Amazon S3 Destination.

Imports

```php
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#firehoseClient = new Aws\Firehose\FirehoseClient([  
    'profile' => 'default',  
    'version' => '2015-08-04',  
    'region' => 'us-east-2'  
]);
```
Create a Delivery Stream Using OpenSearch Service

To establish a Kinesis Data Firehose delivery stream that puts data into an OpenSearch Service cluster, use the `CreateDeliveryStream` operation.

Provide the destination parameters, as described in Destination Parameters. Ensure that you grant Kinesis Data Firehose access to your OpenSearch Service cluster, as described in Grant Kinesis Data Firehose Access to an Amazon ES Destination.

Imports

```php
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#firehoseClient = new Aws\Firehose\FirehoseClient([  'profile' => 'default',  'version' => '2015-08-04',  'region' => 'us-east-2' ]);  
$name = "my_ES_stream_name";
#stream_type = "DirectPut";
#esDomainARN = 'arn:aws:es:us-east-2:0123456789:domain/Name';
#esRole = 'arn:aws:iam::0123456789:policy/Role';
#esIndex = 'root';
#esType = 'PHP_SDK';
#s3bucket = 'arn:aws:s3:::bucket_name';
#s3Role = 'arn:aws:iam::0123456789:policy/Role';
try {
```
Retrieve a Delivery Stream

To get the details about an existing Kinesis Data Firehose delivery stream, use the `DescribeDeliveryStream` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$firehoseClient = new Aws\Firehose\FirehoseClient(['profile' => 'default', 'version' => '2015-08-04', 'region' => 'us-east-2']);
$name = "my_stream_name";
try {
    $result = $firehoseClient->describeDeliveryStream(['DeliveryStreamName' => $name, ]);} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
List Existing Delivery Streams Connected to Kinesis Data Streams

To list all the existing Kinesis Data Firehose delivery streams sending data to Kinesis Data Streams, use the `ListDeliveryStreams` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$firehoseClient = new Aws\Firehose\FirehoseClient([  'profile' => 'default',  'version' => '2015-08-04',  'region' => 'us-east-2' ]);  
try {  
    $result = $firehoseClient->listDeliveryStreams([  'DeliveryStreamType' => 'KinesisStreamAsSource',  ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";  
}
```

List Existing Delivery Streams Sending Data to Other AWS Services

To list all the existing Kinesis Data Firehose delivery streams sending data to Amazon S3, OpenSearch Service, or Amazon Redshift, or to Splunk, use the `ListDeliveryStreams` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$firehoseClient = new Aws\Firehose\FirehoseClient([  'profile' => 'default',  'version' => '2015-08-04',  'region' => 'us-east-2' ]);  
try {  
    $result = $firehoseClient->listDeliveryStreams([  'DeliveryStreamType' => 'KinesisStreamAsSource',  ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";  
}  
```
'profile' => 'default',
'version' => '2015-08-04',
'region' => 'us-east-2'
});

try {
    $result = $firehoseClient->listDeliveryStreams(
        'DeliveryStreamType' => 'DirectPut',
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

Send Data to an Existing Kinesis Data Firehose Delivery Stream

To send data through a Kinesis Data Firehose delivery stream to your specified destination, use the PutRecord operation after you create a Kinesis Data Firehose delivery stream.

Before sending data to a Kinesis Data Firehose delivery stream, use DescribeDeliveryStream to see if the delivery stream is active.

Imports

```
require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$firehoseClient = new Aws\Firehose\FirehoseClient(
    'profile' => 'default',
    'version' => '2015-08-04',
    'region' => 'us-east-2'
);

$name = "my_stream_name";
$content = '{"ticker_symbol":"QXZ", "sector":"HEALTHCARE", "change":-0.05, "price":84.51}";

try {
    $result = $firehoseClient->putRecord(
        'DeliveryStreamName' => $name,
        'Record' => [
            'Data' => $content,
        ],
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```
AWS SDK for PHP Developer Guide
AWS Elemental MediaConvertt

Delete a Kinesis Data Firehose Delivery Stream
To delete a Kinesis Data Firehose delivery stream, use the DeleteDeliveryStreams operation. This also
deletes any data you have sent to the delivery stream.
Imports

require 'vendor/autoload.php';
use Aws\Firehose\FirehoseClient;
use Aws\Exception\AwsException;

Sample Code
$firehoseClient = new Aws\Firehose\FirehoseClient([
'profile' => 'default',
'version' => '2015-08-04',
'region' => 'us-east-2'
]);
$name = "my_stream_name";
try {
$result = $firehoseClient->deleteDeliveryStream([
'DeliveryStreamName' => $name,
]);
var_dump($result);
} catch (AwsException $e) {
// output error message if fails
echo $e->getMessage();
echo "\n";
}

AWS Elemental MediaConvert Examples Using the
AWS SDK for PHP Version 3
AWS Elemental MediaConvert is a ﬁle-based video transcoding service with broadcast-grade features.
You can use it to create assets for broadcast and for video-on-demand (VOD) delivery across the internet.
For more information, see the AWS Elemental MediaConvert User Guide.
The PHP API for AWS Elemental MediaConvert is exposed through the AWS.MediaConvert client class.
For more information, see Class: AWS.MediaConvert in the API reference.
All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials
Before running the example code, conﬁgure your AWS credentials, as described in Credentials for the
AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage
Patterns of the AWS SDK for PHP Version 3 (p. 7).
Topics

211


Getting Your Account-Specific Endpoint for AWS Elemental MediaConvert with AWS SDK for PHP Version 3

In this example, you use the AWS SDK for PHP Version 3 to call AWS Elemental MediaConvert and retrieve your account-specific endpoint. You can retrieve your endpoint URL from the service default endpoint and so do not yet need your account-specific endpoint.

The following examples show how to:

- Retrieve your account-specific endpoint, using DescribeEndpoints.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

To access the MediaConvert client, create an IAM role that gives AWS Elemental MediaConvert access to your input files and the Amazon S3 buckets where your output files are stored. For details, see Set Up IAM Permissions in the AWS Elemental MediaConvert User Guide.

Retrieve Endpoints

Create an object to pass the empty request parameters for the describeEndpoints method of the AWS.MediaConvert client class. To call the describeEndpoints method, create a promise for invoking an AWS Elemental MediaConvert service object, passing the parameters. Handle the response in the promise callback.

Imports

```php
require 'vendor/autoload.php';
use Aws\MediaConvert\MediaConvertClient;
use Aws\Exception\AwsException;
```

Sample Code

Define the region in which to get the endpoint, and create a MediaConvert client object:

```php
$client = new Aws\MediaConvert\MediaConvertClient([  'profile' => 'default',  'version' => '2017-08-29',  'region' => 'us-east-2'
```

212
//retrieve endpoint
try {
    $result = $client->describeEndpoints([]);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

Call the describeEndpoints method to retrieve the endpoints and save the endpoint's URL:

```
$single_endpoint_url = $result['Endpoints'][0]['Url'];
print("Your endpoint is " . $single_endpoint_url);
```

//Create an AWSMediaConvert client object with the endpoint URL that you retrieved:
$mediaConvertClient = new MediaConvertClient(
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default',
    'endpoint' => $single_endpoint_url
);

Creating and Managing Transcoding Jobs in AWS Elemental MediaConvert with AWS SDK for PHP Version 3

In this example, you use the AWS SDK for PHP Version 3 to call AWS Elemental MediaConvert and create a transcoding job. Before you begin, you need to upload the input video to the Amazon S3 bucket you provisioned for the input storage. For a list of supported input video codecs and containers, see Supported Input Codecs and Containers in the AWS Elemental MediaConvert User Guide.

The following examples show how to:

- Create transcoding jobs in AWS Elemental MediaConvert. CreateJob.
- Cancel a transcoding job from the AWS Elemental MediaConvert queue. CancelJob
- Retrieve the JSON for a completed transcoding job. GetJob
- Retrieve a JSON array for up to 20 of the most recently created jobs. ListJobs

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

To access the MediaConvert client, create an IAM role that gives AWS Elemental MediaConvert access to your input files and the Amazon S3 buckets where your output files are stored. For details, see Set Up IAM Permissions in the AWS Elemental MediaConvert User Guide.
Create a Client

Configure the AWS SDK for PHP by creating a MediaConvert client, with the region for your code. In this example, the region is set to us-west-2. Because AWS Elemental MediaConvert uses custom endpoints for each account, configure the \texttt{AWS.MediaConvert} client class to use your account-specific endpoint. To do this, set the endpoint parameter to your account-specific endpoint (p. 212).

Imports

```php
require 'vendor/autoload.php';
use Aws\MediaConvert\MediaConvertClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#mediaConvertClient = new MediaConvertClient([  
    'version' => '2017-08-29',  
    'region' => 'us-east-2',  
    'profile' => 'default',  
    'endpoint' => 'ACCOUNT_ENDPOINT'
]);
```

Defining a Simple Transcoding Job

Create the JSON that defines the transcode job parameters.

These parameters are detailed. You can use the AWS Elemental MediaConvert console to generate the JSON job parameters by choosing your job settings in the console, and then choosing \textit{Show job JSON} at the bottom of the \textbf{Job} section. This example shows the JSON for a simple job.

Sample Code

```php
#jobSetting = [
    "OutputGroups" => [
        [
            "Name" => "File Group",
            "OutputGroupSettings" => [
                "Type" => "FILE_GROUP_SETTINGS",
                "FileGroupSettings" => [
                    "Destination" => "s3://OUTPUT_BUCKET_NAME/"
                ],
            ],
            "Outputs" => [
                [
                    "VideoDescription" => [
                        "ScalingBehavior" => "DEFAULT",
                        "TimecodeInsertion" => "DISABLED",
                        "AntiAlias" => "ENABLED",
                        "Sharpness" => 50,
                        "CodecSettings" => [
                            "Codec" => "H_264",
                            "H264Settings" => [
                                "InterlaceMode" => "PROGRESSIVE",
                                "NumberReferenceFrames" => 3,
                                "Syntax" => "DEFAULT",
                                "Softness" => 0,
                                "GopClosedCadence" => 1,
```
"GopSize" => 90,
"Slices" => 1,
"GopBReference" => "DISABLED",
"SlowPal" => "DISABLED",
"SpatialAdaptiveQuantization" => "ENABLED",
"TemporalAdaptiveQuantization" => "ENABLED",
"FlickerAdaptiveQuantization" => "DISABLED",
"EntropyEncoding" => "CABAC",
"Bitrate" => 5000000,
"FramerateControl" => "SPECIFIED",
"RateControlMode" => "CBR",
"CodecProfile" => "MAIN",
"Telecine" => "NONE",
"MinIInterval" => 0,
"AdaptiveQuantization" => "HIGH",
"CodecLevel" => "AUTO",
"FieldEncoding" => "PAFF",
"SceneChangeDetect" => "ENABLED",
"QualityTuningLevel" => "SINGLE_PASS",
"FramerateConversionAlgorithm" => "DUPLICATE_DROP",
"UnregisteredSeiTimecode" => "DISABLED",
"GopSizeUnits" => "FRAMES",
"ParControl" => "SPECIFIED",
"NumberBFramesBetweenReferenceFrames" => 2,
"RepeatPps" => "DISABLED",
"FramerateNumerator" => 30,
"FramerateDenominator" => 1,
"ParNumerator" => 1,
"ParDenominator" => 1
],
"AfdSignaling" => "NONE",
"DropFrameTimecode" => "ENABLED",
"RespondToAfd" => "NONE",
"ColorMetadata" => "INSERT"
],
"AudioDescriptions" => [
[
  "AudioTypeControl" => "FOLLOW_INPUT",
  "CodecSettings" => [
    "Codec" => "AAC",
    "AacSettings" => [
      "AudioDescriptionBroadcasterMix" => "NORMAL",
      "RateControlMode" => "CBR",
      "CodecProfile" => "LC",
      "CodingMode" => "CODING_MODE_2_0",
      "RawFormat" => "NONE",
      "SampleRate" => 48000,
      "Specification" => "MPEG4",
      "Bitrate" => 64000
    ],
    "LanguageCodeControl" => "FOLLOW_INPUT",
    "AudioSourceName" => "Audio Selector 1"
  ],
  "ContainerSettings" => [
    "Container" => "MP4",
    "Mp4Settings" => [
      "CslgAtom" => "INCLUDE",
      "FreeSpaceBox" => "EXCLUDE",
      "MoovPlacement" => "PROGRESSIVE_DOWNLOAD"
    ],
    "NameModifier" => ".1"
  ]
]
Create a Job

After creating the job parameters JSON, call the createJob method by invoking an AWS.MediaConvert service object, and passing the parameters. The ID of the job created is returned in the response data.

Sample Code

```php
try {
    $result = $mediaConvertClient->createJob([
        "Role" => "IAM_ROLE_ARN",
        "Settings" => $jobSetting, //JobSettings structure
        "Queue" => "JOB_QUEUE_ARN",
        "UserMetadata" => [
            "Customer" => "Amazon",
        ],
    ]);}
    // catch (AwsException $e) {
    //     // output error message if fails
    //     echo $e->getMessage();
    //     echo "\n";
    // }
```

Retrieve a Job

With the JobID returned when you called createJob, you can get detailed descriptions of recent jobs in JSON format.
Sample Code

```php
$mediaConvertClient = new MediaConvertClient(
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default',
    'endpoint' => 'ACCOUNT_ENDPOINT'
);
try {
    $result = $mediaConvertClient->getJob(
        'Id' => 'JOB_ID',
    );
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Cancel a Job

With the JobID returned when you called createjob, you can cancel a job while it is still in the queue. You can't cancel jobs that have already started transcoding.

Sample Code

```php
$mediaConvertClient = new MediaConvertClient(
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default',
    'endpoint' => 'ACCOUNT_ENDPOINT'
);
try {
    $result = $mediaConvertClient->cancelJob(
        'Id' => 'JOB_ID', // REQUIRED The Job ID of the job to be cancelled.
    );
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Listing Recent Transcoding Jobs

Create the parameters JSON, including values to specify whether to sort the list in ASCENDING, or DESCENDING order, the ARN of the job queue to check, and the status of jobs to include. This returns up to 20 Jobs. To retrieve the 20 next most recent jobs use the nextToken string returned with result.

Sample Code

```php
$mediaConvertClient = new MediaConvertClient(
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default',
    'endpoint' => 'ACCOUNT_ENDPOINT'
);```
Amazon S3 Examples Using the AWS SDK for PHP Version 3

Amazon Simple Storage Service (Amazon S3) is a web service that provides highly scalable cloud storage. Amazon S3 provides easy to use object storage, with a simple web service interface to store and retrieve any amount of data from anywhere on the web.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Topics

- Creating and Using Amazon S3 Buckets with the AWS SDK for PHP Version 3 (p. 218)
- Managing Amazon S3 Bucket Access Permissions with the AWS SDK for PHP Version 3 (p. 221)
- Configuring Amazon S3 Buckets with the AWS SDK for PHP Version 3 (p. 222)
- Using Amazon S3 Multipart Uploads with AWS SDK for PHP version 3 (p. 224)
- Amazon S3 Pre-Signed POSTs with AWS SDK for PHP Version 3 (p. 231)
- Amazon S3 Pre-Signed URL with AWS SDK for PHP Version 3 (p. 232)
- Using an Amazon S3 Bucket as a Static Web Host with AWS SDK for PHP Version 3 (p. 234)
- Working with Amazon S3 Bucket Policies with the AWS SDK for PHP Version 3 (p. 235)
- Using S3 Access Point ARNs the AWS SDK for PHP Version 3 (p. 237)

Creating and Using Amazon S3 Buckets with the AWS SDK for PHP Version 3

The following examples show how to:

- Return a list of buckets owned by the authenticated sender of the request using ListBuckets.
• Create a new bucket using `CreateBucket`.
• Add an object to a bucket using `PutObject`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guidegetting-started.html) (p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](https://docs.aws.amazon.com/sdk-for-php/v3/developer-guideintroduction.html) (p. 7).

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

**List Buckets**

Create a PHP file with the following code. First create an AWS.S3 client service that specifies the AWS Region and version. Then call the `listBuckets` method, which returns all Amazon S3 buckets owned by the sender of the request as an array of Bucket structures.

**Sample Code**

```php
#$s3Client = new S3Client(
    [
        'profile' => 'default',
        'region' => 'us-west-2',
        'version' => '2006-03-01'
    ]);

// Listing all S3 Bucket
$_buckets = $s3Client->listBuckets();
foreach ($buckets['Buckets'] as $bucket) {
    echo $bucket['Name'] . "\n";
}
```

**Create a Bucket**

Create a PHP file with the following code. First create an AWS.S3 client service that specifies the AWS Region and version. Then call the `createBucket` method with an array as the parameter. The only required field is the key 'Bucket', with a string value for the bucket name to create. However, you can specify the AWS Region with the 'CreateBucketConfiguration' field. If successful, this method returns the 'Location' of the bucket.

**Sample Code**

```php
function createBucket($s3Client, $bucketName)
{
    try {
        $result = $s3Client->createBucket([ 
            'Bucket' => $bucketName,
        ]);;
    }
```
function createTheBucket()
{
    $s3Client = new S3Client(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => '2006-03-01'
    ]);  
    echo createBucket($s3Client, 'my-bucket'); 
}  
  
// Uncomment the following line to run this code in an AWS account.  
// createTheBucket();

**Put an Object in a Bucket**

To add files to your new bucket, create a PHP file with the following code.

In your command line, execute this file and pass in the name of the bucket where you want to upload your file as a string, followed by the full file path to the file to upload.

**Sample Code**

```php
$USAGE = "\n" .  
"To run this example, supply the name of an S3 bucket and a file to \n" .  
"upload to it. \n".  
"Ex: php PutObject.php <bucketname> <filename> \n";

if (count($argv) <= 2) {  
    echo $USAGE;  
    exit();  
}

$bucket = $argv[1];  
$filePath = $argv[2];  
$key = basename($argv[2]);

try {
    //Create a S3Client  
    $s3Client = new S3Client(
        ['profile' => 'default',
         'region' => 'us-west-2',
         'version' => '2006-03-01'
    ]);  
    $result = $s3Client->putObject(
        ['Bucket' => $bucket,
         'Key' => $key,
         'SourceFile' => $filePath,
        ]);  
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```
Managing Amazon S3 Bucket Access Permissions with the AWS SDK for PHP Version 3

Access control lists (ACLs) are one of the resource-based access policy options you can use to manage access to your buckets and objects. You can use ACLs to grant basic read/write permissions to other AWS accounts. To learn more, see Managing Access with ACLs.

The following example shows how to:

- Get the access control policy for a bucket using `GetBucketAcl`.
- Set the permissions on a bucket using ACLs, using `PutBucketAcl`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Get and Set an Access Control List Policy

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
// Create a S3Client
$s3Client = new S3Client(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2006-03-01'
]);

// Gets the access control policy for a bucket
$bucket = 'my-s3-bucket';
try {
    $resp = $s3Client->getBucketAcl(['Bucket' => $bucket]);
    echo "Succeed in retrieving bucket ACL as follows: \n";
    var_dump($resp);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

// Sets the permissions on a bucket using access control lists (ACL).
$params = [
```
'ACL' => 'public-read',

'AccessControlPolicy' => [

  // Information can be retrieved from `getBucketAcl` response

  'Grants' => [

    // ...

    [
      'Grantee' => [
        'DisplayName' => '<string>',
        'EmailAddress' => '<string>',
        'ID' => '<string>',
        'Type' => 'CanonicalUser',
        'URI' => '<string>',
      ],
      'Permission' => 'FULL_CONTROL',
    ],
    // ...
  ],

  'Owner' => [
    'DisplayName' => '<string>',
    'ID' => '<string>',
  ],

  'Bucket' => $bucket,
];

try {
  $resp = $s3Client->putBucketAcl($params);
  echo "Succeed in setting bucket ACL.\n";
} catch (AwsException $e) {
  // Display error message
  echo $e->getMessage();
  echo "\n";

} // ...
Get the CORS Configuration

Create a PHP file with the following code. First create an AWS.S3 client service, then call the `getBucketCors` method and specify the bucket whose CORS configuration you want.

The only parameter required is the name of the selected bucket. If the bucket currently has a CORS configuration, that configuration is returned by Amazon S3 as a CORSRules object.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new S3Client(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2006-03-01']
);
try {
    $result = $client->getBucketCors(
        ['Bucket' => $bucketName, // REQUIRED
         ]);    
    var_dump($result);
    } catch (AwsException $e) {
        // output error message if fails
        error_log($e->getMessage());
    }
```

Set the CORS Configuration

Create a PHP file with the following code. First create an AWS.S3 client service. Then call the `putBucketCors` method and specify the bucket whose CORS configuration to set, and the CORSConfiguration as a CORSRules JSON object.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new S3Client(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2006-03-01']
);  
```
Using Amazon S3 Multipart Uploads with AWS SDK for PHP version 3

With a single PutObject operation, you can upload objects up to 5 GB in size. However, by using the multipart upload methods (for example, CreateMultipartUpload, UploadPart, CompleteMultipartUpload, AbortMultipartUpload), you can upload objects from 5 MB to 5 TB in size.

The following example shows how to:

- Upload an object to Amazon S3, using ObjectUploader.
- Create a multipart upload for an Amazon S3 object using MultipartUploader.
- Copy objects from one Amazon S3 location to another using ObjectCopier.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Object Uploader

If you’re not sure whether PutObject or MultipartUploader is best for the task, use ObjectUploader. ObjectUploader uploads a large file to Amazon S3 using either PutObject or MultipartUploader, depending on what is best based on the payload size.
Sample Code

```php
$s3Client = new S3Client([  
    'profile' => 'default',  
    'region' => 'us-east-2',  
    'version' => '2006-03-01'  
]);

$bucket = 'your-bucket';
$key = 'my-file.zip';

// Using stream instead of file path
$source = fopen('/path/to/large/file.zip', 'rb');

$uploader = new ObjectUploader($s3Client, $bucket, $key, $source);

do {
    try {
        $result = $uploader->upload();
        if ($result['@metadata']['statusCode'] == '200') {
            print('<p>File successfully uploaded to ' . $result['ObjectURL'] . '.</p>');
        }
        print($result);
    } catch (MultipartUploadException $e) {
        rewind($source);
        $uploader = new MultipartUploader($s3Client, $source, [$
            'state' => $e->getState(),
        ]);  
    }
} while (!isset($result));

fclose($source);
```

MultipartUploader

Multipart uploads are designed to improve the upload experience for larger objects. They enable you to upload objects in parts independently, in any order, and in parallel.

Amazon S3 customers are encouraged to use multipart uploads for objects greater than 100 MB.

MultipartUploader Object

The SDK has a special MultipartUploader object that simplifies the multipart upload process.

Imports

```php
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartUploader;
use Aws\Exception\MultipartUploadException;
```
Sample Code

```php
$s3Client = new S3Client([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2006-03-01'  
]);

// Use multipart upload
$source = '/path/to/large/file.zip';
$uploader = new MultipartUploader($s3Client, $source, [  
    'bucket' => 'your-bucket',  
    'key' => 'my-file.zip',  
]);

try {  
    $result = $uploader->upload();  
    echo "Upload complete: \$result[\'ObjectURL\']\n";  
} catch (MultipartUploadException $e) {  
    echo $e->getMessage() . \n"\n";  
}
```

The uploader creates a generator of part data, based on the provided source and configuration, and attempts to upload all parts. If some part uploads fail, the uploader continues to upload later parts until the entire source data is read. Afterwards, the uploader retries to upload the failed parts or throws an exception containing information about the parts that failed to upload.

Customizing a Multipart Upload

You can set custom options on the createMultipartUpload, uploadPart, and completeMultipartUpload operations executed by the multipart uploader via callbacks passed to its constructor.

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartUploader;
use Aws\Exception\MultipartUploadException;
```

Sample Code

```php
// Create an S3Client
$s3Client = new S3Client([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2006-03-01'  
]);

// Customizing a multipart upload
$source = '/path/to/large/file.zip';
$uploader = new MultipartUploader($s3Client, $source, [  
    'bucket' => 'your-bucket',  
    'key' => 'my-file.zip',  
    'before_initiate' => function ($command) {  
        // $command is a createMultipartUpload operation  
    }
```
$command['CacheControl'] = 'max-age=3600';
},
'before_upload' => function ($command) {
  // $command is an UploadPart operation
  $command['RequestPayer'] = 'requester';
},
'before_complete' => function ($command) {
  // $command is a CompleteMultipartUpload operation
  $command['RequestPayer'] = 'requester';
},
]);

**Manual Garbage Collection Between Part Uploads**

If you are hitting the memory limit with large uploads, this may be due to cyclic references generated by the SDK not yet having been collected by the PHP garbage collector when your memory limit was hit. Manually invoking the collection algorithm between operations may allow the cycles to be collected before hitting that limit. The following example invokes the collection algorithm using a callback before each part upload. Note that invoking the garbage collector does come with a performance cost, and optimal usage will depend on your use case and environment.

```php
$uploader = new MultipartUploader($client, $source, [
  'bucket' => 'your-bucket',
  'key' => 'your-key',
  'before_upload' => function ($command) {
    gc_collect_cycles();
  }
]);
```

**Recovering from Errors**

When an error occurs during the multipart upload process, a MultipartUploadException is thrown. This exception provides access to the UploadState object, which contains information about the multipart upload's progress. The UploadState can be used to resume an upload that failed to complete.

**Imports**

```php
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartUploader;
use Aws\Exception\MultipartUploadException;
```

**Sample Code**

```php
// Create an S3Client
$s3Client = new S3Client([
  'profile' => 'default',
  'region' => 'us-west-2',
  'version' => '2006-03-01'
]);

$source = '/path/to/large/file.zip';
$uploader = new MultipartUploader($s3Client, $source, [
  'bucket' => 'your-bucket',
  'key' => 'my-file.zip',
```
Resuming an upload from an `UploadState` attempts to upload parts that are not already uploaded. The state object tracks the missing parts, even if they are not consecutive. The uploader reads or seeks through the provided source file to the byte ranges that belong to the parts that still need to be uploaded.

`UploadState` objects are serializable, so you can also resume an upload in a different process. You can also get the `UploadState` object, even when you're not handling an exception, by calling `$uploader->getState()`.

**Important**

Streams passed in as a source to a `MultipartUploader` are not automatically rewound before uploading. If you're using a stream instead of a file path in a loop similar to the previous example, reset the `$source` variable inside of the catch block.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartUploader;
use Aws\Exception\MultipartUploadException;
```

**Sample Code**

```php
// Create an S3Client
$s3Client = new S3Client(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2006-03-01'])
;

//Using stream instead of file path
$source = fopen('/path/to/large/file.zip', 'rb');
$uploader = new MultipartUploader($s3Client, $source, [
    'bucket' => 'your-bucket',
    'key' => 'my-file.zip',
]);
```
try {
    $result = $uploader->upload();
} catch (MultipartUploadException $e) {
    rewind($source);
    $uploader = new MultipartUploader($s3Client, $source, [
        'state' => $e->getState(),
    ]);}
} while (!$result);

### Aborting a Multipart Upload

A multipart upload can be aborted by retrieving the UploadId contained in the UploadState object and passing it to abortMultipartUpload.

```php
try {
    $result = $uploader->upload();
} catch (MultipartUploadException $e) {
    // State contains the "Bucket", "Key", and "UploadId"
    $params = $e->getState()->getId();
    $result = $s3Client->abortMultipartUpload($params);
}
```

### Asynchronous Multipart Uploads

Calling `upload()` on the MultipartUploader is a blocking request. If you are working in an asynchronous context, you can get a promise (p. 63) for the multipart upload.

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartUploader;
use Aws\Exception\MultipartUploadException;

### Sample Code

```php
// Create an S3Client
$s3Client = new S3Client([  
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2006-03-01'  
]);

$source = '/path/to/large/file.zip';
$uploader = new MultipartUploader($s3Client, $source, [
    'bucket' => 'your-bucket',
    'key' => 'my-file.zip',
]);

$promise = $uploader->promise();
```

### Configuration

The MultipartUploader object constructor accepts the following arguments:
$client

The Aws\Client\ClientInterface object to use for performing the transfers. This should be an instance of Aws\S3\S3Client.

$source

The source data being uploaded. This can be a path or URL (for example, /path/to/file.jpg), a resource handle (for example, fopen('/path/to/file.jpg', 'r')), or an instance of a PSR-7 stream.

$config

An associative array of configuration options for the multipart upload.

The following configuration options are valid:

acl

(string) Access control list (ACL) to set on the object being upload. Objects are private by default.

before_complete

(callable) Callback to invoke before the CompleteMultipartUpload operation. The callback should have a function signature like function (Aws\Command $command) {...}.

before_initiate

(callable) Callback to invoke before the CreateMultipartUpload operation. The callback should have a function signature like function (Aws\Command $command) {...}.

before_upload

(callable) Callback to invoke before any UploadPart operations. The callback should have a function signature like function (Aws\Command $command) {...}.

bucket

(string, required) Name of the bucket to which the object is being uploaded.

concurrency

(int, default: int(5)) Maximum number of concurrent UploadPart operations allowed during the multipart upload.

key

(string, required) Key to use for the object being uploaded.

part_size

(int, default: int(5242880)) Part size, in bytes, to use when doing a multipart upload. This must between 5 MB and 5 GB, inclusive.

state

(Aws\Multipart\UploadState) An object that represents the state of the multipart upload and that is used to resume a previous upload. When this option is provided, the bucket, key, and part_size options are ignored.

Multipart Copies

The AWS SDK for PHP also includes a MultipartCopy object that is used in a similar way to the MultipartUploader, but is designed for copying objects between 5 GB and 5 TB in size within Amazon S3.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\Exception\AwsException;
use Aws\S3\MultipartCopy;
use Aws\Exception\MultipartUploadException;

Sample Code

// Create an S3Client
$s3Client = new S3Client(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2006-03-01' ];
);

//Copy objects within S3
$copier = new MultipartCopy($s3Client, '/bucket/key?versionId=foo', [
    'bucket' => 'your-bucket',
    'key' => 'my-file.zip',
]);

try {
    $result = $copier->copy();
    echo "Copy complete: \$result[\'ObjectURL\']\n";
} catch (MultipartUploadException $e) {
    echo $e->getMessage() . "\n";
}

Amazon S3 Pre-Signed POSTs with AWS SDK for PHP Version 3

Much like pre-signed URLs, pre-signed POSTs enable you to give write access to a user without giving them AWS credentials. Pre-signed POST forms can be created with the help of an instance of Aws\S3\PostObjectV4.

The following examples show how to:

• Get data for an S3 Object POST upload form using PostObjectV4.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Create PostObjectV4

To create an instance of PostObjectV4, you must provide the following:

• instance of Aws\S3\S3Client
• bucket
• associative array of form input fields
• array of policy conditions (see Policy Construction in the Amazon Simple Storage Service Developer Guide)
• expiration time string for the policy (optional, one hour by default).

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new S3Client(
    ['profile' => 'default',
     'version' => 'latest',
     'region' => 'us-west-2',
    ]);  
$bucket = 'mybucket';

// Set some defaults for form input fields
$formInputs = ['acl' => 'public-read'];

// Construct an array of conditions for policy
$options = [
    ['acl' => 'public-read'],
    ['bucket' => $bucket],
    ['starts-with', '$key', 'user/eric/'],
];

// Optional: configure expiration time string
$expires = '+2 hours';

$postObject = new Aws\S3\PostObjectV4(
    $client,
    $bucket,
    $formInputs,
    $options,
    $expires
);

// Get attributes to set on an HTML form, e.g., action, method, enctype
$formAttributes = $postObject->getFormAttributes();

// Get form input fields. This will include anything set as a form input in
// the constructor, the provided JSON policy, your AWS access key ID, and an
// auth signature.
$formInputs = $postObject->getFormInputs();
```

**Amazon S3 Pre-Signed URL with AWS SDK for PHP Version 3**

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the Authorization HTTP header. This is useful for enabling direct third-party
browser access to your private Amazon S3 data, without proxying the request. The idea is to construct a “pre-signed” request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a pre-signed request by specifying an expiration time.

The following examples show how to:

- Create a pre-signed URL to get an S3 object using `createPresignedRequest`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](#). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](#).

**Creating a Pre-Signed Request**

You can get the pre-signed URL to an Amazon S3 object by using the `Aws\S3\S3Client::createPresignedRequest()` method. This method accepts an `Aws\CommandInterface` object and expired timestamp and returns a pre-signed `Psr\Http\Message\RequestInterface` object. You can retrieve the pre-signed URL of the object using the `getUri()` method of the request.

The most common scenario is creating a pre-signed URL to GET an object.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$s3Client = new Aws\S3\S3Client(
    ['profile' => 'default',
    'region' => 'us-east-2',
    'version' => '2006-03-01',
    ]); $cmd = $s3Client->getCommand('GetObject', [
    'Bucket' => 'my-bucket',
    'Key' => 'testKey'
    ]); $request = $s3Client->createPresignedRequest($cmd, '+20 minutes');
```

**Creating a Pre-Signed URL**

You can create pre-signed URLs for any Amazon S3 operation using the `getCommand` method for creating a command object, and then calling the `createPresignedRequest()` method with the command. When ultimately sending the request, be sure to use the same method and the same headers as the returned request.

**Sample Code**

```php
```
//Creating a presigned URL
$cmd = $s3Client->getCommand('GetObject', [
    'Bucket' => 'my-bucket',
    'Key' => 'testKey'
]);

$request = $s3Client->createPresignedRequest($cmd, '+20 minutes');

// Get the actual presigned-url
$presignedUrl = (string)$request->getUri();

## Getting the URL to an Object

If you only need the public URL to an object stored in an Amazon S3 bucket, you can use the `Aws\S3\S3Client::getObjectUrl()` method. This method returns an unsigned URL to the given bucket and key.

### Sample Code

```php
//Getting the URL to an object
$url = $s3Client->getObjectUrl('my-bucket', 'my-key');
```

**Important**
The URL returned by this method is not validated to ensure that the bucket or key exists, nor does this method ensure that the object allows unauthenticated access.

### Using an Amazon S3 Bucket as a Static Web Host with AWS SDK for PHP Version 3

You can host a static website on Amazon S3. To learn more, see [Hosting a Static Website on Amazon S3](#).

The following example shows how to:
- Get the website configuration for a bucket using `GetBucketWebsite`.
- Set the website configuration for a bucket using `PutBucketWebsite`.
- Remove the website configuration from a bucket using `DeleteBucketWebsite`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](#).

### Credentials

Before running the example code, configure your AWS credentials. See [Credentials for the AWS SDK for PHP Version 3](#).

### Get, Set, and Delete the Website Configuration for a Bucket

#### Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```
Sample Code

```php
$s3Client = new S3Client([  'profile' => 'default',  'region' => 'us-west-2',  'version' => '2006-03-01']);

// Retrieving the Bucket Website Configuration
$bucket = 'my-s3-bucket';
try {  
    $resp = $s3Client->getBucketWebsite([  
        'Bucket' => $bucket
    ]);  
    echo "Succeed in retrieving website configuration for bucket: ". $bucket . "\n";
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}

// Setting a Bucket Website Configuration
$params = [  
    'Bucket' => $bucket,  
    'WebsiteConfiguration' => [  
        'ErrorDocument' => [  
            'Key' => 'foo',  
        ],  
        'IndexDocument' => [  
            'Suffix' => 'bar',  
        ]
    ]
];
try {  
    $resp = $s3Client->putBucketWebsite($params);
    echo "Succeed in setting bucket website configuration.\n";
} catch (AwsException $e) {  
    // Display error message  
    echo $e->getMessage();  
    echo "\n";
}

// Deleting a Bucket Website Configuration
try {  
    $resp = $s3Client->deleteBucketWebsite([  
        'Bucket' => $bucket
    ]);  
    echo "Succeed in deleting policy for bucket: ". $bucket . "\n";
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

Working with Amazon S3 Bucket Policies with the AWS SDK for PHP Version 3

You can use a bucket policy to grant permission to your Amazon S3 resources. To learn more, see Using Bucket Policies and User Policies.
The following example shows how to:

- Return the policy for a specified bucket using `GetBucketPolicy`.
- Replace a policy on a bucket using `PutBucketPolicy`.
- Delete a policy from a bucket using `DeleteBucketPolicy`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

## Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

### Get, Delete, and Replace a Policy on a Bucket

#### Imports

```php
require "vendor/autoload.php";
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

#### Sample Code

```php
$s3Client = new S3Client([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2006-03-01'  
]);

$bucket = 'my-s3-bucket';

// Get the policy of a specific bucket
try {  
    $resp = $s3Client->getBucketPolicy([  
        'Bucket' => $bucket  
    ]);  
    echo "Succeed in receiving bucket policy:\n";  
    echo (string) $resp->get('Policy');  
    echo "\n";
} catch (AwsException $e) {  
    // Display error message  
    echo $e->getMessage();  
    echo "\n";
}

// Deletes the policy from the bucket
try {  
    $resp = $s3Client->deleteBucketPolicy([  
        'Bucket' => $bucket  
    ]);  
    echo "Succeed in deleting policy of bucket: " . $bucket . "\n";
} catch (AwsException $e) {  
    // Display error message  
    echo $e->getMessage();  
    echo "\n";
}
Using S3 Access Point ARNs the AWS SDK for PHP Version 3

S3 introduced access points, a new way to interact with S3 buckets. Access Points can have unique policies and configuration applied to them instead of directly to the bucket. The AWS SDK for PHP allows you to use access point ARNs in the bucket field for API operations instead of specifying bucket name explicitly. More details on how S3 access points and ARNs work can be found here. The following examples show how to:

- Use GetObject with an access point ARN to fetch an object from a bucket.
- Use PutObject with an access point ARN to add an object to a bucket.
- Configure the S3 client to use the ARN region instead of the client region.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Imports

```php
require 'vendor/autoload.php';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;
```

Get Object

First create an AWS.S3 client service that specifies the AWS region and version. Then call the getObject method with your key and an S3 access point ARN in the Bucket field, which will fetch the object from the bucket associated with that access point.

Sample Code

```php
$s3 = new S3Client(
    'version' => 'latest',
    'region'  => 'us-west-2',
);```
Put an Object in a Bucket

First create an AWS.S3 client service that specifies the AWS Region and version. Then call the `putObject` method with the desired key, the body or source file, and an S3 access point ARN in the `Bucket` field, which will put the object in the bucket associated with that access point.

**Sample Code**

```php
$s3 = new S3Client(
    'version' => 'latest',
    'region' => 'us-west-2',
);  
$result = $s3->putObject(
    'Key' => 'MyKey',
    'Body' => 'MyBody'
);  
```

Configure the S3 client to use the ARN region instead of the client region

When using an S3 access point ARN in an S3 client operation, by default the client will make sure that the ARN region matches the client region, throwing an exception if it does not. This behavior can be changed to accept the ARN region over the client region by setting the `use_arn_region` configuration option to `true`. By default, the option is set to `false`.

**Sample Code**

```php
$s3 = new S3Client(
    'version' => 'latest',
    'region' => 'us-west-2',
    'use_arn_region' => true
);  
```

The client will also check an environment variable and a config file option, in the following order of priority:

1. The client option `use_arn_region`, as in the above example.
2. The environment variable `AWS_S3_USE_ARN_REGION`

```bash
export AWS_S3_USE_ARN_REGION=true
```

1. The config variable `s3_use_arn_region` in the AWS shared configuration file (by default in `~/.aws/config`).

```yaml
[default]
s3_use_arn_region = true
```
Managing Secrets Using the Secrets Manager API and the AWS SDK for PHP Version 3

AWS Secrets Manager stores and manages shared secrets such as passwords, API keys, and database credentials. With the Secrets Manager service, developers can replace hard-coded credentials in deployed code with an embedded call to Secrets Manager.

Secrets Manager natively supports automatic scheduled credential rotation for Amazon Relational Database Service (Amazon RDS) databases, increasing application security. Secrets Manager can also seamlessly rotate secrets for other databases and third-party services using AWS Lambda to implement service-specific details.

The following examples show how to:

- Create a secret using `CreateSecret`.
- Retrieve a secret using `GetSecretValue`.
- List all of the secrets stored by Secrets Manager using `ListSecrets`.
- Get details about a specified secret using `DescribeSecret`.
- Update a specified secret using `PutSecretValue`.
- Set up a secret rotation using `RotateSecret`.
- Mark a secret for deletion using `DeleteSecret`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

### Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

### Create a Secret in Secrets Manager

To create a secret in Secrets Manager, use the `CreateSecret` operation.

In this example, a user name and password are stored as a JSON string.

#### Imports

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

#### Sample Code

```php
$client = new SecretsManagerClient([  
    'profile' => 'default',  
    'version' => '2017-10-17',  
    'region' => 'us-west-2'  
]);
```
#Retrieve a Secret from Secrets Manager

To retrieve the value of a secret stored in Secrets Manager, use the **GetSecretValue** operation.

In this example, secret is a string that contains the stored value. If called on the secret we created earlier, this sample outputs `[
{
"username": "<<USERNAME>>",
"password": "<<PASSWORD>>"
}
]`. Use `json.loads` to access indexed values.

##Imports

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

##Sample Code

```php
$client = new SecretsManagerClient(
    'profile' => 'default',
    'version' => '2017-10-17',
    'region' => '<<{{MyRegionName}}>>',
);
$secretName = '<<{{MySecretName}}>>';

try {
    $result = $client->getSecretValue(
        'SecretId' => $secretName,
    );
} catch (AwsException $e) {
    // error = $e->getErrorCode();
    if ($error == 'DecryptionFailureException') {
        // Secrets Manager can't decrypt the protected secret text using the provided AWS KMS key.
        // Handle the exception here, and/or rethrow as needed.
        throw $e;
    } else if ($error == 'InternalServiceErrorException') {
        // An error occurred on the server side.
        // Handle the exception here, and/or rethrow as needed.
    }
}```
throw $e;
}
if ($error == 'InvalidParameterException') {
    // You provided an invalid value for a parameter.
    // Handle the exception here, and/or rethrow as needed.
    throw $e;
}
if ($error == 'InvalidRequestException') {
    // You provided a parameter value that is not valid for the current state of the
    // resource.
    // Handle the exception here, and/or rethrow as needed.
    throw $e;
}
if ($error == 'ResourceNotFoundException') {
    // We can't find the resource that you asked for.
    // Handle the exception here, and/or rethrow as needed.
    throw $e;
}
// Decrypts secret using the associated KMS CMK.
// Depending on whether the secret is a string or binary, one of these fields will be
// populated.
if (isset($result['SecretString'])) {
    $secret = $result['SecretString'];
} else {
    $secret = base64_decode($result['SecretBinary']);
}
// Your code goes here;

List Secrets Stored in Secrets Manager

Get a list of all the secrets that are stored by Secrets Manager using the ListSecrets operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new SecretsManagerClient([
    'profile' => 'default',
    'version' => '2017-10-17',
    'region' => 'us-west-2'
]);
try {
    $result = $client->listSecrets([ ];
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
Retrieve Details about a Secret

Stored secrets contain metadata about rotation rules, when it was last accessed or changed, user-created tags, and the Amazon Resource Name (ARN). To get the details of a specified secret stored in Secrets Manager, use the DescribeSecret operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new SecretsManagerClient([  'profile' => 'default',  'version' => '2017-10-17',  'region' => 'us-west-2']);
$secretName = '<<{{MySecretName}}>>';
try {  $result = $client->describeSecret([  'SecretId' => $secretName,  ]);  var_dump($result);  } catch (AwsException $e) {  // output error message if fails  echo $e->getMessage();  echo "\n";  }
```

Update the Secret Value

To store a new encrypted secret value in Secrets Manager, use the PutSecretValue operation.

This creates a new version of the secret. If a version of the secret already exists, add the VersionStages parameter with the value in AWSCURRENT to ensure that the new value is used when retrieving the value.

Imports

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new SecretsManagerClient([  'profile' => 'default',  'version' => '2017-10-17',  'region' => 'us-west-2']
```

---

242
Rotate the Value to an Existing Secret in Secrets Manager

To rotate the value of an existing secret stored in Secrets Manager, use a Lambda rotation function and the `RotateSecret` operation.

Before you begin, create a Lambda function to rotate your secret. The AWS Code Sample Catalog currently contains several Lambda code examples for rotating Amazon RDS database credentials.

**Note**
For more information about rotating secrets, see Rotating Your AWS Secrets Manager Secrets in the AWS Secrets Manager User Guide.

After you set up your Lambda function, configure a new secret rotation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new SecretsManagerClient([ 'profile' => 'default', 'version' => '2017-10-17', 'region' => 'us-west-2' ]); $secretName = '<<{{MySecretName}}>>'; $lambda_ARN = 'arn:aws:lambda:us-west-2:123456789012:function:MyTestDatabaseRotationLambda'; $rules = ['AutomaticallyAfterDays' => 30]; try { $result = $client->rotateSecret([ 'RotationLambdaARN' => $lambda_ARN, 'RotationRules' => $rules, 'SecretId' => $secretName, ]); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

```bash
$secretName = '<<{{MySecretName}}>>';
$secret = '{"username":"<<USERNAME>>","password":"<<PASSWORD>>"}';

try {
    $result = $client->putSecretValue([ 'SecretId' => $secretName, 'SecretString' => $secret, ]); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
When a rotation is configured, you can implement a rotation using the `RotateSecret` operation.

**Imports**

```
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new SecretsManagerClient([  'profile' => 'default',  'version' => '2017-10-17',  'region' => 'us-west-2']);
$secretName = '<<{{MySecretName}}>>';
try {  $result = $client->rotateSecret([  'SecretId' => $secretName,  ]);  var_dump($result);}
} catch (AwsException $e) {  // output error message if fails  echo $e->getMessage();  echo "\n";
}
```

Delete a Secret from Secrets Manager

To remove a specified secret from Secrets Manager, use the `DeleteSecret` operation. To prevent deleting a secret accidentally, a DeletionDate stamp is automatically added to the secret that specifies a window of recovery time in which you can reverse the deletion. If the time isn't specified for the recovery window, the default amount of time is 30 days.

**Imports**

```
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$client = new SecretsManagerClient([  'profile' => 'default',  'version' => '2017-10-17',  'region' => 'us-west-2']);
$secretName = '<<{{MySecretName}}>>';
try {  $result = $client->deleteSecret([  'SecretId' => $secretName,  ]);  var_dump($result);}
} catch (AwsException $e) {  // output error message if fails  echo $e->getMessage();  echo "\n";
}
'version' => '2017-10-17',
'region' => 'us-west-2'
]);

$secretName = '<<{{MySecretName}}>>';

try {
    $result = $client->deleteSecret(
        ['SecretId' => $secretName,
        ]); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

Related Information

The AWS SDK for PHP examples use the following REST operations from the AWS Secrets Manager API Reference:

- CreateSecret
- GetSecretValue
- ListSecrets
- DescribeSecret
- PutSecretValue
- RotateSecret
- DeleteSecret

For more information about using AWS Secrets Manager, see the AWS Secrets Manager User Guide.

Amazon SES Examples Using the AWS SDK for PHP Version 3

Amazon Simple Email Service (Amazon SES) is an email platform that provides an easy, money-saving way for you to send and receive email using your own email addresses and domains. For more information about Amazon SES, see the Amazon SES Developer Guide.

All the example code for the AWS SDK for PHP version 3 is available here on GitHub.

Topics

- Verifying Email Identities Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 246)
- Creating Custom Email Templates Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 250)
- Managing Email Filters Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 254)
- Creating and Managing Email Rules Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 257)
- Monitoring Your Sending Activity Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 263)
- Authorizing Senders Using the Amazon SES API and the AWS SDK for PHP Version 3 (p. 264)
Verifying Email Identities Using the Amazon SES API and the AWS SDK for PHP Version 3

When you first start using your Amazon Simple Email Service (Amazon SES) account, all senders and recipients must be verified in the same AWS Region that you are sending emails to. For more information about sending emails, see Sending Email with Amazon SES.

The following examples show how to:

- Verify an email address using VerifyEmailIdentity.
- Verify an email domain using VerifyDomainIdentity.
- List all email addresses using ListIdentities.
- List all email domains using ListIdentities.
- Remove an email address using DeleteIdentity.
- Remove an email domain using DeleteIdentity.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon SES, see the Amazon SES Developer Guide.

Verifying Email addresses

Amazon SES can send email only from verified email addresses or domains. By verifying an email address, you demonstrate that you’re the owner of that address and want to allow Amazon SES to send email from that address.

When you run the following code example, Amazon SES sends an email to the address you specified. When you (or the recipient of the email) click the link in the email, the address is verified.

To add an email address to your Amazon SES account, use the VerifyEmailIdentity operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2',
    ]);
$email = 'email_address';
try {
```
$result = $SesClient->verifyEmailIdentity([  'EmailAddress' => $email,]);  
var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}

Verify an Email Domain

Amazon SES can send email only from verified email addresses or domains. By verifying a domain, you demonstrate that you're the owner of that domain. When you verify a domain, you allow Amazon SES to send email from any address on that domain.

When you run the following code example, Amazon SES provides you with a verification token. You have to add the token to your domain's DNS configuration. For more information, see Verifying a Domain with Amazon SES in the Amazon Simple Email Service Developer Guide.

To add a sending domain to your Amazon SES account, use the VerifyDomainIdentity operation.

Imports

```
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```
#SesClient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2']);
#domain = 'domain.name';

try {
    $result = $SesClient->verifyDomainIdentity([  'Domain' => $domain,]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

List Email Addresses

To retrieve a list of email addresses submitted in the current AWS Region, regardless of verification status, use the ListIdentities operation.

Imports
Verifying Email Addresses

Sample Code

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;

function verifyEmail($email)
{
    $SesClient = new Aws\Ses\SesClient(
        ['profile' => 'default',
         'version' => '2010-12-01',
         'region' => 'us-east-2']
    );
    try {
        $result = $SesClient->verifyEmailAddress($email);
        var_dump($result);
    } catch (AwsException $e) {
        // output error message if fails
        echo $e->getMessage();
        echo "\n";
    }
}
```

List Email Domains

To retrieve a list of email domains submitted in the current AWS Region, regardless of verification status use the `ListIdentities` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;

function listEmailDomains()
{
    $SesClient = new Aws\Ses\SesClient(
        ['profile' => 'default',
         'version' => '2010-12-01',
         'region' => 'us-east-2']
    );
    try {
        $result = $SesClient->listIdentities(
            ['IdentityType' => 'Domain',
             ]
        );
        var_dump($result);
    } catch (AwsException $e) {
        // output error message if fails
        echo $e->getMessage();
        echo "\n";
    }
}
```
Delete an Email Address

To delete a verified email address from the list of identities, use the `DeleteIdentity` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2']);
$email = 'email_address';
try {
    $result = $SesClient->deleteIdentity([  'Identity' => $email,
    ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Delete an Email Domain

To delete a verified email domain from the list of verified identities, use the `DeleteIdentity` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2']);
$domain = 'domain.name';
try {
    $result = $SesClient->deleteIdentity([  'Identity' => $domain,
    ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
Creating Custom Email Templates Using the Amazon SES API and the AWS SDK for PHP Version 3

Amazon Simple Email Service (Amazon SES) enables you to send emails that are personalized for each recipient by using templates. Templates include a subject line and the text and HTML parts of the email body. The subject and body sections can also contain unique values that are personalized for each recipient.

For more information, see Sending Personalized Email Using the Amazon SES in the Amazon Simple Email Service Developer Guide.

The following examples show how to:

- Create an email template using `CreateTemplate`.
- List all email templates using `ListTemplates`.
- Retrieve an email template using `GetTemplate`.
- Update an email template using `UpdateTemplate`.
- Remove an email template using `DeleteTemplate`.
- Send a templated email using `SendTemplatedEmail`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon SES, see the Amazon SES Developer Guide.

**Create an Email Template**

To create a template to send personalized email messages, use the `CreateTemplate` operation. The template can be used by any account authorized to send messages in the AWS Region to which the template is added.

**Note**
Amazon SES doesn't validate your HTML, so be sure that `HtmlPart` is valid before sending an email.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
```
Get an Email Template

To view the content for an existing email template including the subject line, HTML body, and plain text, use the GetTemplate operation. Only TemplateName is required.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2'  ]);
$name = 'Template_Name';
$subject = 'Amazon SES test (AWS SDK for PHP)';
try {  $result = $SesClient->getTemplate([    'TemplateName' => $name,    'subject' => $subject,  ]);  var_dump($result);} catch (AwsException $e) {
    // output error message if fails    echo $e->getMessage();    echo "\n";}
```
List All Email Templates

To retrieve a list of all email templates that are associated with your AWS account in the current AWS Region, use the `ListTemplates` operation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new Aws\Ses\SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-2'  
]);

try {  
    $result = $SesClient->listTemplates([  
        'MaxItems' => 10,  
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

Update an Email Template

To change the content for a specific email template including the subject line, HTML body, and plain text, use the `UpdateTemplate` operation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new Aws\Ses\SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-2'  
]);

try {  
    $result = $SesClient->listTemplates([  
        'MaxItems' => 10,  
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```
$sesclient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2']);
$name = 'Template_Name';
$html_body = '<h1>AWS Amazon Simple Email Service Test Email</h1>
<p>This email was sent with <a href="https://aws.amazon.com/ses/">Amazon SES</a> using the <a href="https://aws.amazon.com/sdk-for-php/">AWS SDK for PHP</a>.</p>';
$subject = 'Amazon SES test (AWS SDK for PHP)';
plaintext_body = 'This email was sent with Amazon SES using the AWS SDK for PHP.';
try {  $result = $sesclient->updateTemplate([  'Template' => [  'HtmlPart' => $html_body,  'SubjectPart' => $subject,  'TemplateName' => $name,  'TextPart' => $plaintext_body,  ],  ]);  var_dump($result);}
catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";}

Delete an Email Template

To remove a specific email template, use the DeleteTemplate operation. All you need is the TemplateName.

Imports

require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;

Sample Code

$name = 'Template_Name';
try {  $result = $sesclient->deleteTemplate([  'TemplateName' => $name,  ]);  var_dump($result);}
catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
Send an Email with a Template

To use a template to send an email to recipients, use the `SendTemplatedEmail` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']);
$template_name = 'Template_Name';
$sender_email = 'email_address';
$recipient_emails = ['email_address'];

try {
    $result = $SesClient->sendTemplatedEmail(
        ['Destination' => [
            'ToAddresses' => $verified_recipient_emails,
        ],
        'ReplyToAddresses' => [$sender_email],
        'Source' => $sender_email,
        'Template' => $template_name,
        'TemplateData' => '{ }'
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";  
}
```

Managing Email Filters Using the Amazon SES API and the AWS SDK for PHP Version 3

In addition to sending emails, you can also receive email with Amazon Simple Email Service (Amazon SES). An IP address filter enables you to optionally specify whether to accept or reject mail that originates from an IP address or range of IP addresses. For more information, see Managing IP Address Filters for Amazon SES Email Receiving.
The following examples show how to:

- Create an email filter using `CreateReceiptFilter`.
- List all email filters using `ListReceiptFilters`.
- Remove an email filter using `DeleteReceiptFilter`.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

For more information about using Amazon SES, see the [Amazon SES Developer Guide](https://docs.aws.amazon.com/SES/latest/DeveloperGuide/).

**Create an Email Filter**

To allow or block emails from a specific IP address, use the `CreateReceiptFilter` operation. Provide the IP address or range of addresses and a unique name to identify this filter.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-2'
]);
$filter_name = 'FilterName';
$ip_address_range = '10.0.0.1/24';
try {
    $result = $SesClient->createReceiptFilter(
        ['Filter' => [
            'IpFilter' => [
                'Cidr' => $ip_address_range,
                'Policy' => 'Block|Allow',
            ],
            'Name' => $filter_name,
        ]],
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```
List All Email Filters

To list the IP address filters associated with your AWS account in the current AWS Region, use the `ListReceiptFilters` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);
try {
    $result = $SesClient->listReceiptFilters();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Delete an Email Filter

To remove an existing filter for a specific IP address use the `DeleteReceiptFilter` operation. Provide the unique filter name to identify the receipt filter to delete.

If you need to change the range of addresses that are filtered, you can delete a receipt filter and create a new one.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);
$filter_name = 'FilterName';
try {
    #filter_name = 'FilterName';
    try {
```
Creating and Managing Email Rules Using the Amazon SES API and the AWS SDK for PHP Version 3

In addition to sending emails, you can also receive email with Amazon Simple Email Service (Amazon SES). Receipt rules enable you to specify what Amazon SES does with email it receives for the email addresses or domains you own. A rule can send email to other AWS services including but not limited to Amazon S3, Amazon SNS, or AWS Lambda.

For more information, see Managing Receipt Rule Sets for Amazon SES Email Receiving and Managing Receipt Rules for Amazon SES Email Receiving.

The following examples show how to:

- Create a receipt rule set using `CreateReceiptRuleSet`.
- Create a receipt rule using `CreateReceiptRule`.
- Describe a receipt rule set using `DescribeReceiptRuleSet`.
- Describe a receipt rule using `DescribeReceiptRule`.
- List all receipt rule sets using `ListReceiptRuleSets`.
- Update a receipt rule using `UpdateReceiptRule`.
- Remove a receipt rule using `DeleteReceiptRule`.
- Remove a receipt rule set using `DeleteReceiptRuleSet`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon SES, see the Amazon SES Developer Guide.

**Create a Receipt Rule Set**

A receipt rule set contains a collection of receipt rules. You must have at least one receipt rule set associated with your account before you can create a receipt rule. To create a receipt rule set, provide a unique `RuleSetName` and use the `CreateReceiptRuleSet` operation.

**Imports**

```php
require 'vendor/autoload.php';
```
Create a Receipt Rule

Control your incoming email by adding a receipt rule to an existing receipt rule set. This example shows you how to create a receipt rule that sends incoming messages to an Amazon S3 bucket, but you can also send messages to Amazon SNS and AWS Lambda. To create a receipt rule, provide a rule and the RuleSetName to the CreateReceiptRule operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SesClient = new Aws\Ses\SesClient([ 'profile' => 'default', 'version' => '2010-12-01', 'region' => 'us-east-2 ' ]);
$name = 'Rule_Set_Name';
try {
    $result = $SesClient->createReceiptRuleSet([ 'RuleSetName' => $name, ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Create a Receipt Rule

Control your incoming email by adding a receipt rule to an existing receipt rule set. This example shows you how to create a receipt rule that sends incoming messages to an Amazon S3 bucket, but you can also send messages to Amazon SNS and AWS Lambda. To create a receipt rule, provide a rule and the RuleSetName to the CreateReceiptRule operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SesClient = new Aws\Ses\SesClient([ 'profile' => 'default', 'version' => '2010-12-01', 'region' => 'us-east-2 ' ]);  
$rule_name = 'Rule_Name';
$rule_set_name = 'Rule_Set_Name';
$s3_bucket = 'Bucket_Name';
try {
    $result = $SesClient->createReceiptRule([ 'Rule' => [ 'Actions' => [ [ 'S3Action' => [ 'BucketName' => $s3_bucket, 
```
Describe a Receipt Rule Set

Once per second, return the details of the specified receipt rule set. To use the DescribeReceiptRuleSet operation, provide the RuleSetName.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-2']);
$name = 'Rule_Set_Name';
try {
    $result = $SesClient->describeReceiptRuleSet([  'RuleSetName' => $name,  ]);}
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Describe a Receipt Rule

Return the details of a specified receipt rule. To use the DescribeReceiptRule operation, provide the RuleName and RuleSetName.

Imports
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);
$rule_name = 'Rule_Name';
$rule_set_name = 'Rule_Set_Name';

try {
    $result = $SesClient->describeReceiptRule(
        ['RuleName' => $rule_name,
         'RuleSetName' => $rule_set_name,
        ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

List All Receipt Rule Sets

To list the receipt rule sets that exist under your AWS account in the current AWS Region, use the `ListReceiptRuleSets` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);

try {
    $result = $SesClient->listReceiptRuleSets(
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
Update a Receipt Rule

This example shows you how to update a receipt rule that sends incoming messages to an AWS Lambda function, but you can also send messages to Amazon SNS and Amazon S3. To use the `UpdateReceiptRule` operation, provide the new receipt rule and the `RuleSetName`.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2',
    ]);

$rule_name = 'Rule_Name';
$rule_set_name = 'Rule_Set_Name';
$lambda_arn = 'Amazon Resource Name (ARN) of the AWS Lambda function';
$sns_topic_arn = 'Amazon Resource Name (ARN) of the Amazon SNS topic';

try {
    $result = $SesClient->updateReceiptRule([ 
        'Rule' => [ 
            'Actions' => [ 
                'LambdaAction' => [ 
                    'FunctionArn' => $lambda_arn, 
                    'TopicArn' => $sns_topic_arn, 
                ],
            ],
            'Enabled' => true,
            'Name' => $rule_name,
            'ScanEnabled' => false,
            'TlsPolicy' => 'Require',
        ],
        'RuleSetName' => $rule_set_name,
    ]); 
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Delete a Receipt Rule Set

Remove a specified receipt rule set that isn't currently disabled. This also deletes all of the receipt rules it contains. To delete a receipt rule set, provide the `RuleSetName` to the `DeleteReceiptRuleSet` operation.
Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);

$name = 'Rule_Set_Name';

try {
    $result = $SesClient->deleteReceiptRuleSet(
        ['RuleSetName' => $name],
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Delete a Receipt Rule

To delete a specified receipt rule, provide the RuleName and RuleSetName to the `DeleteReceiptRule` operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new Aws\Ses\SesClient(
    ['profile' => 'default',
     'version' => '2010-12-01',
     'region' => 'us-east-2']
);

$rule_name = 'Rule_Name';
$rule_set_name = 'Rule_Set_Name';

try {
    $result = $SesClient->deleteReceiptRule(
        ['RuleName' => $rule_name,
         'RuleSetName' => $rule_set_name,
        ]);
```
Monitor Your Sending Activity Using the Amazon SES API and the AWS SDK for PHP Version 3

Amazon Simple Email Service (Amazon SES) provides methods for monitoring your sending activity. We recommend that you implement these methods so that you can keep track of important measures, such as your account's bounce, complaint, and reject rates. Excessively high bounce and complaint rates can jeopardize your ability to send emails using Amazon SES.

The following examples show how to:

- Check your sending quota using GetSendQuota.
- Monitor your sending activity using GetSendStatistics.

All the example code for the AWS SDK for PHP Version 3 is available [here on GitHub](https://github.com/aws/aws-sdk-php).

**Credentials**

Before running the example code, configure your AWS credentials, as described in [Credentials for the AWS SDK for PHP Version 3](p. 42). Then import the AWS SDK for PHP, as described in [Basic Usage Patterns of the AWS SDK for PHP Version 3](p. 7).

For more information about using Amazon SES, see the [Amazon SES Developer Guide](https://docs.aws.amazon.com/ses/latest/DeveloperGuide/).

**Check Your Sending Quota**

You are limited to sending only a certain amount of messages in a single 24-hour period. To check how many messages you are still allowed to send, use the GetSendQuota operation. For more information, see [Managing Your Amazon SES Sending Limits](https://docs.aws.amazon.com/ses/latest/DeveloperGuide/).

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new SesClient(
    ['profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-1'
    ]);

try {
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
Monitor Your Sending Activity

To retrieve metrics for messages you've sent in the past two weeks, use the GetSendStatistics operation. This example returns the number of delivery attempts, bounces, complaints, and rejected messages in 15-minute increments.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SesClient = new SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-1'
]);

try {  
    $result = $SesClient->getSendStatistics([  
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

Authorizing Senders Using the Amazon SES API and the AWS SDK for PHP Version 3

To enable another AWS account, AWS Identity and Access Management user, or AWS service to send email through Amazon Simple Email Service (Amazon SES) on your behalf, you create a sending authorization policy. This is a JSON document that you attach to an identity that you own.

The policy expressly lists who you are allowing to send for that identity, and under which conditions. All senders, other than you and the entities you explicitly grant permissions to in the policy, are not allowed
to send emails. An identity can have no policy, one policy, or multiple policies attached to it. You can also have one policy with multiple statements to achieve the effect of multiple policies.

For more information, see Using Sending Authorization with Amazon SES.

The following examples show how to:

- Create an authorized sender using PutIdentityPolicy.
- Retrieve polices for an authorized sender using GetIdentityPolicies.
- List authorized senders using ListIdentityPolicies.
- Revoke permission for an authorized sender using DeleteIdentityPolicy.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

For more information about using Amazon SES, see the Amazon SES Developer Guide.

Create an Authorized Sender

To authorize another AWS account to send emails on your behalf, use an identity policy to add or update authorization to send emails from your verified email addresses or domains. To create an identity policy, use the PutIdentityPolicy operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SesClient = new SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-1'  
]);

#identity = "arn:aws:ses:us-east-1:123456789012:identity/example.com";
#other_aws_account = "0123456789";
#policy = <<<EOT
  {
    "Id":"ExampleAuthorizationPolicy",
    "Version":"2012-10-17",
    "Statement":[
      {
        "Sid":"AuthorizeAccount",
        "Effect":"Allow",
        "Resource":="#identity",
        "Principal":{
          "AWS":[ "#other_aws_account" ]
        },
        "Action":[
```
Retrieve Polices for an Authorized Sender

Return the sending authorization policies that are associated with a specific email identity or domain identity. To get the sending authorization for a given email address or domain, use the \GetIdentityPolicy\ operation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
#SesClient = new SesClient([  'profile' => 'default',  'version' => '2010-12-01',  'region' => 'us-east-1']);

#identity = "arn:aws:ses:us-east-1:123456789012:identity/example.com";
#policies = ["policyName"];

try {
    $result = $SesClient->getIdentityPolicies([  'Identity' => $identity,  'PolicyNames' => $policies,  ]);  var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```
List Authorized Senders

To list the sending authorization policies that are associated with a specific email identity or domain identity in the current AWS Region, use the `ListIdentityPolicies` operation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-1'  
]);

$identity = "arn:aws:ses:us-east-1:123456789012:identity/example.com";

try {  
    $result = $SesClient->listIdentityPolicies([  
        'Identity' => $identity,  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    echo $e->getMessage();  
    echo "\n";
}
```

Revoke Permission for an Authorized Sender

Remove sending authorization for another AWS account to send emails with an email identity or domain identity by deleting the associated identity policy with the `DeleteIdentityPolicy` operation.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Ses\SesClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SesClient = new SesClient([  
    'profile' => 'default',  
    'version' => '2010-12-01',  
    'region' => 'us-east-1'  
]);
```
Amazon SNS Examples Using the AWS SDK for PHP Version 3

Amazon Simple Notification Service (Amazon SNS) is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients.

In Amazon SNS, there are two types of clients: publishers (also referred to as producers) and subscribers (also referred to as consumers). Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel. Subscribers (web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (Amazon SQS, HTTP/HTTPS URLs, email, AWS SMS, Lambda) when they are subscribed to the topic.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Topics
- Managing Topics in Amazon SNS with the AWS SDK for PHP Version 3 (p. 268)
- Managing Subscriptions in Amazon SNS with AWS SDK for PHP Version 3 (p. 272)
- Sending SMS Messages in Amazon SNS with the AWS SDK for PHP Version 3 (p. 277)

Managing Topics in Amazon SNS with the AWS SDK for PHP Version 3

To send notifications to Amazon Simple Queue Service (Amazon SQS), HTTP/HTTPS URLs, email, AWS SMS, or AWS Lambda, you must first create a topic that manages the delivery of messages to any subscribers of that topic.

In terms of the observer design pattern, a topic is like the subject. After a topic is created, you add subscribers that are notified automatically when a message is published to the topic.

Learn more about subscribing to topics in Managing Subscriptions in Amazon SNS with AWS SDK for PHP Version 3 (p. 272).

The following examples show how to:
- Create a topic to publish notifications to using CreateTopic.
• Return a list of the requester's topics using ListTopics.
• Delete a topic and all of its subscriptions using DeleteTopic.
• Return all of the properties of a topic using GetTopicAttributes.
• Allow a topic owner to set an attribute of the topic to a new value using SetTopicAttributes.

For more information about using Amazon SNS, see Amazon SNS Topic Attributes for Message Delivery Status.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Create a Topic

To create a topic, use the CreateTopic operation.

Each topic name in your AWS account must be unique.

Imports

```
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$Snsclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topicname = 'myTopic';

try {
    $result = $Snsclient->createTopic([
        'Name' => $topicname,
    ]); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

List Your Topics

To list up to 100 existing topics in the current AWS Region, use the ListTopics operation.

Imports

```
require 'vendor/autoload.php';
```
Managing Topics

Sample Code

```php
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

# $SnsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);

try {
    $result = $SnsClient->listTopics([  
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

Delete a Topic

To remove an existing topic and all of its subscriptions, use the `DeleteTopic` operation.

Any messages that have not been delivered yet to subscribers will also be deleted.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
# $SnsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnsClient->deleteTopic([  
        'TopicArn' => $topic,
    ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

Get Topic Attributes

To retrieve properties of a single existing topic, use the `GetTopicAttributes` operation.

Imports
Sample Code

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

Sample Code

```php
#SnsClient = new SnsClient(
 'profile' => 'default',
 'region' => 'us-east-1',
 'version' => '2010-03-31'
));
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnsClient->getTopicAttributes(
        'TopicArn' => $topic,
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Set Topic Attributes

To update properties of a single existing topic, use the `SetTopicAttributes` operation.

You can set only the Policy, DisplayName, and DeliveryPolicy attributes.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SnsClient = new SnsClient(
 'profile' => 'default',
 'region' => 'us-east-1',
 'version' => '2010-03-31'
));
$attribute = 'Policy | DisplayName | DeliveryPolicy';
$value = 'First Topic';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnsClient->setTopicAttributes(
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn' => $topic,
    );
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
Managing Subscriptions in Amazon SNS with AWS SDK for PHP Version 3

Use Amazon Simple Notification Service (Amazon SNS) topics to send notifications to Amazon Simple Queue Service (Amazon SQS), HTTP/HTTPS, email addresses, AWS Server Migration Service (AWS SMS), or AWS Lambda.

Subscriptions are attached to a topic that manages sending messages to subscribers. Learn more about creating topics in Managing Topics in Amazon SNS with the AWS SDK for PHP Version 3 (p. 268).

The following examples show how to:

- Subscribe to an existing topic using Subscribe.
- Verify a subscription using ConfirmSubscription.
- List existing subscriptions using ListSubscriptionsByTopic.
- Delete a subscription using Unsubscribe.
- Send a message to all subscribers of a topic using Publish.

For more information about using Amazon SNS, see Using Amazon SNS for System-to-System Messaging.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Subscribe an Email Address to a Topic

To initiate a subscription to an email address, use the Subscribe operation.

You can use the subscribe method to subscribe several different endpoints to an Amazon SNS topic, depending on the values used for parameters passed. This is shown in other examples in this topic.

In this example, the endpoint is an email address. A confirmation token is sent to this email. Verify the subscription with this confirmation token within three days of receipt.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
```
Subscribe an Application Endpoint to a Topic

To initiate a subscription to a web app, use the Subscribe operation.

You can use the subscribe method to subscribe several different endpoints to an Amazon SNS topic, depending on the values used for parameters passed. This is shown in other examples in this topic.

In this example, the endpoint is a URL. A confirmation token is sent to this web address. Verify the subscription with this confirmation token within three days of receipt.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SnsClient = new SnsClient(
  ['profile' => 'default',
   'region' => 'us-east-1',
   'version' => '2010-03-31']);

$protocol = 'https';
$endpoint = 'https://';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
  $result = $SnsClient->subscribe(
    ['Protocol' => $protocol,
     'Endpoint' => $endpoint,
     'ReturnSubscriptionArn' => true,
     'TopicArn' => $topic,
   ]);    
  var_dump($result);
} catch (AwsException $e) {
  // output error message if fails
  error_log($e->getMessage());
}
```

Subscribe a Lambda Function to a Topic

To initiate a subscription to a Lambda function, use the Subscribe operation.
You can use the subscribe method to subscribe several different endpoints to an Amazon SNS topic, depending on the values used for parameters passed. This is shown in other examples in this topic.

In this example, the endpoint is a Lambda function. A confirmation token is sent to this Lambda function. Verify the subscription with this confirmation token within three days of receipt.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SnsClient = new SnsClient([  'profile' => 'default',  'region' => 'us-east-1',  'version' => '2010-03-31']);
$protocol = 'lambda';
$endpoint = 'arn:aws:lambda:us-east-1:123456789023:function:messageStore';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnsClient->subscribe([  'Protocol' => $protocol,  'Endpoint' => $endpoint,  'ReturnSubscriptionArn' => true,  'TopicArn' => $topic,]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Subscribe a Text SMS to a Topic

To send SMS messages to multiple phone numbers at the same time, subscribe each number to a topic.

To initiate a subscription to a phone number, use the Subscribe operation.

You can use the subscribe method to subscribe several different endpoints to an Amazon SNS topic, depending on the values used for parameters passed. This is shown in other examples in this topic.

In this example, the endpoint is a phone number in E.164 format, a standard for international telecommunications.

A confirmation token is sent to this phone number. Verify the subscription with this confirmation token within three days of receipt.

For an alternative way to send SMS messages with Amazon SNS, see Sending SMS Messages in Amazon SNS with the AWS SDK for PHP Version 3 (p. 277).

Imports

```php
require 'vendor/autoload.php';
```
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

### Sample Code

```php
# $SnSclient = new SnsClient([  
'profile' => 'default',  
'region' => 'us-east-1',  
'version' => '2010-03-31'  
]);

# $protocol = 'sms';  
# $endpoint = '+1XXX5550100';  
# $topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {  
    $result = $SnSclient->subscribe([  
        'Protocol' => $protocol,  
        'Endpoint' => $endpoint,  
        'ReturnSubscriptionArn' => true,  
        'TopicArn' => $topic,  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

### Confirm Subscription to a Topic

To actually create a subscription, the endpoint owner must acknowledge intent to receive messages from the topic using a token sent when a subscription is established initially, as described earlier. Confirmation tokens are valid for three days. After three days, you can resend a token by creating a new subscription.

To confirm a subscription, use the `ConfirmSubscription` operation.

### Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;  
use Aws\Exception\AwsException;
```

### Sample Code

```php
# $SnSclient = new SnsClient([  
'profile' => 'default',  
'region' => 'us-east-1',  
'version' => '2010-03-31'  
]);

# $subscription_token = 'arn:aws:sns:us-east-1:111122223333:MyTopic:123456-abcd-12ab-1234-12ba3dc1234a';  
# $topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {  
    $result = $SnSclient->subscribe([  
        'Token' => $subscription_token,  
        'TopicArn' => $topic,  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```
List Subscriptions to a Topic

To list up to 100 existing subscriptions in a given AWS Region, use the ListSubscriptions operation.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SnSclient = new SnsClient([ 'profile' => 'default', 'region' => 'us-east-1', 'version' => '2010-03-31']);
try {
    $result = $SnSclient->listSubscriptions();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Unsubscribe from a Topic

To remove an endpoint subscribed to a topic, use the Unsubscribe operation.

If the subscription requires authentication for deletion, only the owner of the subscription or the topic's owner can unsubscribe, and an AWS signature is required. If the unsubscribe call doesn't require authentication and the requester isn't the subscription owner, a final cancellation message is delivered to the endpoint.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SnSclient = new SnsClient([ 'profile' => 'default', 'region' => 'us-east-1', 'version' => '2010-03-31']);
try {
    $result = $SnSclient->listSubscriptions();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $Snsclient->unsubscribe(['SubscriptionArn' => $subscription, ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

Publish a Message to an Amazon SNS Topic

To deliver a message to each endpoint that's subscribed to an Amazon SNS topic, use the Publish operation.

Create an object that contains the parameters for publishing a message, including the message text and the Amazon Resource Name (ARN) of the Amazon SNS topic.

Sample Code

```php
$Snsclient = new SnsClient(['profile' => 'default', 'region' => 'us-east-1', 'version' => '2010-03-31']);

$message = 'This message is sent from an Amazon SNS code sample.';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $Snsclient->publish(['Message' => $message, 'TopicArn' => $topic, ]);  
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Sending SMS Messages in Amazon SNS with the AWS SDK for PHP Version 3

You can use Amazon Simple Notification Service (Amazon SNS) to send text messages, or SMS messages, to SMS-enabled devices. You can send a message directly to a phone number, or you can send a message to multiple phone numbers at once by subscribing those phone numbers to a topic and sending your message to the topic.
Use Amazon SNS to specify preferences for SMS messaging, such as how your deliveries are optimized (for cost or for reliable delivery), your monthly spending limit, how message deliveries are logged, and whether to subscribe to daily SMS usage reports. These preferences are retrieved and set as SMS attributes for Amazon SNS.

When you send an SMS message, specify the phone number using the E.164 format. E.164 is a standard for the phone number structure used for international telecommunications. Phone numbers that follow this format can have a maximum of 15 digits, and are prefixed with the plus character (+) and the country code. For example, a US phone number in E.164 format would appear as +1001XXX5550100.

The following examples show how to:

- Retrieve the default settings for sending SMS messages from your account using `GetSMSAttributes`.
- Update the default settings for sending SMS messages from your account using `SetSMSAttributes`.
- Discover if a given phone number owner has opted out of receiving SMS messages from your account using `CheckIfPhoneNumberISOptedOut`.
- List phone numbers where the owner has opted out of receiving SMS messages from your account using `ListPhoneNumberOptedOut`.
- Send a text message (SMS message) directly to a phone number using `Publish`.

For more information about using Amazon SNS, see Using Amazon SNS for User Notifications with a Mobile Phone Number as a Subscriber (Send SMS).

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

**Get SMS Attributes**

To retrieve the default settings for SMS messages, use the `GetSMSAttributes` operation.

This example gets the `DefaultSMSMessageType` attribute. This attribute controls whether SMS messages are sent as Promotional, which optimizes message delivery to incur the lowest cost, or as Transactional, which optimizes message delivery to achieve the highest reliability.

**Imports**

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

**Sample Code**

```php
$SnSclient = new SnsClient([  'profile' => 'default',  'region' => 'us-east-1',  'version' => '2010-03-31']);
try {  $result = $SnSclient->getSMSAttributes([  ];```
Set SMS Attributes

To update the default settings for SMS messages, use the `SetSMSAttributes` operation.

This example sets the `DefaultSMSType` attribute to `Transactional`, which optimizes message delivery to achieve the highest reliability.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$SnSClient = new SnsClient(['profile' => 'default',
                           'region' => 'us-east-1',
                           'version' => '2010-03-31']);
try {
    $result = $SnSClient->SetSMSAttributes(['attributes' => [
        'DefaultSMSType' => 'Transactional',
    ]]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Check If a Phone Number Has Opted Out

To determine if a given phone number owner has opted out of receiving SMS messages from your account, use the `CheckIfPhoneNumberIsOptedOut` operation.

In this example, the phone number is in E.164 format, a standard for international telecommunications.

Imports

```php
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
#SnsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);
try {
    $result = $SnsClient->SetSMSAttributes([  
        'attributes' => [  
            'DefaultSMSType' => 'Transactional',  
        ],  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
# $SnSclient = new SnsClient([  'profile' => 'default',  'region' => 'us-east-1',  'version' => '2010-03-31']);  

# phone = '+1XXX5550100';

try {  
    $result = $SnSclient->checkIfPhoneNumberIsOptedOut([  'phoneNumber' => $phone,  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}

## List Opted-Out Phone Numbers

To retrieve a list of phone numbers where the owner has opted out of receiving SMS messages from your account, use the ListPhoneNumbersOptedOut operation.

### Imports

```
require 'vendor/autoload.php';
use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;
```

### Sample Code

```
$SnSclient = new SnsClient([  'profile' => 'default',  'region' => 'us-east-1',  'version' => '2010-03-31']);

try {  
    $result = $SnSclient->listPhoneNumbersOptedOut([  ]);  
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

## Publish to a Text Message (SMS Message)

To deliver a text message (SMS message) directly to a phone number, use the Publish operation.

In this example, the phone number is in E.164 format, a standard for international telecommunications. SMS messages can contain up to 140 bytes. The size limit for a single SMS publish action is 1,600 bytes.

For more details on sending SMS messages, see Sending an SMS Message.

### Imports
require 'vendor/autoload.php';

use Aws\Sns\SnsClient;
use Aws\Exception\AwsException;

Sample Code

```php
$SnsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';

try {  
    $result = $SnsClient->publish([  
        'Message' => $message,  
        'PhoneNumber' => $phone,  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

Amazon SQS Examples Using the AWS SDK for PHP Version 3

Amazon Simple Queue Service (SQS) is a fast, reliable, scalable, fully managed message queuing service. Amazon SQS lets you decouple the components of a cloud application. Amazon SQS includes standard queues with high throughput and at-least-once processing, and FIFO queues that provide FIFO (first-in, first-out) delivery and exactly-once processing.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Topics**

- Enabling Long Polling in Amazon SQS with AWS SDK for PHP Version 3 (p. 281)
- Managing Visibility Timeout in Amazon SQS with AWS SDK for PHP Version 3 (p. 284)
- Sending and Receiving Messages in Amazon SQS with AWS SDK for PHP Version 3 (p. 285)
- Using Dead-Letter Queues in Amazon SQS with AWS SDK for PHP Version 3 (p. 287)
- Using Queues in Amazon SQS with AWS SDK for PHP version 3 (p. 288)

**Enabling Long Polling in Amazon SQS with AWS SDK for PHP Version 3**

Long polling reduces the number of empty responses by allowing Amazon SQS to wait a specified time for a message to become available in the queue before sending a response. Also, long polling eliminates false empty responses by querying all of the servers instead of a sampling of servers. To enable long polling, specify a non-zero wait time for received messages. To learn more, see SQS Long Polling.
The following examples show how to:

- Set attributes on an Amazon SQS queue to enable long polling, using `SetQueueAttributes`.
- Retrieve one or more messages with long polling using `ReceiveMessage`.
- Create a long polling queue using `CreateQueue`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

**Credentials**

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

**Set Attributes on a Queue to Enable Long Polling**

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueUrl = "QUEUE_URL";

$client = new SqsClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2012-11-05'  
]);

try {  
    $result = $client->setQueueAttributes(array(  
        'Attributes' => [  
            'ReceiveMessageWaitTimeSeconds' => 20  
        ],  
        'QueueUrl' => $queueUrl, // REQUIRED  
    ));
    var_dump($result);
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());
}
```

**Retrieve Messages with Long Polling**

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```
Sample Code

```php
$queueUrl = "QUEUE_URL";

$client = new SqsClient([ 'profile' => 'default', 'region' => 'us-west-2', 'version' => '2012-11-05' ]); try { $result = $client->receiveMessage(array( 'AttributeNames' => ['SentTimestamp'], 'MaxNumberOfMessages' => 1, 'MessageAttributeNames' => ['All'], 'QueueUrl' => $queueUrl, // REQUIRED 'WaitTimeSeconds' => 20, )); var_dump($result); } catch (AwsException $e) { // output error message if fails error_log($e->getMessage()); }
```

Create a Queue with Long Polling

Imports

```php
require 'vendor/autoload.php'; use Aws\Sqs\SqsClient; use Aws\Exception\AwsException;
```

Sample Code

```php
$queueName = "QUEUE_NAME";

$client = new SqsClient([ 'profile' => 'default', 'region' => 'us-west-2', 'version' => '2012-11-05' ]); try { $result = $client->createQueue(array( 'QueueName' => $queueName, 'Attributes' => array( 'ReceiveMessageWaitTimeSeconds' => 20 ), )); var_dump($result); } catch (AwsException $e) { // output error message if fails error_log($e->getMessage()); }
```
Managing Visibility Timeout in Amazon SQS with AWS SDK for PHP Version 3

A visibility timeout is a period of time during which Amazon SQS prevents other consuming components from receiving and processing a message. To learn more, see Visibility Timeout.

The following example shows how to:

- Change the visibility timeout of specified messages in a queue to new values, using ChangeMessageVisibilityBatch.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Change the Visibility Timeout of Multiple Messages

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueUrl = "QUEUE_URL";
$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->receiveMessage(array(
        'AttributeNames' => ['SentTimestamp'],
        'MaxNumberOfMessages' => 10,
        'MessageAttributeNames' => ['All'],
        'QueueUrl' => $queueUrl, // REQUIRED
    ));
    $messages = $result->get('Messages');
    if ($messages != null) {
        $entries = array();
        for ($i = 0; $i < count($messages); $i++) {
            array_push($entries, [
                'Id' => 'unique_is_msg' . $i, // REQUIRED
                'ReceiptHandle' => $messages[$i]['ReceiptHandle'], // REQUIRED
                'VisibilityTimeout' => 3600
            ]);;
        }
        $result = $client->changeMessageVisibilityBatch(
            'Entries' => $entries,
        );
    }
```
Sending and Receiving Messages in Amazon SQS with AWS SDK for PHP Version 3

To learn about Amazon SQS messages, see Sending a Message to an SQS Queue and Receiving and Deleting a Message from an SQS Queue in the Service Quotas User Guide.

The following examples show how to:

• Deliver a message to a specified queue using `SendMessage`.
• Retrieve one or more messages (up to 10) from a specified queue using `ReceiveMessage`.
• Delete a message from a queue using `DeleteMessage`.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Send a Message

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new SqsClient(
    ['profile' => 'default',
     'region' => 'us-west-2',
     'version' => '2012-11-05']
);
$params = [
    'DelaySeconds' => 10,
    'MessageAttributes' => [
        "Title" => [
            'DataType' => "String",
```
Sending and Receiving Messages

Receive and Delete Messages

Imports

```php
require 'vendor/autoload.php';

use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueUrl = "QUEUE_URL";

$client = new SqsClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2012-11-05'  
]);

try {
    $result = $client->receiveMessage(array(  
        'AttributeNames' => ['SentTimestamp'],  
        'MaxNumberOfMessages' => 1,  
        'MessageAttributeNames' => ['All'],  
        'QueueUrl' => $queueUrl, // REQUIRED  
        'WaitTimeSeconds' => 0,  
    ));
    if (!empty($result->get('Messages'))) {
        var_dump($result->get('Messages')[0]);
        $result = $client->deleteMessage([  
            'QueueUrl' => $queueUrl, // REQUIRED  
            'ReceiptHandle' => $result->get('Messages')[0]['ReceiptHandle'] // REQUIRED  
        ]);  
    } else {  
        echo "No messages in queue. \n";
    }
```
Using Dead-Letter Queues in Amazon SQS with AWS SDK for PHP Version 3

A dead-letter queue is one that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead-letter queue to determine why their processing did not succeed. You must individually configure each source queue that sends messages to a dead-letter queue. Multiple queues can target a single dead-letter queue.

To learn more, see Using SQS Dead Letter Queues.

The following example shows how to:

• Enable a dead-letter queue using SetQueueAttributes.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Enable a Dead-Letter Queue

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueUrl = "QUEUE_URL";
$client = new SqsClient(
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
);

try {
    $result = $client->setQueueAttributes(
        'Attributes' => [
            'RedrivePolicy' => "{"deadLetterTargetArn":"DEAD_LETTER_QUEUE_ARN","maxReceiveCount":"10"}",
            'QueueUrl' => $queueUrl // REQUIRED
        ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
Using Queues in Amazon SQS with AWS SDK for PHP version 3

To learn about Amazon SQS queues, see How SQS Queues Work.

The following examples show how to:

- Return a list of your queues using ListQueues.
- Create a new queue using CreateQueue.
- Return the URL of an existing queue using GetQueueUrl.
- Delete a specified queue using DeleteQueue.

All the example code for the AWS SDK for PHP Version 3 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials for the AWS SDK for PHP Version 3 (p. 42). Then import the AWS SDK for PHP, as described in Basic Usage Patterns of the AWS SDK for PHP Version 3 (p. 7).

Return a List of Queues

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$client = new SqsClient([  
    'profile' => 'default',  
    'region' => 'us-west-2',  
    'version' => '2012-11-05'
]);
try {
    $result = $client->listQueues();
    foreach ($result->get('QueueUrls') as $queueUrl) {
        echo $queueUrl
    }
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
```
Create a Queue

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueName = "SQS_QUEUE_NAME";
$client = new SqsClient(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05']);
try {
    $result = $client->createQueue(array(
        'QueueName' => $queueName,
        'Attributes' => array(
            'DelaySeconds' => 5,
            'MaximumMessageSize' => 4096, // 4 KB
        ),
    ),
); var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Return the URL of a Queue

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueName = "SQS_QUEUE_NAME";
$client = new SqsClient(
    ['profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05']);
try {
    $result = $client->getQueueUrl(['
        'QueueName' => $queueName // REQUIRED
    ]);
Delete a Queue

Imports

```php
require 'vendor/autoload.php';
use Aws\Sqs\SqsClient;
use Aws\Exception\AwsException;
```

Sample Code

```php
$queueUrl = "SQS_QUEUE_URL";
$client = new SqsClient(["profile" => 'default',
                        "region" => 'us-west-2',
                        "version" => '2012-11-05']);
try {
    $result = $client->deleteQueue(["QueueUrl" => $queueUrl // REQUIRED
                                    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```
Security for AWS SDK for PHP

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

**Security of the Cloud**– AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

**Security in the Cloud**– Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization’s requirements, and applicable laws and regulations.

**Topics**
- Data Protection in AWS SDK for PHP (p. 291)
- Identity and Access Management for AWS SDK for PHP (p. 292)
- Compliance Validation for AWS SDK for PHP (p. 292)
- Resilience for AWS SDK for PHP (p. 293)
- Infrastructure Security for AWS SDK for PHP (p. 293)
- Amazon S3 Encryption Client Migration (p. 293)

Data Protection in AWS SDK for PHP

The shared responsibility model applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management. That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customers’ account numbers, into free-form fields such as a Name field. This includes when you work with AWS SDK
Identity and Access Management for AWS SDK for PHP

AWS Identity and Access Management (IAM) is an Amazon Web Services (AWS) service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use resources AWS services. IAM is an AWS service that you can use with no additional charge.

To use AWS SDK for PHP to access AWS, you need an AWS account and AWS credentials. To increase the security of your AWS account, we recommend that you use an IAM user to provide access credentials instead of using your AWS account credentials.

For details about working with IAM, see IAM.

For an overview of IAM users and why they are important for the security of your account, see AWS Security Credentials in the Amazon Web Services General Reference.

AWS SDK for PHP follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Compliance Validation for AWS SDK for PHP

AWS SDK for PHP follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

The security and compliance of Amazon Web Services (AWS) services is assessed by third-party auditors as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others. AWS provides a frequently updated list of AWS services in scope of specific compliance programs at AWS Services in Scope by Compliance Program.

Third-party audit reports are available for you to download using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

For more information about AWS compliance programs, see AWS Compliance Programs.

Your compliance responsibility when using AWS SDK for PHP to access an AWS service is determined by the sensitivity of your data, your organization’s compliance objectives, and applicable laws and regulations. If your use of an AWS service is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- **Security and Compliance Quick Start Guides**— Deployment guides that discuss architectural considerations and provide steps for deploying security-focused and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper**— A whitepaper that describes how companies can use AWS to create HIPAA-compliant applications.
AWS Compliance Resources— A collection of workbooks and guides that might apply to your industry and location.

AWS Config— A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

AWS Security Hub— A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience for AWS SDK for PHP

The Amazon Web Services (AWS) global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

AWS SDK for PHP follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Infrastructure Security for AWS SDK for PHP

AWS SDK for PHP follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

For information about AWS security processes, see the AWS: Overview of Security Processes whitepaper.

Amazon S3 Encryption Client Migration

This topic shows how to migrate your applications from Version 1 (V1) of the Amazon Simple Storage Service (Amazon S3) encryption client to Version 2 (V2), and ensure application availability throughout the migration process.

Migration Overview

This migration happens in two phases:

1. **Update existing clients to read new formats.** First, deploy an updated version of the AWS SDK for PHP to your application. This allows existing V1 encryption clients to decrypt objects written by the new V2 clients. If your application uses multiple AWS SDKs, you must upgrade each SDK separately.

2. **Migrate encryption and decryption clients to V2.** Once all of your V1 encryption clients can read new formats, you can migrate your existing encryption and decryption clients to their respective V2 versions.
Update Existing Clients to Read New Formats

The V2 encryption client uses encryption algorithms that older versions of the client don’t support. The first step in the migration is to update your V1 decryption clients to the latest SDK release. After completing this step, your application’s V1 clients will be able to decrypt objects encrypted by V2 encryption clients. See details below for each major version of the AWS SDK for PHP.

Upgrading AWS SDK for PHP Version 3

Version 3 is the latest version of the AWS SDK for PHP. To complete this migration, you must use version 3.148.0 or later of the `aws/aws-sdk-php` package.

Installing from the Command Line

For projects that were installed using Composer, in the Composer file, update the SDK package to version 3.148.0 of the SDK and then run the following command.

```
composer update aws/aws-sdk-php
```

Installing Using the Phar or Zip File

Use one of the following methods. Be sure to place the updated SDK file in the location required by your code, which is determined by the require statement.

For projects that were installed using the Phar file, download the updated file: `aws.phar`.

```
<?php
    require '/path/to/aws.phar';
?>
```

For projects that were installed using the Zip file, download the updated file: `. `.

```
<?php
    require '/path/to/aws-autoloader.php';
?>
```

Migrate Encryption and Decryption Clients to V2

After updating your clients to read the new encryption formats, you can update your applications to the V2 encryption and decryption clients. The following steps show you how to successfully migrate your code from V1 to V2.

Requirements for Updating to V2 Clients

1. The AWS KMS encryption context must be passed into the `S3EncryptionClientV2::putObject` and `S3EncryptionClientV2::putObjectAsync` methods. AWS KMS encryption context is an associative array of key-value pairs, which you must add to the encryption context for AWS KMS key encryption. If no additional context is required, you can pass an empty array.

2. `@SecurityProfile` must be passed into the `getObject` and `getObjectAsync` methods in `S3EncryptionClientV2`. `@SecurityProfile` is a new mandatory parameter of the `getObject...` methods. If set to `'V2'`, only objects that are encrypted in V2-compatible format can be decrypted. Setting this parameter to `'V2_AND_LEGACY'` also allows objects encrypted in V1-compatible format to be decrypted. To support migration, set `@SecurityProfile` to `'V2_AND_LEGACY'`. Use `'V2'` only for new application development.
3. (optional) Include the @KmsAllowDecryptWithAnyCmk parameter in the
S3EncryptionClientV2::getObject and S3EncryptionClientV2::getObjectAsync*
methods. A new parameter has been added called @KmsAllowDecryptWithAnyCmk. Setting this
parameter to true enables decryption without supplying a KMS key. The default value is false.

4. For decryption with a V2 client, if the @KmsAllowDecryptWithAnyCmk parameter isn’t
set to true for the “getObject...” method calls, a kms-key-id must be supplied to the
KmsMaterialsProviderV2 constructor.

Migration Examples

Example 1: Migrating to V2 Clients

Pre-migration

```php
use Aws\S3\Crypto\S3EncryptionClient;
use Aws\S3\S3Client;

$encryptionClient = new S3EncryptionClient(
    new S3Client(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => 'latest',
    ]
));
```

Post-migration

```php
use Aws\S3\Crypto\S3EncryptionClientV2;
use Aws\S3\S3Client;

$encryptionClient = new S3EncryptionClientV2(
    new S3Client(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => 'latest',
    ]
));
```

Example 2: Using KMS with kms-key-id

**Note**
These examples use imports and variables defined in Example 1. For example,
$encryptionClient.

Pre-migration

```php
use Aws\Crypto\KmsMaterialsProvider;
use Aws\Kms\KmsClient;

#kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProvider(
    new KmsClient(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => 'latest',
    ]),
    #kmsKeyId
);
```
#bucket = 'the-bucket-name';
#key = 'the-file-name';
$cipherOptions = [
    'Cipher' => 'gcm',
    'KeySize' => 256,
];

$encryptionClient->putObject(
    ['@MaterialsProvider' => $materialsProvider,
     '@CipherOptions' => $cipherOptions,
     'Bucket' => $bucket,
     'Key' => $key,
     'Body' => fopen('file-to-encrypt.txt', 'r'),
  ]);  

$result = $encryptionClient->getObject(
    ['@MaterialsProvider' => $materialsProvider,
     '@CipherOptions' => $cipherOptions,
     'Bucket' => $bucket,
     'Key' => $key,
  ]);    

\---

use Aws\Crypto\KmsMaterialsProviderV2;
use Aws\Kms\KmsClient;

$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV2(
    new KmsClient(
        ['profile' => 'default',
         'region' => 'us-east-1',
         'version' => 'latest'],
    ),
    $kmsKeyId
);

#bucket = 'the-bucket-name';
#key = 'the-file-name';
$cipherOptions = [
    'Cipher' => 'gcm',
    'KeySize' => 256,
];

$encryptionClient->putObject(
    ['@MaterialsProvider' => $materialsProvider,
     '@CipherOptions' => $cipherOptions,
     '@KmsEncryptionContext' => ['context-key' => 'context-value'],
     'Bucket' => $bucket,
     'Key' => $key,
     'Body' => fopen('file-to-encrypt.txt', 'r'),
  ]);  

$result = $encryptionClient->getObject(
    ['@KmsAllowDecryptWithAnyCmk' => true,
      '@SecurityProfile' => 'V2_AND_LEGACY',
      '@MaterialsProvider' => $materialsProvider,
      '@CipherOptions' => $cipherOptions,
      'Bucket' => $bucket,
      'Key' => $key,
  ]);
FAQ for AWS SDK for PHP Version 3

What methods are available on a client?

The AWS SDK for PHP uses service descriptions and dynamic magic __call() methods to execute API operations. You can find a full list of methods available for a web service client in the API documentation of the client.

What do I do about a cURL SSL certificate error?

This issue can occur when using an out-of-date CA bundle with cURL and SSL. You can get around this issue by updating the CA bundle on your server or downloading a more up-to-date CA bundle from the cURL website directly.

By default, the AWS SDK for PHP will use the CA bundle that is configured when PHP is compiled. You can change the default CA bundle used by PHP by modifying the openssl.cafile PHP .ini configuration setting to be set to the path of a CA file on disk.

What API versions are available for a client?

A version option is required when creating a client. A list of available API versions can be found on each client's API documentation page ::aws-php-class:<index.html>. If you're unable to load a specific API version, you might need to update your copy of the AWS SDK for PHP.

You can provide the string latest to the "version" configuration value to use the most recent available API version that your client's API provider can find (the default api_provider will scan the src/data directory of the SDK for API models).

Warning
We don't recommend using latest in a production application because pulling in a new minor version of the SDK that includes an API update could break your production application.

What Region versions are available for a client?

A region option is required when creating a client, and is specified using a string value. For a list of available AWS Regions and endpoints, see AWS Regions and Endpoints in the AWS General Reference.

```php
// Set the Region to the EU (Frankfurt) Region.
$s3 = new Aws\S3\S3Client([  'region' => 'eu-central-1',  'version' => '2006-03-01'
]);
```
Why can’t I upload or download files larger than 2 GB?

Because PHP’s integer type is signed, and many platforms use 32-bit integers, the AWS SDK for PHP doesn’t correctly handle files larger than 2 GB on a 32-bit stack (where “stack” includes CPU, OS, web server, and PHP binary). This is a well-known PHP issue. In the case of Microsoft Windows, only builds of PHP 7 support 64-bit integers.

The recommended solution is to use a 64-bit Linux stack, such as the 64-bit Amazon Linux AMI, with the latest version of PHP installed.

For more information, see PHP filesize: Return values.

How can I see what data is sent over the wire?

You can get debug information, including the data sent over the wire, using the debug option in a client constructor. When this option is set to true, all of the mutations of the command being executed, the request being sent, the response being received, and the result being processed are emitted to STDOUT. This includes the data that is sent and received over the wire.

```php
$s3Client = new Aws\S3\S3Client([  
    'region' => 'us-standard',
    'version' => '2006-03-01',
    'debug' => true
]);
```

How can I set arbitrary headers on a request?

You can add any arbitrary headers to a service operation by adding a custom middleware to the Aws\HandlerList of an Aws\CommandInterface or Aws\ClientInterface. The following example shows how to add an X-Foo-Baz header to a specific Amazon S3 PutObject operation using the Aws\Middleware::mapRequest helper method.

See mapRequest (p. 72) for more information.

How can I sign an arbitrary request?


See Signing Custom Amazon CloudSearch Domain Requests with AWS SDK for PHP Version 3 (p. 136) for a full example of how to do this.

How can I modify a command before sending it?

You can modify a command before sending it by adding a custom middleware to the Aws\HandlerList of an Aws\CommandInterface or Aws\ClientInterface. The following example shows how to add custom command parameters to a command before it’s sent, essentially adding default options. This example uses the Aws\Middleware::mapCommand helper method.
What is a CredentialsException?

If you are seeing an `Aws\Exception\CredentialsException` while using the AWS SDK for PHP, it means that the SDK was not provided with any credentials and was unable to find credentials in the environment.

If you instantiate a client without credentials, the first time that you perform a service operation the SDK will attempt to find credentials. It first checks in some specific environment variables, then it looks for instance profile credentials, which are only available on configured Amazon EC2 instances. If absolutely no credentials are provided or found, an `Aws\Exception\CredentialsException` is thrown.

If you are seeing this error and you are intending to use instance profile credentials, you need to be sure that the Amazon EC2 instance that the SDK is running on is configured with an appropriate IAM role.

If you are seeing this error and you are not intending to use instance profile credentials, you need to be sure that you are properly providing credentials to the SDK.

For more information, see [Credentials for the AWS SDK for PHP Version 3](p. 42).

Does the AWS SDK for PHP work on HHVM?

The AWS SDK for PHP doesn’t currently run on HHVM, and won’t be able to until the [issue with the yield semantics in HHVM](p. 299) is resolved.

How do I disable SSL?

You can disable SSL by setting the `scheme` parameter in a client factory method to ‘http’. It is important to note that not all services support http access. See [AWS Regions and Endpoints](p. 299) in the AWS General Reference for a list of regions, endpoints, and the supported schemes.

```php
$client = new Aws\DynamoDb\DynamoDbClient([  'version' => '2012-08-10',  'region'  => 'us-west-2',  'scheme'  => 'http']);
```

**Warning**

Because SSL requires all data to be encrypted and requires more TCP packets to complete a connection handshake than just TCP, disabling SSL may provide a small performance improvement. However, with SSL disabled, all data is sent over the wire unencrypted. Before disabling SSL, you must carefully consider the security implications and the potential for eavesdropping over the network.

What do I do about a “Parse error”?

The PHP engine will throw parsing errors when it encounters syntax it doesn’t understand. This is almost always encountered when attempting to run code that was written for a different version of PHP.
Why is the Amazon S3 client decompressing gzipped files?

Some HTTP handlers, including the default Guzzle 6 HTTP handler, will inflate compressed response bodies by default. You can override this behavior by setting the `decode_content` (p. 32) HTTP option to `false`. For backward-compatibility reasons, this default cannot be changed, but we recommend that you disable content decoding at the S3 client level.

See `decode_content` (p. 32) for an example of how to disable automatic content decoding.

How do I disable body signing in Amazon S3?

You can disable body signing by setting the `ContentSHA256` parameter in the command object to `Aws\Signature\S3SignatureV4::UNSIGNED_PAYLOAD`. Then the AWS SDK for PHP will use it as the 'x-amz-content-sha-256' header and the body checksum in the canonical request.

```php
$s3Client = new Aws\S3\S3Client([    'version' => '2006-03-01',    'region' => 'us-standard']);

$params = [    'Bucket' => 'foo',    'Key' => 'baz',    'ContentSHA256' => Aws\Signature\S3SignatureV4::UNSIGNED_PAYLOAD ];

// Using operation methods creates command implicitly
$result = $s3Client->putObject($params);

// Using commands explicitly.
$command = $s3Client->getCommand('PutObject', $params);
$result = $s3Client->execute($command);
```

How is retry scheme handled in the AWS SDK for PHP?

The AWS SDK for PHP has a `RetryMiddleware` that handles retry behavior. In terms of 5xx HTTP status codes for server errors, the SDK retries on 500, 502, 503 and 504.

Throttling exceptions, including `RequestLimitExceeded`, `Throttling`, `ProvisionedThroughputExceeded`, `ThrottlingException`, `RequestThrottled` and `BandwidthLimitExceeded`, are also handled with retries.

The AWS SDK for PHP also integrates exponential delay with a backoff and jitter algorithm in the retry scheme. Furthermore, default retry behavior is configured as 3 for all services except Amazon DynamoDB, which is 10.
How do I handle exceptions with error codes?

Besides AWS SDK for PHP-customized Exception classes, each AWS service client has its own exception class that inherits from AwsException AwsException. You can determine more specific error types to catch with the API-specific errors listed under the Errors section of each method.

Error code information is available with getAwsErrorCode() from Aws\Exception\AwsException.

```php
$sns = new \Aws\Sns\SnsClient([  
    'region' => 'us-west-2',  
    'version' => 'latest',  
]);

try {
    $sns->publish([  
        // parameters
    ]);  
    // Do something
} catch (SnsException $e) {
    switch ($e->getAwsErrorCode()) {
        case 'EndpointDisabled':
        case 'NotFound':
            // Do something
            break;
    }
}
```
Glossary

API Version

Services have one or more API versions, and which version you are using dictates which operations and parameters are valid. API versions are formatted like a date. For example, the latest API version for Amazon S3 is 2006-03-01. Specify a version (p. 42) when you configure a client object.

Client

Client objects are used to execute operations for a service. Each service that is supported in the SDK has a corresponding client object. Client objects have methods that correspond one-to-one with the service operations. See the basic usage guide (p. 7) for details on how to create and use client objects.

Command

Command objects encapsulate the execution of an operation. When following the basic usage patterns (p. 7) of the SDK, you will not deal directly with command objects. Command objects can be accessed using the getCommand() method of a client, in order to use advanced features of the SDK like concurrent requests and batching. See the Command Objects in the AWS SDK for PHP Version 3 (p. 58) guide for more details.

Credentials

To interact with AWS services, authenticate with the service using your credentials, or AWS access keys. Your access keys consist of two parts: your access key ID, which identifies your account, and your secret access, which is used to create signatures when executing operations. Provide credentials (p. 42) when you configure a client object.

Handler

A handler is a function that performs the actual transformation of a command and request into a result. A handler typically sends HTTP requests. Handlers can be composed with middleware to augment their behavior. A handler is a function that accepts an Aws\CommandInterface and a Psr\Http\Message\RequestInterface and returns a promise that is fulfilled with an Aws\ResultInterface or rejected with an Aws\Exception\AwsException reason.

JMESPath

JMESPath is a query language for JSON-like data. The AWS SDK for PHP uses JMESPath expressions to query PHP data structures. JMESPath expressions can be used directly on Aws\Result and Aws\ResultPaginator objects via the search($expression) method.

Middleware

Middleware is a special type of high-level function that augments the behavior of transferring a command and delegating to a "next" handler. Middleware functions accept an Aws\CommandInterface and a Psr\Http\Message\RequestInterface and return a promise that is fulfilled with an Aws\ResultInterface or rejected with an Aws\Exception\AwsException reason.

Operation

Refers to a single operation within a service's API (e.g., CreateTable for DynamoDB, RunInstances for Amazon EC2). In the SDK, operations are executed by calling a method of the same name on the corresponding service's client object. Executing an operation involves preparing and sending an HTTP request to the service and parsing the response. This process of executing an operation is abstracted by the SDK via command objects.
Paginator

Some AWS service operations are paginated and respond with truncated results. For example, Amazon S3's ListObjects operation only returns up to 1000 objects at a time. Operations like these require making subsequent requests with token (or marker) parameters to retrieve the entire set of results. Paginators are a feature of the SDK that act as an abstraction over this process to make it easier for developers to use paginated APIs. They are accessed via the `getPaginator()` method of the client. See the Paginators in the AWS SDK for PHP Version 3 (p. 78) guide for more details.

Promise

A promise represents the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its then method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled.

Region

Services are supported in one or more geographical regions. Services may have different endpoints/URLs in each region, which exist to reduce data latency in your applications. Provide a region (p. 37) when you configure a client object, so that the SDK can determine which endpoint to use with the service.

SDK

The term “SDK” can refer to the AWS SDK for PHP library as a whole, but also refers to the `Aws\Sdk` class (docs), which acts as a factory for the client objects for each service. The `Sdk` class also lets you provide a set of global configuration values (p. 23) that are applied to all client objects that it creates.

Service

A general way to refer to any of the AWS services (e.g., Amazon S3, Amazon DynamoDB, AWS OpsWorks, etc.). Each service has a corresponding client object in the SDK that supports one or more API versions. Each service also has one or more operations that make up its API. Services are supported in one or more regions.

Signature

When executing operations, the SDK uses your credentials to create a digital signature of your request. The service then verifies the signature before processing your request. The signing process is encapsulated by the SDK, and happens automatically using the credentials you configure for the client.

Waiter

Waiters are a feature of the SDK that make it easier to work with operations that change the state of a resource and that are eventually consistent or asynchronous in nature. For example, the Amazon DynamoDB CreateTable operation sends a response back immediately, but the table may not be ready to access for several seconds. Executing a waiter allows you to wait until a resource enters into a particular state by sleeping and polling the resource's status. Waiters are accessed using the `waitUntil()` method of the client. See the Waiters in the AWS SDK for PHP Version 3 (p. 80) guide for more details.

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.
Additional Resources

The following links provide additional resources you can use with the SDK for PHP version 3.

PHP SDK Forum

You can find questions and discussions on matters of interest to users of the SDK for PHP in the PHP SDK Forum.

PHP SDK v3 and Developer Guide on GitHub

There are several repos on GitHub for the AWS SDK for PHP.

- The current AWS SDK for PHP is available in the SDK repo.
- The SDK for PHP Developer Guide is available in restructured text format in its own documentation repo.
- The sample code included in this guide is available in the SDK sample code repo.

PHP SDK on Gitter

You can also find questions and discussions about the AWS SDK for PHP in the PHP SDK community on Gitter.
Document History

The following table describes the important changes since the last release of the AWS SDK for PHP Developer Guide.

_Last major documentation update: Aug 7th, 2020_

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3 encryption</td>
<td>Add topic on Amazon S3 encryption client migration</td>
<td>Aug 7th, 2020</td>
</tr>
<tr>
<td>Secrets Manager examples</td>
<td>Add more service examples</td>
<td>Mar 27th, 2019</td>
</tr>
<tr>
<td>Endpoint discovery</td>
<td>Configuration of endpoint discovery</td>
<td>Feb 15th, 2019</td>
</tr>
<tr>
<td>Amazon CloudFront</td>
<td>Add more service examples</td>
<td>Jan 25th, 2019</td>
</tr>
<tr>
<td>Service features</td>
<td>SDK metrics</td>
<td>Jan 11th, 2018</td>
</tr>
<tr>
<td>Amazon Kinesis, Amazon SNS</td>
<td>Add more service examples</td>
<td>Dec 14th, 2018</td>
</tr>
<tr>
<td>Amazon SES examples</td>
<td>Add more service examples</td>
<td>Oct 5th, 2018</td>
</tr>
<tr>
<td>AWS KMS examples</td>
<td>Add more service examples</td>
<td>Aug 8th, 2018</td>
</tr>
<tr>
<td>Credentials</td>
<td>Clarify and simplify credentials guide</td>
<td>June 30th, 2018</td>
</tr>
<tr>
<td>MediaConvert examples</td>
<td>Add more service examples</td>
<td>June 15th, 2018</td>
</tr>
<tr>
<td>New web layout</td>
<td>Documentation switched to AWS style</td>
<td>May 9th, 2018</td>
</tr>
<tr>
<td>Amazon S3 encryption</td>
<td>Client side encryption</td>
<td>Nov 17th, 2017</td>
</tr>
<tr>
<td>Amazon S3, Amazon SQS</td>
<td>Add more service examples</td>
<td>Mar 26th, 2017</td>
</tr>
<tr>
<td>Amazon S3, IAM, Amazon EC2</td>
<td>Add more service examples</td>
<td>Mar 17th, 2017</td>
</tr>
<tr>
<td>Add credentials</td>
<td>Adds support for AssumeRole and ini</td>
<td>Jan 17th, 2017</td>
</tr>
<tr>
<td>S3 Examples</td>
<td>S3 Multi-Region and presigned posts</td>
<td>Mar 18th, 2016</td>
</tr>
<tr>
<td>OpenSearch Service and Amazon CloudSearch</td>
<td>Add more service examples</td>
<td>Dec 28th, 2015</td>
</tr>
<tr>
<td>Command line</td>
<td>Add command parameters</td>
<td>Aug 13th, 2015</td>
</tr>
<tr>
<td>Service features</td>
<td>Adds service features for S3 and AWS</td>
<td>Apr 30th, 2015</td>
</tr>
<tr>
<td>New SDK version</td>
<td>Version 3 of the AWS SDK for PHP released.</td>
<td>May 26th, 2015</td>
</tr>
</tbody>
</table>