Developer Guide

AWS SDK for PHP

AWS SDK for PHP: Developer Guide

Table of Contents

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, and S3 Glacier. 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.

> **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.

Get started with the SDK

If you're ready to get hands-on with the SDK, follow the Get started chapter. It guides you through authentication with AWS, setting up your development environment, and creating your first basic application using Amazon S3.

Additional resources

- FAQ
- Glossary
- AWS SDKs and Tools Reference Guide: Contains settings, features, and other foundational concepts common among AWS SDKs.
- Guzzle Documentation
- Code examples using the AWS SDK for PHP is available in the awsdocs/aws-doc-sdk-examples repo.
- PHP SDK community on Gitter.
- AWS re:Post.

GitHub:

- Source code for the AWS SDK for PHP is available in the aws/aws-sdk-php repo.
- Contributing to the SDK
- 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 Reference Guide:

- AWS SDKs and Tools maintenance policy
- AWS SDKs and Tools version support matrix

Get started

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

Topics

SDK authentication with AWS

You must establish how your code authenticates with AWS when developing with AWS services. You can configure programmatic access to AWS resources in different ways depending on the environment and the AWS access available to you.

To choose your method of authentication and configure it for the SDK, see Authentication and access in the *AWS SDKs and Tools Reference Guide*.

We recommend that new users who are developing locally and are not given a method of authentication by their employer should set up AWS IAM Identity Center. This method includes installing the AWS CLI for ease of configuration and for regularly signing in to the AWS access portal. If you choose this method, your environment should contain the following elements after you complete the procedure for IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*:

- The AWS CLI, which you use to start an AWS access portal session before you run your application.
- A shared `AWSconfig` file that has a `[default]` profile with a set of configuration values that can be referenced by the SDK. To find the location of this file, see Location of the shared files in the *AWS SDKs and Tools Reference Guide*.
- The shared `config` file contains the `region` setting. This sets the default AWS Region that the SDK uses for requests. This Region is used for SDK service requests that aren't explicitly configured with a `region` property.

- The SDK uses the profile's SSO token provider configuration to acquire credentials before sending requests to AWS. The `sso_role_name` value, which is an IAM role connected to an IAM Identity Center permission set, allows access to the AWS services used in your application.

  The following sample `config` file shows a default profile set up with SSO token provider configuration. The profile's `sso_session` setting refers to the named `sso-session` section. The `sso-session` section contains settings to initiate an AWS access portal session.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

The AWS SDK for PHP does not need additional packages (such as SSO and SSOOIDC) to be added to your application to use IAM Identity Center authentication.

Start an AWS access portal session

Before running an application that accesses AWS services, you need an active AWS access portal session for the SDK to use IAM Identity Center authentication to resolve credentials. Depending on your configured session lengths, your access will eventually expire and the SDK will encounter an authentication error. To sign in to the AWS access portal, run the following command in the AWS CLI.

```
aws sso login
```

If you followed the guidance and have a default profile setup, you do not need to call the command with a `--profile` option. If your SSO token provider configuration is using a named profile, the command is `aws sso login --profile named-profile`.

To optionally test if you already have an active session, run the following AWS CLI command.

```
aws sts get-caller-identity
```

If your session is active, the response to this command reports the IAM Identity Center account and permission set configured in the shared `config` file.

> **Note**
>
> If you already have an active AWS access portal session and run `aws sso login`, you will not be required to provide credentials.
> The sign-in process might prompt you to allow the AWS CLI access to your data. Because the AWS CLI is built on top of the SDK for Python, permission messages might contain variations of the `botocore` name.

Learn more about authentication

- For more details about using IAM Identity Center for authentication, see Understand IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*
- To learn more about best practices, see Security best practices in IAM in the *IAM User Guide*.
- To create short-term AWS credentials, see Temporary Security Credentials in the *IAM User Guide*.
- To learn about other credential providers that AWS SDK for PHP can use, see Standardized credential providers in the *AWS SDKs and Tools Reference Guide*.

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 CloudFront 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.

Install cURL 7.16.2 or later	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.
Use OPCache	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. When running Amazon Linux, you need to install the php56-opcache or php55-opcache yum package to use the OPCache extension.
Uninstall Xdebug in production environments	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 `-o` or `==optimize-autoloader` option to Composer's install command.

Compatibility test

Run the `compatibility-test.php` file located in the SDK code base 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.

Install 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.

> **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 on the Download Composer page.

- For **Windows**, follow the Windows Installer instructions.
- For **Linux**, follow the Command-line installation instructions.

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

Installing Composer creates several folders and files in your environment. The primary file you will use is `autoload.php`, which is in the `vendor` folder in your environment.

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

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

Hello tutorial for the AWS SDK for PHP

Say hello to Amazon S3 using the AWS SDK for PHP. The following example displays a list of your Amazon S3 buckets.

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.

Installation Technique	Require Statement
Using Composer	`require '/path/to/vendor/autoload.php';`
Using the phar	`require '/path/to/aws.phar';`
Using the ZIP	`require '/path/to/aws-autoloader.php';`

In this topic, we 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.

Write the code

Copy and paste the following code into a new source file. Save and name the file `hello-s3.php`.

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

use Aws\S3\S3Client;


/**
 * List your Amazon S3 buckets.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */
```

```
//Create a S3Client
$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";
}
```

Running the program

Open a command prompt to run your PHP program. The typical command syntax to run a PHP program is:

```
php [source filename] [arguments...]
```

This sample code uses no arguments. To run this code, enter the following into the command prompt:

```
$ php hello-s3.php
```

Next steps

To test out many other Amazon S3 operations, check out the AWS Code Examples Repository on GitHub.

Use AWS Cloud9 with the AWS SDK for PHP

AWS Cloud9 is a web-based integrated development environment (IDE) that contains a collection of tools that you use to code, build, run, test, debug, and release software in the cloud. You can use AWS Cloud9 with the AWS SDK for PHP to write and run your PHP code by using a browser. AWS Cloud9 includes tools such as a code editor and terminal. Because the AWS Cloud9 IDE is cloud based, you can work on your projects from your office, home, or anywhere by 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
- Step 2: Set up your AWS Cloud9 development environment
- Step 3: Set up the AWS SDK for PHP
- Step 4: Download example code
- Step 5: Run example code

Step 1: Set up your AWS account to use AWS Cloud9

To use AWS Cloud9, sign in to the AWS Cloud9 console from the AWS Management Console.

> **Note**
>
> If you are using AWS IAM Identity Center to authenticate, you might need to add the required permission of `iam:ListInstanceProfilesForRole` to the user-attached policy in the IAM console.

To set up an IAM entity in your AWS account to access AWS Cloud9 and 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.

For details, see Creating an Environment in AWS Cloud9 in the *AWS Cloud9 User Guide*.

> **Note**
>
> As you create your environment in the console for the first time, we recommend that you choose the option to **Create a new instance for environment (EC2)**. This option tells AWS Cloud9 to create an environment, launch an Amazon EC2 instance, and then connect the new instance to the new environment. This is the fastest way to begin using AWS Cloud9.

If the terminal isn't already open in the IDE, open it. On the menu bar in the IDE, choose **Window, New Terminal**. You can use the terminal window to install tools and build your applications.

Step 3: Set up the AWS SDK for PHP

After AWS Cloud9 opens the IDE for your development environment, use the terminal window to set up the AWS SDK for PHP in your environment.

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 on the Download Composer page.

- For **Windows**, follow the Windows Installer instructions.
- For **Linux**, follow the Command-line installation instructions.

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

Installing Composer creates several folders and files in your environment. The primary file you will use is `autoload.php`, which is in the `vendor` folder in your environment.

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

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

Step 4: Download example code

Use the terminal window to download example code for the AWS SDK for PHP into the AWS Cloud9 development environment.

To download a copy of all the code examples used in the official AWS SDK documentation into your environment's root directory, run the following command:

```
$ git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

The code examples for the AWS SDK for PHP are located at `ENVIRONMENT_NAME/aws-doc-sdk-examples/php` directory, where `ENVIRONMENT_NAME` is the name of your development environment.

To follow along using an Amazon S3 example, we recommend starting with code example `ENVIRONMENT_NAME/aws-doc-sdk-examples/php/example_code/s3/ListBuckets.php`. This example will list your Amazon S3 buckets. Use the terminal window to navigate to the `s3` directory and list the files.

```
$ cd aws-doc-sdk-examples/php/example_code/s3
$ ls
```

To open the file in AWS Cloud9, you can click on the `ListBuckets.php` directly in the terminal window.

For more support in understanding code examples, see AWS SDK for PHP Code Examples.

Step 5: Run example code

To run code in your AWS Cloud9 development environment, choose the **Run** button in the top menu bar. AWS Cloud9 automatically detects the `.php` file extension and uses the **PHP (built-in web server)** runner to run the code. However, for this example we actually want the **PHP (`cli`)**

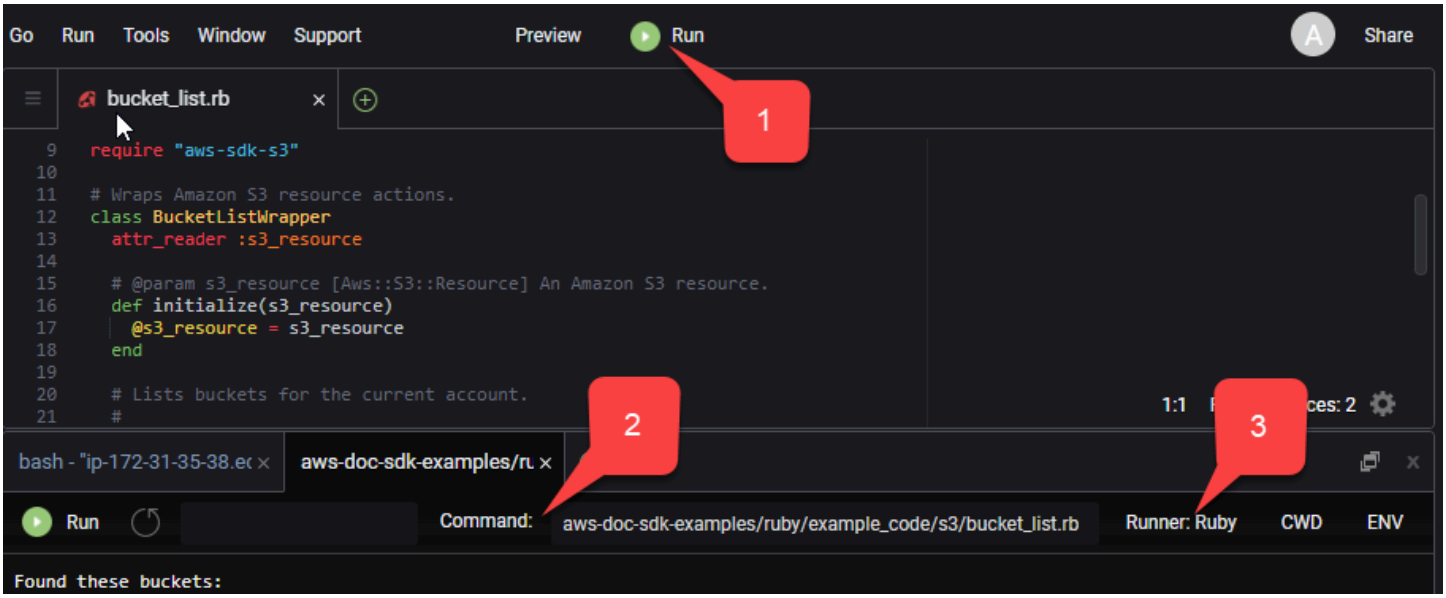
option. For more information about running code in AWS Cloud9, see Run Your Code in the *AWS Cloud9 User Guide*.

In the following screenshot, note these basic areas:

- 1: Run. The **Run** button is located on the top menu bar. This opens a new tab for your results.

> **Note**
>
> You can also manually create new run configurations. On the menu bar, choose **Run**, **Run Configurations**, **New Run Configuration**.

- 2: Command. AWS Cloud9 populates the **Command** text box with the path and file name to the file you run. If your code expects any command line parameters to be passed in, these can be added to the command line in the same way that you would when running the code through a terminal window.
- 3: Runner. AWS Cloud9 detects that your file extension is `.php` and selects the **PHP (built-in web server)** Runner to run your code. Select **PHP (`cli`)** to run this example instead.



Any output generated from the running code displays in the tab.

Configure 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.

The AWS SDKs and Tools Reference Guide also contains settings, features, and other foundational concepts common among many of the AWS SDKs.

Topics

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 install the SDK
- Before you use the AWS SDK for PHP, you must authenticate with AWS. For information about setting up authentication, see SDK authentication with AWS

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.

Installation Technique	Require Statement
Using Composer	`require '/path/to/vendor/autoload.php';`
Using the phar	`require '/path/to/aws.phar';`
Using the ZIP	`require '/path/to/aws-autoloader.php';`

In this topic, we 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 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

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

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

Sample Code

```
//Create an S3Client
$s3 = new Aws\S3\S3Client([
    'region' => 'us-east-2'  // Since version 3.277.10 of the SDK,
]);                          // the 'version' parameter defaults to 'latest'.
```

Information about the optional "version" parameter is available in the configuration options topic.

Notice that we did **not** explicitly provide credentials to the client. That's because the SDK should detect the credentials from environment variables, the Shared `config` and `credentials` files in your HOME directory, AWS Identity and Access Management (IAM) instance profile credentials, or credential providers.

All of the general client configuration options are described in detail in Configuration for the AWS SDK for PHP Version 3. 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. Many of the 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

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

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

Sample Code

```
// 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'
];
// 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.).

```
$sdk = new Aws\Sdk([
    'region'   => 'us-west-2',
    'DynamoDb' => [
        'region' => 'eu-central-1'
    ]
]);

// Creating an Amazon DynamoDb client will use the "eu-central-1" AWS Region
$client = $sdk->createDynamoDb();
```

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

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

```
use Aws\S3\S3Client;
```

Sample Code

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

// 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.

```
// 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

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

Sample Code

```
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk([
    'region'   => 'us-west-2'
]);
// 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

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

Sample Code

```
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk([
    'region'   => 'us-west-2'
]);
// 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

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

Sample Code

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

// 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

```
$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

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

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

Sample Code

```
// Create an SDK class used to share configuration across clients.
$sdk = new Aws\Sdk([
    'region'   => 'us-west-2'
]);

// 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

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

use Aws\S3\S3Client;
use Aws\Exception\AwsException;
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();
}
```

Configuration for the AWS SDK for PHP Version 3

Client constructor 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. For example, this may be useful if you are restricting file access to certain directories with `open_basedir`.

For more information on the location and formatting of the shared AWS `config` and `credentials` files, see Configuration in the *AWS SDKs and Tools Reference Guide*.

For details on all global configuration settings that you can set in the AWS configuration files or as environment variables, see Configuration and authentication settings reference in the *AWS SDKs and Tools Reference Guide*.

Configuration Options

- api_provider
- credentials
- debug
- stats
- endpoint
- endpoint_provider
- endpoint_discovery
- handler
- http
- http_handler
- profile
- region
- retries
- scheme
- service
- signature_provider
- signature_version
- ua_append
- use_aws_shared_config_files
- validate
- version

The following example shows how to pass options into an Amazon S3 client constructor.

```
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 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. The following specifies that the IAM Identity Center credential provider should be used. This provider is also known as the SSO credential provider.

```
$credentials = Aws\Credentials\CredentialProvider::sso('profile default');

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

If you use a named profile, substitute the name of your profile for 'default' in the previous example. To learn more about setting up named profiles, see Shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

If you don't specify a credential provider to use, and rely on the credential provider chain, the error message resulting from failed authentication is typically generic. It is generated from the last provider in the list of sources being checked for valid credentials, which may not be the provider you are attempting to use. When you specify which credential provider to use, any resulting error message is more helpful and relevant because it results from only that provider. To learn more about the chain of sources checked for credentials, see Credential provider chain in the *AWS SDKs and Tools Reference Guide*.

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

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

Pass a callable credential provider function to create credentials using a function.

```
use Aws\Credentials\CredentialProvider;

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

$s3 = new Aws\S3\S3Client([
    '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.

```
use Aws\Credentials\CredentialProvider;
use Aws\PsrCacheAdapter;
use Symfony\Component\Cache\Adapter\FilesystemAdapter;


$cache = new PsrCacheAdapter(new FilesystemAdapter);
$provider = CredentialProvider::defaultProvider();
```

```
$cachedProvider = CredentialProvider::cache($provider, $cache);

$s3 = new Aws\S3\S3Client([
    '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 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.

```
$s3 = new Aws\S3\S3Client([
    '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`.

```
$s3 = new Aws\S3\S3Client([
    'region'  => 'us-west-2',
    'debug'   => [
        'logfn'        => function ($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**
>
> This option also outputs the underlying HTTP handler information produced by the `http` debug option. 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.

```
$s3 = new Aws\S3\S3Client([
    '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.

```
$s3 = new Aws\S3\S3Client([
    'region'  => 'us-west-2',
    '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 AWS Elemental 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:

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

See the AWS Regions and Endpoints for a list of available AWS Regions and endpoints.

endpoint_provider

Type

`Aws\EndpointV2\EndpointProviderV2|callable`

An optional instance of EndpointProviderV2 or 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.

```
$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.

```
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([
    'region' => 'us-east-2',
    'endpoint_discovery' => $config,

]);
```

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

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

$s3 = new S3Client([
```

```
    'region'      => 'us-west-2',
    'endpoint_discovery' => new DoctrineCacheAdapter(new ApcuCache),
]);
```

Pass an array to endpoint discovery.

```
use Aws\S3\S3Client;

$s3 = new S3Client([
    '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\Promise\PromiseInterface`) that is fulfilled with an `Aws\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.

```
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-east-1',
    '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.

```
use Aws\S3\S3Client;
```

```
$client = new S3Client([
    'region'  => 'us-west-2',
    'http'    => ['cert' => '/path/to/cert.pem']
]);
```

- Set as an array containing the path and password.

```
use Aws\S3\S3Client;

$client = new S3Client([
    'region'  => 'us-west-2',
    '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).

```
use Aws\DynamoDb\DynamoDbClient;

// Timeout after attempting to connect for 5 seconds
$client = new DynamoDbClient([
    'region'  => 'us-west-2',
    '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.

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',
    '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.

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.

> **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.

```
use Aws\S3\S3Client;

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

// 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',
    'http'    => [
        'proxy' => 'http://192.168.16.1:10'
    ]
]);

// Send requests through a different proxy per scheme
$client = new DynamoDbClient([
    'region'  => 'us-west-2',
    'http'    => [
        'proxy' => [
            'http' => 'tcp://192.168.16.1:10',
            'https' => 'tcp://192.168.16.1:11',
        ]
    ]
]);
```

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).

```
use Aws\DynamoDb\DynamoDbClient;
```

```
// Timeout after 5 seconds
$client = new DynamoDbClient([
    'region'  => 'us-west-2',
    '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 (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.

```
use Aws\DynamoDb\DynamoDbClient;

// Use a custom CA bundle
$client = new DynamoDbClient([
    'region'  => 'us-west-2',
    'http'    => [
        'verify' => '/path/to/my/cert.pem'
    ]
```

```
]);

// Disable SSL/TLS verification
$client = new DynamoDbClient([
    'region'  => 'us-west-2',
    'http'    => [
        'verify' => false
    ]
]);
```

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:

```
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',
    'http_handler' => $handler
]);
```

Note

The `http_handler` option supersedes any provided `handler` option.

profile

Type

`string`

The "profile" option specifies which profile to use when credentials are created from the AWS credentials file in your HOME directory (typically `~/.aws/credentials`). This setting overrides the `AWS_PROFILE` environment variable.

> **Note**
>
> When you specify the "profile" option, the "`credentials`" option is ignored and credential-related settings in the AWS configuration file (typically `~/.aws/config`) are 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 for more information about configuring credentials and the .ini file format.

region

Type

`string`

Required

true

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.

```
// 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

```
// Disable retries by setting "retries" to 0
$client = new Aws\DynamoDb\DynamoDbClient([
    'version' => '2012-08-10',
    'region'  => 'us-west-2',
    'retries' => 6
]);
```

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.

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

$enabled = true;
$cache_limit = 1000;

$config = new Aws\Retry\Configuration('adaptive', 10);

$s3 = new Aws\S3\S3Client([
    'region' => 'us-east-2',
    'retries' => $config,
]);
```

This example passes in an array for retry configuration.

```
use Aws\S3\S3Client;
```

```
$s3 = new S3Client([
    '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.

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

$s3 = new S3Client([
    '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”.

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

See the AWS Regions and 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:

```
// Set a preferred signature version
$s3 = new Aws\S3\S3Client([
    'version'           => '2006-03-01',
    'region'            => 'us-west-2',
    '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.

```
// 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).
- `min` - Validate the minimum length of a value (on by default).
- `max` - Validate the maximum length of a value.
- `pattern` - Validate that the value matches a regular expression.

```
// Validate only that required values are present
$s3 = new Aws\S3\S3Client([
    'version'  => '2006-03-01',
    'region'   => 'eu-west-1',
    'validate' => ['required' => true]
]);
```

version

Type

`string`

Required

false

This option specifies the version of the web service to use (e.g., `2006-03-01`).

Beginning with version 3.277.10 of the SDK, the "version" option is not required. If you don't specify the "version" option, the SDK uses the latest version of the service client.

Two situations require a "version" parameter when you create a service client.

- You use a version of the PHP SDK earlier than 3.277.10.
- You use version 3.277.10 or later and want to user a version other than the 'latest' for a service client.

For example, the following snippet uses version 3.279.7 of the SDK, but not the latest version for the `Ec2Client`.

```
$ec2Client = new \Aws\Ec2\Ec2Client([
    'version' => '2015-10-01',
    'region' => 'us-west-2'
]);
```

Specifying a version constraint ensures that your code will not be affected by a breaking change made to the service.

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.

Credentials for the AWS SDK for PHP Version 3

For reference information on available credentials mechanisms for the AWS SDKs, see Credentials and access in the *AWS SDKs and Tools Reference Guide*.

Important

For security, we *strongly recommend* that you do **not** use the root account for AWS access. Always refer to the Security best practices in IAM in the *IAM User Guide* for the latest security recommendations.

Precedence of settings

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. To learn more about the chain of sources checked for credentials, see Credential provider chain in the *AWS SDKs and Tools Reference Guide*.

The AWS SDK for PHP has a series of places that it checks in order to find values for global settings and credential providers. The following is the order of precedence:

1. Any explicit setting set in the code or on a service client itself takes precedence over anything else.
2. Use credentials from environment variables.

   Setting environment variables is useful if you're doing development work on a machine other than an Amazon EC2 instance.
3. Shared `config` and `credentials` files.

   These are the same files used by other SDKs and the AWS CLI.

Credential providers

- Using a credential provider.

  Provide custom logic for credentials when constructing the client.
- Assume an IAM role.

  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.
- Using temporary credentials from AWS STS.

  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.

- Creating anonymous clients.

Create a client that isn't associated with any credentials when the service allows anonymous access.

Use 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. For instructions on how to obtain these values, see Authenticate using short-term credentials in the *AWS SDKs and Tools Reference Guide*.

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

For more information on how to set environment variables, see Environment variables support in the *AWS SDKs and Tools Reference Guide*. Also, for a list of all environment variables supported by most AWS SDKs, see Environment variables list.

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

Linux

```
$ 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 temporary session key for your AWS account.
   # 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

```
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 temporary session key for your AWS account.
   # 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.
```

Assume an IAM role

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. For a list of the services that accept temporary security credentials, see AWS services that work with IAM in the *IAM User Guide*.

> **Note**
>
> To avoid hitting the metadata service every time, you can pass an instance of `Aws\CacheInterface` in as the `'credentials'` option to a client constructor. This lets the SDK use cached instance profile credentials instead. For details, see Configuration for the AWS SDK for PHP Version 3.

For more information on developing Amazon EC2 applications using the SDKs, see Using IAM roles for Amazon EC2 instances in the *AWS SDKs and Tools Reference Guide*.

Create and assign an IAM role to an Amazon EC2 instance

1. Create an IAM client.

   #### Imports

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

   use Aws\Iam\IamClient;
   ```

   #### Sample Code

   ```
   $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

   ```
   $result = $client->createRole([
       'AssumeRolePolicyDocument' => 'IAM JSON Policy', // REQUIRED
       'Description' => 'Description of Role',
       'RoleName' => 'RoleName', // REQUIRED
   ]);
   ```

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

   ```
   $IPN = 'InstanceProfileName';

   $result = $client->createInstanceProfile([
       'InstanceProfileName' => $IPN ,
   ]);
   ```

```
$ARN = $result['Arn'];
$InstanceID =  $result['InstanceProfileId'];
```

4. Create an Amazon EC2 client.

   Imports

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

use Aws\Ec2\Ec2Client;
```

   Sample Code

```
$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

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

For more information, see IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide*.

Using IAM roles for Amazon ECS tasks

A task in Amazon Elastic Container Service (Amazon ECS) can assume an IAM role to make AWS API calls. This is a strategy for managing credentials for your applications to use, similar to how Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

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

For more information on using IAM roles that container tasks can assume, see the Task IAM role topic in the *Amazon ECS Developer Guide*. For examples of using the task IAM role in the form of a `taskRoleArn` in task definitions, see Example task definitions also in the *Amazon ECS Developer Guide*.

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.

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. Credentials last an hour by default.

Sample Code

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

$ARN = "arn:aws:iam::123456789012:role/xaccounts3access";
$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 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

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

$ARN = "arn:aws:iam::123456789012:role/xaccounts3access";
$sessionName = "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.

Assume role with profile

Define profiles in `~/.aws/credentials`

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

Create a new profile with the `role_arn` setting for the role you want assumed. Also include the `source_profile` setting for another profile with credentials that have permissions to assume the IAM role. For more details on these configuration settings, see Assume role credentials in the *AWS SDKs and Tools Reference Guide*.

For example, in the following `~/.aws/credentials`, the `project1` profile sets the `role_arn` and specifies the `default` profile as the source for credentials to verify that the entity associated with them can assume the role.

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

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

If you set the `AWS_PROFILE` environment variable, or you use `profile` parameter when you instantiate a service client, the role specified in `project1` is assumed, using the `default` profile as the source credentials.

The following snippet shows the use of the `profile` parameter in an `S3Client` constructor. The `S3Client` will have the permissions associated with the role associated with the `project1` profile.

```
$s3 = new Aws\S3\S3Client([
    'region' => 'us-east-1',
    'version' => '2006-03-01',
    'profile' => 'project1'
]);
```

Define profiles in `~/.aws/config`

The `~/.aws/config` file can also contain profiles that you want to be assumed. If you set the environment variable `AWS_SDK_LOAD_NONDEFAULT_CONFIG`, the SDK for PHP loads profiles from the `config` file. When `AWS_SDK_LOAD_NONDEFAULT_CONFIG` is set, the SDK loads profiles from both `~/.aws/config` and `~/.aws/credentials`. Profiles from `~/.aws/credentials` are loaded last and 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.

The following example uses the `project1` profile defined in the `config` file and the `default` profile in the `credentials` file. The `AWS_SDK_LOAD_NONDEFAULT_CONFIG` is also set.

```
# 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.

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

When the `S3Client` constructor runs that is shown the following snippet, the role defined in the `project1` profile will be assumed using credentials associated with the `default` profile.

```
$s3 = new Aws\S3\S3Client([
    'region' => 'us-east-1',
```

```
    'version' => '2006-03-01',
    'profile' => 'project1'
]);
```

Use 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`.

```
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. For more information on configuring the standardized providers and the credential provider chain in AWS SDKs, see Standardized credential providers in the *AWS SDKs and Tools Reference Guide*.

> **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.

```
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";

$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.

SSO provider

`Aws\Credentials\CredentialProvider::sso` is the single sign-on credential provider. This provider is also known as the AWS IAM Identity Center credential provider.

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

$credentials = new Aws\CredentialProvider::sso('profile default');

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

If you use a named profile, substitute the name of your profile for '`default`' in the previous example. To learn more about setting up named profiles, see Shared `config` and `credentials` files in the *AWS SDKs and Tools Reference Guide*. Alternatively, you can use the `AWS_PROFILE` environment variable to specify which profile's settings to use.

To understand more how the IAM Identity Center provider works, see Understand IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.

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.

```
// 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;

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

assumeRoleWithWebIdentityCredentialProvider 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',
    'version'     => '2006-03-01',
    'credentials' => $provider
]);
```

ini provider

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

```
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.

```
$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. By default, the SDK attempts to load the "default" profile from the shared AWS `credentials` 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:

```
{
    "Version": 1,
    "AccessKeyId": "",
    "SecretAccessKey": "",
```

```
    "SessionToken": "",
    "Expiration": ""
}
```

`SessionToken` and `Expiration` are optional. If present, the credentials will be treated as temporary.

```
use Aws\Credentials\CredentialProvider;
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.

```
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
// across multiple clients. Each time a new client is constructed,
// 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.

Use 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. For deeper understanding, see Temporary Security Credentials in the *IAM User Guide*. You can use temporary security credentials to access most AWS services. For a list of the services that accept temporary security credentials, see AWS services that work with IAM in the *IAM User Guide*.

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).

Getting temporary credentials

AWS STS has several operations that return temporary credentials, but the `GetSessionToken` operation is the simplest to demonstrate. The following snippet retrieves temporary credentials by calling the `getSessionToken` method of the PHP SDK's STS client.

```
$sdk = new Aws\Sdk([
    'region'   => 'us-east-1',
]);

$stsClient = $sdk->createSts();

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

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

```
Array
(
    ...
    [Credentials] => Array
    (
        [SessionToken] => '<base64 encoded session token value>'
        [SecretAccessKey] => '<temporary secret access key value>'
        [Expiration] => 2013-11-01T01:57:52Z
        [AccessKeyId] => '<temporary 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.

Create 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`.

```
$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 `Aws\S3\S3Client::putObject()`, the SDK actually creates an `Aws\CommandInterface` object based on the provided parameters, executes the command, and returns a populated `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., `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.

```
$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.

`@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 `@http` parameter are the same as the ones you can set when you instantiate the client with the "http" client option.

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

`@retries`

Like the "retries" client option, `@retries` controls how many times a command can be retried before it is considered to have failed. Set it to `0` to disable retries.

```
// Disable retries
$command['@retries'] = 0;
```

> **Note**
>
> If you have disabled retries on a client, you cannot selectively enable them on individual commands passed to that client.

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.

```
$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 `HandlerList`

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.

```
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 = $client->execute($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.

```
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`.

```
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
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 (\Iterator $files, $bucket) use ($client) {
    foreach ($files as $file) {
        // Skip "." and ".." files
        if ($file->isDot()) {
            continue;
        }
        $filename = $file->getPath() . '/' . $file->getFilename();
        // Yield a command that is executed by the pool
        yield $client->getCommand('PutObject', [
            'Bucket' => $bucket,
            'Key'    => $file->getBaseName(),
            'Body'   => fopen($filename, 'r')
        ]);
    }
};

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

// 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,
        PromiseInterface $aggregatePromise
    ) {
        echo "Completed {$iterKey}: {$result}\n";
    },
    // Invoke this function for each failed transfer
    'rejected' => function (
        AwsException $reason,
        $iterKey,
        PromiseInterface $aggregatePromise
    ) {
        echo "Failed {$iterKey}: {$reason}\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.

```
$pool = new CommandPool($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, waiters, command pools, multipart uploads, S3 directory/bucket transfers, 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.

```
$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).

```
$promise->then(
    function ($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.

```
use GuzzleHttp\Promise\Utils;

$sdk = new Aws\Sdk([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

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

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

```
// Wait for both promises to complete.
$results = Utils::unwrap($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.

```
$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.

```
$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.

```
use Aws\Exception\AwsException;

$promise = $client->listTablesAsync();

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.

```
$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 promises that are awaiting delivery from the promise. A canceled promise is rejected with a `GuzzleHttp\Promise\RejectionException`.

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 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.

```
use GuzzleHttp\Promise;

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

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

// Create a generator that yields promises
$promiseGenerator = function () use ($s3, $ddb) {
    yield $s3->listBucketsAsync();
    yield $ddb->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.

```
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.

```
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.

```
// 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`.

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

> **Note**
>
> To change the handler of a multi-region client after it's constructed, use the `useCustomHandler` method of an `Aws\MultiRegionClient`.

```
$multiRegionClient->useCustomHandler($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.

```
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();
```

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.

```
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 handle 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 it 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) {
        // 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 `Aws\CommandInterface` object and `Psr\Http\Message\RequestInterface` object, and returns a `GuzzleHttp\Promise\PromiseInterface` that is fulfilled with an `Aws\ResultInterface` or rejected with an `Aws\Exception\AwsException`.

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

- connect_timeout
- debug
- decode_content (optional)
- delay

- progress (optional)
- proxy
- sink
- synchronous (optional)
- stream (optional)
- timeout
- verify
- http_stats_receiver (optional) - A function to invoke with an associative array of HTTP transfer statistics if requested using the stats 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 `User-Agent` header that takes the following form, where "3.X" can be replaced with `Aws\Sdk::VERSION` and "HandlerSpecificData/version ..." should be replaced with your handler-specific User-Agent string.

```
User-Agent: aws-sdk-php/3.X HandlerSpecificData/version ...
```

Streams in the AWS SDK for PHP Version 3

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

> **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 `GuzzleHttp\Psr7\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 `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.

```
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.

```
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 S3 `ListObjects` 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.

```
$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 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.

```
$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.

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

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

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.

```
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'my-bucket'
]);

$promise = $results->each(function ($result) {
    echo 'Got ' . var_export($result, true) . "\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.

```
// 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 waiters 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 specific 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.

```
// 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.

```
// 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.

```
use Aws\CommandInterface;

$s3Client->waitUntil('BucketExists', [
    'Bucket'  => 'my-bucket',
    '@waiter' => [
        'before' => function (CommandInterface $command, $attempts) {
            printf(
                "About to send %s. Attempt %d\n",
                $command->getName(),
                $attempts
            );
        }
    ]
]);
```

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($name, array $args = [])` method. Use the `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.

```
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?

```
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
// first waiter to complete (or the first bucket to become available)
$any = Promise\any($promises)
    ->then(function (CommandInterface $command) {
        // This is invoked with the command that succeeded in polling the
        // resource. Here we can know which bucket won the race.
        echo "The {$command['Bucket']} waiter completed first!\n";
    });

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

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 and Waiters in the AWS SDK for PHP Version 3, 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.

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

```
{
    "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"
                }
            ],
            "VolumeType": "standard",
            "VolumeId": "vol-e11a5288",
            "State": "in-use",
            "SnapshotId": "snap-f23ec1c8",
            "CreateTime": "2013-09-17T00:55:03.000Z",
            "Size": 30
        },
        {
            "AvailabilityZone": "us-west-2a",
            "Attachments": [
                {
                    "AttachTime": "2013-09-18T20:26:16.000Z",
                    "InstanceId": "i-4b41a37c",
                    "VolumeId": "vol-2e410a47",
                    "State": "attached",
                    "DeleteOnTermination": true,
                    "Device": "/dev/sda1"
                }
            ],
            "VolumeType": "standard",
            "VolumeId": "vol-2e410a47",
            "State": "in-use",
            "SnapshotId": "snap-708e8348",
            "CreateTime": "2013-09-18T20:26:15.000Z",
            "Size": 8
```

```
        }
    ],
    "@metadata": {
        "statusCode": 200,
        "effectiveUri": "https:\/\/ec2.us-west-2.amazonaws.com",
        "headers": {
            "content-type": "text\/xml;charset=UTF-8",
            "transfer-encoding": "chunked",
            "vary": "Accept-Encoding",
            "date": "Wed, 06 May 2015 18:01:14 GMT",
            "server": "AmazonEC2"
        }
    }
}
```

First, we can retrieve only the first volume from the Volumes list with the following command.

```
$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.

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

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

```
array(2) {
  [0] =>
  array(3) {
    'AZ' =>
    string(10) "us-west-2a"
    'ID' =>
    string(12) "vol-e11a5288"
    'Size' =>
    int(30)
  }
  [1] =>
  array(3) {
    'AZ' =>
    string(10) "us-west-2a"
```

```
    'ID' =>
    string(12) "vol-2e410a47"
    'Size' =>
    int(8)
  }
}
```

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.)

```
$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:

```
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.

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

Running the previous search produces the following data:

```
array(2) {
  [0] =>
  array(4) {
    [0] =>
    string(12) "vol-e11a5288"
    [1] =>
    string(10) "i-a071c394"
    [2] =>
    string(10) "us-west-2a"
    [3] =>
    int(30)
  }
  [1] =>
  array(4) {
    [0] =>
    string(12) "vol-2e410a47"
    [1] =>
    string(10) "i-4b41a37c"
    [2] =>
    string(10) "us-west-2a"
    [3] =>
    int(8)
  }
}
```

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 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);
}
```

Use the AWS Common Runtime (AWS CRT) extension

The AWS CRT libraries provide basic functionality with good performance and minimal footprint for several AWS SDKs. This topic discusses when the AWS CRT is used by the SDK for PHP and how to install the AWS CRT extension.

When you need the AWS CRT extension installed

The SDK for PHP uses the authorization and checksum functionality of the AWS CRT libraries. The AWS CRT extension is required when you work with:

- Amazon S3 Multi-Region Access Points
- Amazon EventBridge global endpoints
- A CRC-32C checksum algorithm in Amazon Simple Storage Service (Amazon S3)

If you use a feature listed above and the AWS CRT extention is not installed in your PHP environment, the SDK for PHP will report an error message and remind you to install the extension.

Install the AWS Common Runtime (AWS CRT) extension

Instructions on how to install the AWS CRT extension are available on the main page of the GitHub repository for the aws-crt-php.

Upgrade 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 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 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.

```
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 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.

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 RDSDescribeEngineDefaultParameters result in version 3 now includes a wrapping "EngineDefaults" element. In version 2, this element was not present.

```
$client = new Aws\Rds\RdsClient([
    'region'  => 'us-west-1',
    'version' => '2014-09-01'
]);

// Version 2
$result = $client->describeEngineDefaultParameters();
$family = $result['DBParameterGroupFamily'];
$marker = $result['Marker'];
```

```
// Version 3
$result = $client->describeEngineDefaultParameters();
$family = $result['EngineDefaults']['DBParameterGroupFamily'];
$marker = $result['EngineDefaults']['Marker'];
```

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. 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.

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

```
// 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";
    }
}
```

Paginator objects have a `search()` method that enables you to use JMESPath expressions to extract data more easily from the result set.

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

Note

The `getIterator()` method is still supported to allow for a smooth transition to version 3, but we encourage you to migrate your code to use paginators.

Many Higher-Level Abstractions Have Changed

In general, many of the higher-level abstractions (service-specific helper objects, aside from the clients) have been improved or updated. Some have even been removed.

- **Updated:**
  - The way you use Amazon S3 Multipart Upload has changed. Amazon S3 Glacier Multipart Upload has been changed in similar ways.
  - The way to create Amazon S3 pre-signed URLs has changed.
  - The `Aws\S3\Sync` namespace has been replaced by the `Aws\S3\Transfer` class. The `S3Client::uploadDirectory()` and `S3Client::downloadBucket()` methods are still available, but have different options. See the documentation for Amazon S3 Transfer Manager with AWS SDK for PHP Version 3.
  - `Aws\S3\Model\ClearBucket` and `Aws\S3\Model\DeleteObjectsBatch` have been replaced by `Aws\S3\BatchDelete` and `S3Client::deleteMatchingObjects()`.

  - The options and behaviors for the Using the DynamoDB Session Handler with AWS SDK for PHP Version 3 have changed slightly.
  - The `Aws\DynamoDb\Model\BatchRequest` namespace has been replaced by `Aws\DynamoDb\WriteRequestBatch`. See the documentation for DynamoDB WriteRequestBatch.
  - The `Aws\Ses\SesClient` now handles base64 encoding the `RawMessage` when using `SendRawEmail` operation.
- **Removed:**
  - Amazon DynamoDBItem, `Attribute`, and `ItemIterator` classes - These were previously deprecated in Version 2.7.0.
  - Amazon SNS message validator - This is now a separate, lightweight project that does not require the SDK as a dependency. This project is, however, included in the Phar and ZIP distributions of the SDK. You can find a getting started guide on the AWS PHP Development blog.
  - Amazon `S3AcpBuilder` and related objects were removed.

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

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

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(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'profile' => 'my_profile',
                'region'  => 'us-east-1'
            )
        ),
        'dynamodb' => array(
            'extends' => 'dynamodb',
            'params' => array(
                'region'  => 'us-west-2'
            )
        ),
    )
);
```

```
<?php

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

use Aws\Common\Aws;

$aws = Aws::factory('path/to/my/config.php');

$sqs = $aws->get('sqs');
// Note: SQS client will be configured for us-east-1.

$dynamodb = $aws->get('dynamodb');
// Note: DynamoDB client will be configured for us-west-2.
```

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 `create<Service>()` methods instead of `get('<service>')`.

```
<?php

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

$sdk = new Aws\Sdk([
    'profile' => 'my_profile',
    'region' => 'us-east-1',
    'version' => 'latest',
    'DynamoDb' => [
        'region' => 'us-west-2',
    ],
]);

$sqs = $sdk->createSqs();
// Note: Amazon SQS client will be configured for us-east-1.

$dynamodb = $sdk->createDynamoDb();
// Note: DynamoDB client will be configured for us-west-2.
```

Shared `config` and `credentials` files

The shared AWS `config` and `credentials` files are the most common way that you can specify authentication and configuration for the AWS SDK for PHP. Use these files to store settings that your tools and applications can use across the AWS SDKs and the AWS Command Line Interface.

The shared AWS `config` and `credentials` files are plaintext files that reside by default in a folder named `.aws` that is placed in the "home" folder on your computer. For details on the location of these files, see Location of the shared `config` and `credentials` files in the *AWS SDKs and Tools Reference Guide*.

For all settings that you can store in these files, see Configuration and authentication settings reference in the *AWS SDKs and Tools Reference Guide*. This reference also covers the precedence of applying settings from alternative sources such as environment variables.

Named profiles

Settings within the shared `config` and `credentials` files are associated with a specific profile. With multiple profiles, you can create different settings configurations to apply in different scenarios. One of the profiles is designated as the `default` profile and is used automatically when you don't explicitly specify a profile to use.

To learn more about setting up named profiles, see Shared `config` and `credentials` files in the *AWS SDKs and Tools Reference Guide*.

You can specify a named profile to use when instantiating a client by using the `profile` option:

```
use Aws\DynamoDb\DynamoDbClient;

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

Work with AWS services in the AWS SDK for PHP

The following sections contain examples, tutorials, tasks, and guides that show you how to use the AWS SDK for PHP to work with AWS services.

Topics

Use features and options of the AWS SDK for PHP Version 3

The AWS SDK for PHP Version 3 provides support for additional features and options to work with AWS service APIs. The sections in this topic cover these options by service.

Topics

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.

```
use Aws\DynamoDb\SessionHandler;

$dynamoDb = new Aws\DynamoDb\DynamoDbClient([
    'region'=>'us-east-1'  // Since version 3.277.10 of the SDK,
]);                        // the 'version' parameter defaults to 'latest'.

$sessionHandler = SessionHandler::fromClient($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.

```
// 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.

```
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name'                    => 'sessions',
    'hash_key'                      => 'id',
    'data_attribute'                => 'data',
    'data_attribute_type'           => 'string',
    'session_lifetime'              => 3600,
    'session_lifetime_attribute'    => 'expires',
    'consistent_read'               => true,
    'locking'                       => false,
    'batch_config'                  => [],
    'max_lock_wait_time'            => 10,
    'min_lock_retry_microtime'      => 5000,
    'max_lock_retry_microtime'      => 50000,
]);
```

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.

Read via `session_start()`	• 1 read operation (only 0.5 if `consisten t_read` is `false`). • (Conditional) 1 write operation to delete the session if it is expired.
Read via `session_start()` (Using session locking)	• A minimum of 1 *write* operation. • (Conditional) Additional write operations for each attempt at acquiring a lock on the session. Based on configured lock wait time and retry options. • (Conditional) 1 write operation to delete the session if it is expired.
Write via `session_write_close()`	• 1 write operation.
Delete via `session_destroy()`	• 1 write operation.
Garbage Collection	• 0.5 read operations **per 4 KB of data in the table** to scan for expired sessions. • 1 write operation **per expired item** to delete it.

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`.

```
$sessionHandler = SessionHandler::fromClient($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.

```
$sessionHandler = SessionHandler::fromClient($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.

```
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name'   => 'sessions',
    'batch_config' => [
        '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 SessionHhandler, your configured credentials must have permission to use the DynamoDB table that you created in a previous step. 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.

```
{
  "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 features and options

This topic discusses additional features and options provided by AWS SDK for PHP Version 3 to work with Amazon S3.

Topics

- Amazon S3 multi-Region client with AWS SDK for PHP Version 3
- Amazon S3 stream wrapper with AWS SDK for PHP Version 3
- Amazon S3 transfer manager with AWS SDK for PHP Version 3
- Amazon S3 client-side encryption with the AWS SDK for PHP Version 3
- Amazon S3 checksums with AWS SDK for PHP Version 3

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, 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 redispatch 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.

Download 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.

```
// 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.

```
// 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.

Open 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.

```
$context = stream_context_create([
    's3' => ['seekable' => true]
]);

if ($stream = fopen('s3://bucket/key', 'r', false, $context)) {
    // Read bytes from the stream
    fread($stream, 1024);
    // Seek back to the beginning of the stream
    fseek($stream, 0);
    // Read the same bytes that were previously read
    fread($stream, 1024);
    fclose($stream);
}
```

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.

Upload data

You can upload data to Amazon S3 using `file_put_contents()`.

```
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.

```
$stream = fopen('s3://bucket/key', 'w');
fwrite($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 for streams that target Amazon S3 objects.

r	A read-only stream where the object must already exist.
w	A write-only stream. If the object already exists, it is overwritten.
a	A write-only stream. If the object already exists, it is downloaded to a temporary stream and any writes to the stream is appended to any previously uploaded data.
x	A write-only stream. An error is raised if the object already exist.

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.

unlink()	Delete an object from a bucket.
	`// Delete an object from a bucket`
	`unlink('s3://bucket/key');`
	You can pass in any options available to the `DeleteObject` operation to modify how the object is deleted (e.g. specifying a specific object version).
	`// Delete a specific version of an object from a bucket`
	`unlink('s3://bucket/key', stream_context_create([`
	`'s3' => ['VersionId' => '123']`
	`]);`

filesize()	Get the size of an object.
	`// Get the Content-Length of an object`
	`$size = filesize('s3://bucket/key', );`

is_file()	Checks if a URL is a file.
	`if (is_file('s3://bucket/key')) {`
	`echo 'It is a file!';`
	`}`

file_exists()	Checks if an object exists.
	`if (file_exists('s3://bucket/key')) {`
	`echo 'It exists!';`
	`}`

filetype()	Checks if a URL maps to a file or bucket (dir).

file()	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.
filemtime()	Get the last modified date of an object.
rename()	Rename an object by copying the object then deleting the original. You can pass in options available to the `CopyObject` and `DeleteObject` operations to the stream context parameters to modify how the object is copied and deleted.

> **Note**
>
> Although `copy` generally works with the Amazon S3 stream wrapper, some errors might not be properly reported due to the internals of the `copy` function in PHP. We recommend that you use an instance of AwsS3ObjectCopier instead.

Work with buckets and folders

Use `mkdir()` to work with buckets

You can create and browse Amazon S3 buckets similarly to how PHP allows you to create and traverse directories on your file system.

Here's an example that creates a bucket.

```
mkdir('s3://my-bucket');
```

> **Note**
>
> In April 2023, Amazon S3 automatically enabled S3 Block Public Access and disabled access control lists for all newly created buckets. This change also affects how the

`StreamWrapper`'s `mkdir` function works with permissions and ACLs. More information is available in this What's New with AWS article.

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.

```
// Create a bucket in the EU (Ireland) Region
mkdir('s3://my-bucket', 0500, true,
        stream_context_create([
            's3' => ['LocationConstraint' => 'eu-west-1']
        ]));
```

You can delete buckets using the `rmdir()` function.

```
// Delete a bucket
rmdir('s3://my-bucket);
```

Note

A bucket can only be deleted if it is empty.

Use `mkdir()` to work with folders

After you create a bucket, you can use `mkdir()` to create objects that function as folders as in a file system.

The following code snippet adds a folder object named 'my-folder' to the existing bucket named 'my-bucket'. Use the forward slash (/) character to separate a folder object name from the bucket name and any additional folder name.

```
mkdir('s3://my-bucket/my-folder')
```

The previous note about permission changes after April 2023 also come into play when you create folder objects. This blog post has information about how to adjust permissions if needed.

Use the `rmdir()` function to delete an empty folder object as shown in the following snippet.

```
rmdir('s3://my-bucket/my-folder')
```

List 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.

```
$dir = "s3://bucket/";

if (is_dir($dir) && ($dh = opendir($dir))) {
    while (($file = readdir($dh)) !== false) {
        echo "filename: {$file} : filetype: " . filetype($dir . $file) . "\n";
    }
    closedir($dh);
}
```

You can recursively list each object and prefix in a bucket using PHP's RecursiveDirectoryIterator.

```
$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 `Aws\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 sourced 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.

```
$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.

```
// 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.

```
$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. The following transfer options are valid:

`add_content_md5` (bool)

Set to `true` to calculate the MD5 checksum for uploads.

`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.

```
$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).

```
$promise->otherwise(function ($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.

```
$uploader = new Transfer($s3Client, $source, $dest, [
    'before' => function (\Aws\Command $command) {
        // Commands can vary for multipart uploads, so check which command
        // is being processed.
        if (in_array($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.

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.

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.

```
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,
    '@KmsEncryptionContext' => ['context-key' => 'context-value'],
    'Bucket' => $bucket,
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);
```

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.

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.

```
$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 `S3EncryptionClientV2` and `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.

`'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.

`'Aad'` (string)

Optional 'Additional authentication data' to include with your encrypted payload. This information is validated on decryption. `Aad` is available only when using the 'gcm' cipher.

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 `Aws\Crypto\MetadataStrategyInterface`. This simple interface handles saving and loading the `Aws\Crypto\MetadataEnvelope` that contains your envelope encryption materials. The SDK provides two classes that implement this: `Aws\S3\Crypto\HeadersMetadataStrategy` and `Aws\S3\Crypto\InstructionFileMetadataStrategy`. `HeadersMetadataStrategy` is used by default.

```
$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,
]);
```

Class name constants for the `HeadersMetadataStrategy` and `InstructionFileMetadataStrategy` can also be supplied by invoking *::class*.

```
$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.

```
$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.

Amazon S3 checksums with AWS SDK for PHP Version 3

Amazon Simple Storage Service (Amazon S3) provides the ability to specify a checksum when you upload an object. When you specify a checksum, it is stored with the object and can be validated when the object is downloaded.

Checksums provide an additional layer of data integrity when you transfer files. With checksums, you can verify data consistency by confirming that the received file matches the original file. For more information about checksums with Amazon S3, see the Amazon Simple Storage Service User Guide.

Amazon S3 currently supports four checksum algorithms: SHA-1, SHA-256, CRC-32, and CRC-32C. You have the flexibility to choose the algorithm that best fits your needs and let the SDK calculate the checksum. Alternatively, you can specify their own pre-computed checksum value by using one of the four supported algorithms.

> **Important**
>
> To work with the CRC-32C algorithm, your PHP environment requires the installation of the AWS Common Runtime (AWS CRT) extension.

We discuss checksums in two request phases: uploading an object and downloading an object.

Upload an object

You upload objects to Amazon S3 by using the putObject method of the `S3Client`. Use the `ChecksumAlgorithm` pair in the parameter array to enable checksum computation and specify the algorithm. Valid values for the algorithm are `CRC32`, `CRC32C`, `SHA1`, and `SHA256`.

The following code snippet shows a request to upload an object with a CRC-32 checksum. When the SDK sends the request, it calculates the CRC-32 checksum and uploads the object. Amazon S3 stores the checksum with the object.

```
$client = new \Aws\S3\S3Client(['region' => 'us-east-2']); // See the note below.
$result = $client->putObject([
        'Bucket' => 'bucketname',
        'Key' => 'key',
        'ChecksumAlgorithm' => 'CRC32',
        'Body' => 'Object contents to test the checksum.'
]);
```

If the checksum that the SDK calculates doesn't match the checksum that Amazon S3 calculates when it receives the request, an error is returned.

> **Note**
>
> Since version 3.277.10 of the SDK for PHP, `'version'` is no longer a required parameter in a service client constructor. If `'version'` is not provided, the value defaults to `'latest'`.

Use a pre-calculated checksum value

A pre-calculated checksum value provided with the request disables automatic computation by the SDK and uses the provided value instead.

The following example shows a request with a pre-calculated SHA-256 checksum.

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

$client = new S3Client([
    'region' => 'us-east-1',
]);
```

```
// Calculate the SHA-256 checksum of the contents to be uploaded.
$contents = 'Object contents to test the checksum.';
$body = Psr7\Utils::streamFor($contents);
$sha256 = base64_encode(Psr7\Utils::hash($body, 'sha256', true));

$result = $client->putObject([
    'Bucket' => 'bucketname',
    'Key' => 'key',
    'Body' => $body,
    'ChecksumSHA256' => $sha256
]);
```

If Amazon S3 determines the checksum value is incorrect for the specified algorithm, the service returns an error response.

Multipart uploads

You can also use checksums with multipart uploads. As shown in the following example, specify the checksum algorithm as a key-value pair in the `params` array of the MultipartUploader constructor.

```
$s3Client = new S3Client([
    'region' => 'us-east-1'
]);

$stream = fopen("/path/to/large/file", "r");

$mpUploader = new MultipartUploader($s3Client, $stream, [
    'bucket' => 'bucketname',
    'key' => 'key',
    'params' => ['ChecksumAlgorithm' => 'CRC32']
]);
```

Download an object

When you use the getObject method to download an object, the SDK automatically validates the checksum when the `ChecksumMode` key's value is `enabled`.

The request in the following snippet directs the SDK to validate the checksum in the response by calculating the checksum and comparing the values.

```
$result = $client->getObject([
```

```
        'Bucket' => 'remiss-us-east-2',
        'Key' => 'test-checksum',
        'ChecksumMode' => 'enabled',
]);
```

If the object wasn't uploaded with a checksum, no validation takes place.

An object in Amazon S3 can have multiple checksums, but only one checksum is validated on download. The following precedence— based on the efficiency of the checksum algorithm— determines which checksum the SDK validates:

1. CRC-32C
2. CRC-32
3. SHA-1
4. SHA-256

For example, if a response contains both CRC-32 and SHA-256 checksums, only the CRC-32 checksum is validated.

Code examples with guidance for the AWS SDK for PHP

This section contains code examples that demonstrate common AWS scenarios that use the AWS SDK for PHP.

All the example code for the AWS SDK for PHP is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Topics

- Amazon CloudFront examples using the AWS SDK for PHP Version 3
- Signing custom Amazon CloudSearch domain requests with AWS SDK for PHP Version 3
- Amazon CloudWatch examples using the AWS SDK for PHP Version 3
- Amazon EC2 examples using the AWS SDK for PHP Version 3

- Signing an Amazon OpenSearch Service search request with AWS SDK for PHP Version 3
- AWS Identity and Access Management examples using the AWS SDK for PHP Version 3
- AWS Key Management Service examples using the AWS SDK for PHP Version 3
- Amazon Kinesis examples using the AWS SDK for PHP Version 3
- AWS Elemental MediaConvert examples using the AWS SDK for PHP Version 3
- Amazon S3 examples using the AWS SDK for PHP Version 3
- Managing secrets using the Secrets Manager API and the AWS SDK for PHP Version 3
- Amazon SES examples using the AWS SDK for PHP Version 3
- Amazon SNS examples using the AWS SDK for PHP Version 3
- Amazon SQS examples using the AWS SDK for PHP Version 3
- Send events to Amazon EventBridge global endpoints

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. F or 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
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],
        'MaxTTL' => 0,
        'MinTTL' => 0,
        'SmoothStreaming' => false,
        'TargetOriginId' => $originName,
        'TrustedSigners' => [
            'Enabled' => false,
            'Quantity' => 0
        ],
        'ViewerProtocolPolicy' => 'allow-all'
    ];
    $enabled = false;
    $origin = [
        'Items' => [
            [
                'DomainName' => $s3BucketURL,
                'Id' => $originName,
                'OriginPath' => '',
                'CustomHeaders' => ['Quantity' => 0],
                'S3OriginConfig' => ['OriginAccessIdentity' => '']
            ]
        ],
        'Quantity' => 1
    ];
    $distribution = [
        'CallerReference' => $callerReference,
        'Comment' => $comment,
        'DefaultCacheBehavior' => $defaultCacheBehavior,
        'Enabled' => $enabled,
        'Origins' => $origin
    ];

    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient([
```

```
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    echo createS3Distribution($cloudFrontClient, $distribution);
}

// Uncomment the following line to run this code in an AWS account.
// createsTheS3Distribution();
```

Retrieve a CloudFront distribution

To retrieve the status and details of a specified CloudFront distribution, use the GetDistribution operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
function getDistribution($cloudFrontClient, $distributionId)
{
    try {
        $result = $cloudFrontClient->getDistribution([
            'Id' => $distributionId
        ]);

        $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;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
function listDistributions($cloudFrontClient)
{
```

```
    try {
        $result = $cloudFrontClient->listDistributions([]);
        return $result;
    } catch (AwsException $e) {
        exit('Error: ' . $e->getAwsErrorMessage());
    }
}

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'] . '.' . "\n";
        }
    }
}

// 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'];
    } 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 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\Exception\AwsException;
```

Sample Code

```
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']
["DefaultCacheBehavior"],
        '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']
    );
}

// Uncomment the following line to run this code in an AWS account.
// disableADistribution();
```

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
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';
```

```
    $cloudFrontClient = new Aws\CloudFront\CloudFrontClient([
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    // To delete 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']);
    } else {
        echo deleteDistribution(
            $cloudFrontClient,
            $distributionId,
            $eTag['ETag']
        );
    }
}

// Uncomment the following line to run this code in an AWS account.
// deleteADistribution();
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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.

Imports

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

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 = '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

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

use Aws\Exception\AwsException;
```

Sample Code

```
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

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

use Aws\Exception\AwsException;
```

Sample Code

```
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'] . '.' . "\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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```
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 = 'AAPKAJIKZATYYYEXAMPLE';

    $cloudFrontClient = new CloudFrontClient([
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    echo signPrivateDistribution(
        $cloudFrontClient,
        $resourceKey,
        $expires,
        $privateKey,
        $keyPairId
    );
}

// Uncomment the following line to run this code in an AWS account.
// signAPrivateDistribution();
```

Use a custom policy when creating CloudFront URLs

To use a custom policy, provide the `policy` key instead of `expires`.

Imports

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

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```
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",
            "Condition": {
                "IpAddress": {"AWS:SourceIp": "{$_SERVER['REMOTE_ADDR']}/32"},
                "DateLessThan": {"AWS:EpochTime": $expires}
            }
        }
    ]
}
POLICY;
    $privateKey = dirname(__DIR__) . '/cloudfront/my-private-key.pem';
```

```
    $keyPairId = 'AAPKAJIKZATYYYEXAMPLE';

    $cloudFrontClient = new CloudFrontClient([
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    echo signPrivateDistributionPolicy(
        $cloudFrontClient,
        $resourceKey,
        $customPolicy,
        $privateKey,
        $keyPairId
    );
}

// Uncomment the following line to run this code in an AWS account.
// signAPrivateDistributionPolicy();
```

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>
<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({
            file: "<?= $streamHostUrl ?>/cfx/st/<?= $signedUrlCannedPolicy ?>",
            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

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

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```
function signCookie(
    $cloudFrontClient,
    $resourceKey,
    $expires,
    $privateKey,
    $keyPairId
) {
    try {
        $result = $cloudFrontClient->getSignedCookie([
            'url' => $resourceKey,
            'expires' => $expires,
            'private_key' => $privateKey,
            'key_pair_id' => $keyPairId
        ]);
```

```
        return $result;
    } catch (AwsException $e) {
        return [ 'Error' => $e->getAwsErrorMessage() ];
    }
}

function signACookie()
{
    $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 = 'AAPKAJIKZATYYYEXAMPLE';

    $cloudFrontClient = new CloudFrontClient([
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    $result = signCookie(
        $cloudFrontClient,
        $resourceKey,
        $expires,
        $privateKey,
        $keyPairId
    );

    /* If successful, returns something like:
    CloudFront-Expires = 1589926678
    CloudFront-Signature = Lv1DyC2q...2HPXaQ__
    CloudFront-Key-Pair-Id = AAPKAJIKZATYYYEXAMPLE
    */
    foreach ($result as $key => $value) {
        echo $key . ' = ' . $value . "\n";
    }
}

// Uncomment the following line to run this code in an AWS account.
// signACookie();
```

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

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

use Aws\CloudFront\CloudFrontClient;
use Aws\Exception\AwsException;
```

Sample Code

```
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 = 'AAPKAJIKZATYYYEXAMPLE';

    $cloudFrontClient = new CloudFrontClient([
        'profile' => 'default',
        'version' => '2018-06-18',
        'region' => 'us-east-1'
    ]);

    $result = signCookiePolicy(
        $cloudFrontClient,
        $customPolicy,
        $privateKey,
        $keyPairId
    );

    /* If successful, returns something like:
    CloudFront-Policy = eyJTdGF0...fX19XX0_
    CloudFront-Signature = RowqEQWZ...N8vetw__
    CloudFront-Key-Pair-Id = AAPKAJIKZATYYYEXAMPLE
    */
    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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Sign Amazon CloudSearch 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 > 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 = '7kbitd6nyiglhdtmssxEXAMPLE';
    $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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Topics

- Working with Amazon CloudWatch alarms with AWS SDK for PHP Version 3
- Getting metrics from Amazon CloudWatch with AWS SDK for PHP Version 3
- Publishing custom metrics in Amazon CloudWatch with AWS SDK for PHP Version 3
- Sending events to Amazon CloudWatch Events with AWS SDK for PHP Version 3
- Using alarm actions with Amazon CloudWatch alarms with AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Describe alarms

Imports

```
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

```
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 {
            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.
```

```
// putTheMetricAlarm();
```

Delete alarms

Imports

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

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code

```
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.
```

```
// deleteTheAlarms();
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

List metrics

Imports

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

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code

```
function listMetrics($cloudWatchClient)
{
    try {
        $result = $cloudWatchClient->listMetrics();

        $message = '';

        if (isset($result['@metadata']['effectiveUri'])) {
            $message .= 'For the effective URI at ' .
                $result['@metadata']['effectiveUri'] . ":\n\n";
```

```
            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'] . ":\n";

                    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'] . ":\n\n";

            if (
                (isset($result['MetricAlarms'])) and
```

```
                (count($result['MetricAlarms']) > 0)
            ) {
                $message .= 'Matching alarms for ' . $metricName . ":\n\n";

                foreach ($result['MetricAlarms'] as $alarm) {
                    $message .= $alarm['AlarmName'] . "\n";
                }
            } else {
                $message .= 'No matching alarms found for ' . $metricName . '.';
            }
        } else {
            $message .= 'No matching alarms found for ' . $metricName . '.';
        }

        return $message;
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

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;
```

Sample Code

```
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\n";

            if (
                (isset($result['Datapoints'])) and
                (count($result['Datapoints']) > 0)
            ) {
                $message .= "Datapoints found:\n\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.';
        }

        return $message;
    } 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'
        ],
        [
            'Name' => 'Type',
            'Value' => 'Resource'
        ],
        [
            'Name' => 'Class',
            'Value' => 'Standard/OnDemand'
        ]
    ];
    $startTime = strtotime('-3 hours');
    $endTime = strtotime('now');
    $period = 300; // Seconds. (5 minutes = 300 seconds.)
    $statistics = ['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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Publish metric data

Imports

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

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code

```
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,
            '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.
// putTheMetricAlarm();
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create a rule

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$client = new Aws\cloudwatchevents\cloudwatcheventsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2015-10-07'
]);

try {
    $result = $client->putRule([
        'Name' => 'DEMO_EVENT', // REQUIRED
        'RoleArn' => 'IAM_ROLE_ARN',
        'ScheduleExpression' => 'rate(5 minutes)',
        'State' => 'ENABLED',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Add targets to a rule

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

```

require 'vendor/autoload.php';

use Aws\Exception\AwsException;
```

Sample Code

```
$client = new Aws\cloudwatchevents\cloudwatcheventsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2015-10-07'
]);
```

```
try {
    $result = $client->putEvents([
        'Entries' => [ // REQUIRED
            [
                'Detail' => '<string>',
                'DetailType' => '<string>',
                'Resources' => ['<string>'],
                'Source' => '<string>',
                'Time' => time()
            ],
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Enable alarm actions

Imports

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

use Aws\CloudWatch\CloudWatchClient;
use Aws\Exception\AwsException;
```

Sample Code

```
function enableAlarmActions($cloudWatchClient, $alarmNames)
{
    try {
        $result = $cloudWatchClient->enableAlarmActions([
            'AlarmNames' => $alarmNames
        ]);

        if (isset($result['@metadata']['effectiveUri'])) {
            return 'At the effective URI of ' .
                $result['@metadata']['effectiveUri'] .
                ', actions for any matching alarms have been enabled.';
        } else {
            return'Actions for some matching alarms ' .
                'might not have been enabled.';
        }
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

function enableTheAlarmActions()
{
    $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();
```

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 disableAlarmActions($cloudWatchClient, $alarmNames);
}

// Uncomment the following line to run this code in an AWS account.
// disableTheAlarmActions();
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Topics

- Managing Amazon EC2 instances using the AWS SDK for PHP Version 3
- Using Elastic IP addresses with Amazon EC2 with AWS SDK for PHP Version 3
- Using regions and availability zones for Amazon EC2 with AWS SDK for PHP Version 3
- Working with Amazon EC2 key pairs with AWS SDK for PHP Version 3
- Working with security groups in Amazon EC2 with AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Describe instances

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 = $ec2Client->describeInstances();
echo "Instances: \n";
foreach ($result['Reservations'] as $reservation) {
    foreach ($reservation['Instances'] as $instance) {
        echo "InstanceId: {$instance['InstanceId']} - {$instance['State']['Name']} \n";
    }
}
```

Enable and disable monitoring

Imports

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$instanceIds = ['InstanceID1', 'InstanceID2'];

$monitorInstance = 'ON';

if ($monitorInstance == 'ON') {
    $result = $ec2Client->monitorInstances([
        'InstanceIds' => $instanceIds
    ]);
} else {
    $result = $ec2Client->unmonitorInstances([
        'InstanceIds' => $instanceIds
    ]);
}

var_dump($result);
```

Start and stop an instance

Imports

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);
```

```
$action = 'START';

$instanceIds = ['InstanceID1', 'InstanceID2'];

if ($action == 'START') {
    $result = $ec2Client->startInstances([
        'InstanceIds' => $instanceIds,
    ]);
} else {
    $result = $ec2Client->stopInstances([
        'InstanceIds' => $instanceIds,
    ]);
}

var_dump($result);
```

Reboot an instance

Imports

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$instanceIds = ['InstanceID1', 'InstanceID2'];

$result = $ec2Client->rebootInstances([
    'InstanceIds' => $instanceIds
]);

var_dump($result);
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Describe an instance

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 = $ec2Client->describeInstances();
echo "Instances: \n";
foreach ($result['Reservations'] as $reservation) {
```

```
    foreach ($reservation['Instances'] as $instance) {
        echo "InstanceId: {$instance['InstanceId']} - {$instance['State']['Name']} \n";
    }
}
```

Allocate and associate an address

Imports

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

Sample Code

```
$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

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$associationID = 'AssociationID';

$allocationID = 'AllocationID';

$result = $ec2Client->disassociateAddress([
    'AssociationId' => $associationID,
]);

$result = $ec2Client->releaseAddress([
    '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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Describe availability zones

Imports

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

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

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);
```

```
$result = $ec2Client->describeRegions();

var_dump($result);
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create a key pair

Imports

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$keyPairName = 'my-keypair';
```

```
$result = $ec2Client->createKeyPair(array(
    'KeyName' => $keyPairName
));

// Save the private key
$saveKeyLocation = getenv('HOME') . "/.ssh/{$keyPairName}.pem";
file_put_contents($saveKeyLocation, $result['keyMaterial']);

// Update the key's permissions so it can be used with SSH
chmod($saveKeyLocation, 0600);
```

Delete a key pair

Imports

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

Sample Code

```
$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

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$result = $ec2Client->describeKeyPairs();

var_dump($result);
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Describe security groups

Imports

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

Sample Code

```
$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

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

Sample Code

```
$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

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

Sample Code

```
$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

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

Sample Code

```
$ec2Client = new Aws\Ec2\Ec2Client([
    'region' => 'us-west-2',
```

```
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$securityGroupId = 'my-security-group-id';

$result = $ec2Client->deleteSecurityGroup([
    'GroupId' => $securityGroupId
]);

var_dump($result);
```

Signing an Amazon OpenSearch Service search request with AWS SDK for PHP Version 3

Amazon 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. 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 list of Regions supported by OpenSearch Service can be found on the AWS Regions

and Endpoints page in the Amazon Web Services General Reference. However, in this example, we sign requests against an OpenSearch Service domain in the `us-west-2` region.

You need to provide credentials, which you can do either with the SDK's default provider chain or with any form of credentials described in Credentials for the AWS SDK for PHP Version 3. You'll also need a PSR-7 request (assumed in the code below to be named `$psr7Request`).

```
// Pull credentials from the default provider chain
$provider = Aws\Credentials\CredentialProvider::defaultProvider();
$credentials = call_user_func($provider)->wait();

// Create a signer with the service's signing name and Region
$signer = new Aws\Signature\SignatureV4('es', 'us-west-2');

// Sign your request
$signedRequest = $signer->signRequest($psr7Request, $credentials);
```

AWS Identity and Access Management examples using the AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Topics

- Managing IAM access keys with AWS SDK for PHP Version 3
- Managing IAM users with AWS SDK for PHP Version 3
- Using IAM account aliases with AWS SDK for PHP Version 3
- Working with IAM policies with AWS SDK for PHP Version 3

- Working with IAM server certificates with AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create an access key

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$client = new IamClient([
    'profile' => 'default',
```

```
    'region' => 'us-west-2',
    'version' => '2010-05-08'
]);

try {
    $result = $client->createAccessKey([
        'UserName' => 'IAM_USER_NAME',
    ]);
    $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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

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\Exception\AwsException;
use Aws\Iam\IamClient;
```

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\Exception\AwsException;
use Aws\Iam\IamClient;
```

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\Exception\AwsException;
use Aws\Iam\IamClient;
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create an IAM user

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->updateUser([
        // UserName is required
        'UserName' => 'string1',
        'NewUserName' => 'string'
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
```

```
}
```

Get information about an IAM user

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->getUser([
        'UserName' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete an IAM user

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->deleteUser([
        // UserName is required
        'UserName' => 'string'
    ]);
    var_dump($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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create an alias

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->listAccountAliases();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete an alias

Imports

```

require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->deleteAccountAlias([
        // AccountAlias is required
        'AccountAlias' => 'string',
    ]);
    var_dump($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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Create a policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

$myManagedPolicy = '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "RESOURCE_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Scan",
            "dynamodb:UpdateItem"
        ],
            "Resource": "RESOURCE_ARN"
        }
    ]
}';

try {
    $result = $client->createPolicy(array(
        // PolicyName is required
        'PolicyName' => 'myDynamoDBPolicy',
        // PolicyDocument is required
        'PolicyDocument' => $myManagedPolicy
    ));
```

```
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Attach a policy to a role

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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 is required
        'RoleName' => $roleName,
        // PolicyArn is required
        'PolicyArn' => $policyArn
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Attach a policy to a user

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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 is required
        'UserName' => $userName,
        // PolicyArn is required
        'PolicyArn' => $policyArn,
    ));
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Attach a policy to a group

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->detachUserPolicy([
        // UserName is required
        'UserName' => 'string',
        // PolicyArn is required
        'PolicyArn' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Detach a group policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->detachGroupPolicy([
        // GroupName is required
        'GroupName' => 'string',
        // PolicyArn is required
        'PolicyArn' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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 (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a role policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
```

```
    $result = $client->deleteRolePolicy([
        // RoleName is required
        'RoleName' => 'string',
        // PolicyName is required
        'PolicyName' => 'string'
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a user policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->deleteUserPolicy([
        // UserName is required
        'UserName' => 'string',
        // PolicyName is required
        'PolicyName' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
```

```
}
```

Delete a group policy

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

List server certificates

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->getServerCertificate([
        // ServerCertificateName is required
        'ServerCertificateName' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Update a server certificate

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

try {
    $result = $client->updateServerCertificate([
        // ServerCertificateName is required
        'ServerCertificateName' => 'string',
        'NewServerCertificateName' => 'string',
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a server certificate

Imports

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

use Aws\Exception\AwsException;
use Aws\Iam\IamClient;
```

Sample Code

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

```
try {
    $result = $client->deleteServerCertificate([
        // ServerCertificateName is required
        'ServerCertificateName' => 'string',
    ]);
    var_dump($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 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
- Encrypting and decrypting AWS KMS data keys using the AWS SDK for PHP Version 3
- Working with AWS KMS key policies using the AWS SDK for PHP Version 3
- Working with grants using the AWS KMS API and the AWS SDK for PHP version 3
- Working with aliases using the AWS KMS API and the AWS SDK for PHP Version 3

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 AWS KMS keys. You can use a KMS key to encrypt your data.

The following examples show how to:

- Create a customer KMS key using CreateKey.

- Generate a data key using GenerateDataKey.
- View a KMS key using DescribeKey.
- Get key IDs and key ARNS of KMS keys using ListKeys.
- Enable KMS keys using EnableKey.
- Disable KMS keys using DisableKey.

All the example code for the AWS SDK for PHP is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

Create a KMS key

To create a KMS key, use the CreateKey operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 AWS KMS key under which to generate the data key.

Imports

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

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';
$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 KMS key

To get detailed information about a KMS key, including the KMS key'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

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

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';

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 KMS key

To get the ID and ARN of the KMS key, use the ListAliases operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 KMS key

To enable a disabled KMS key, use the EnableKey operation.

Imports

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

```
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';

try {
    $result = $KmsClient->enableKey([
        'KeyId' => $keyId,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Disable a KMS key

To disable a KMS key, use the DisableKey operation. Disabling a KMS key prevents it from being used.

Imports

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

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';

try {
    $result = $KmsClient->disableKey([
        'KeyId' => $keyId,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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 an AWS Key Management Service's (AWS KMS) AWS KMS key to generate, encrypt, and decrypt data keys.

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 KMS key using ReEncrypt.

All the example code for the AWS SDK for PHP is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 `Encypt` method when you're moving encrypted data to a new AWS Region and want to encrypt its data key by using a KMS key in the new Region.

Imports

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

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';
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 KMS key, 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

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

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';
$ciphertextBlob = 'Place your cipher text blob here';

try {
    $result = $KmsClient->reEncrypt([
        'CiphertextBlob' => $ciphertextBlob,
        'DestinationKeyId' => $keyId,
    ]);
    var_dump($result);
} 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 KMS key, you determine who can use and manage that KMS key. 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 KMS key, but you cannot edit the key policy for an AWS managed KMS key. 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 KMS key, use the `ListKeyPolicies` operation.

Imports

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

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->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 KMS key, use the `GetKeyPolicy` operation.

`GetKeyPolicy` requires a policy name. Unless you created a key policy when you created the KMS key, the only valid policy name is the default. Learn more about the Default key policy in the *AWS Key Management Service Developer Guide*.

Imports

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

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';
$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 KMS key, use the `PutKeyPolicy` operation.

`PutKeyPolicy` requires a policy name. Unless you created a Key Policy when you created the KMS key, the only valid policy name is the default. Learn more about the Default key policy in the *AWS Key Management Service Developer Guide*.

Imports

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

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';
$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
    echo $e->getMessage();
    echo "\n";
}
```

Working with grants using the AWS KMS API and the AWS SDK for PHP version 3

A grant is another mechanism for providing permissions. It is 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 AWS KMS keys. For more information, see Grants in AWS KMS in the *AWS Key Management Service Developer Guide*.

The following examples show how to:

- Create a grant for a KMS key using CreateGrant.
- View a grant for a KMS key using ListGrants.
- Retire a grant for a KMS key using RetireGrant.

- Revoke a grant for a KMS key using RevokeGrant.

All the example code for the AWS SDK for PHP is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 key, use the CreateGrant operation.

Imports

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

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';
$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 key, use the ListGrants operation.

Imports

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

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 key, use the RetireGrant operation. Retire a grant to clean up after you finish using it.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 key, use the RevokeGrant operation. You can revoke a grant to explicitly deny operations that depend on it.

Imports

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

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';
$grantId = "grant1";

try {
    $result = $KmsClient->revokeGrant([
        'KeyId' => $keyId,
        'GrantId' => $grantId,
    ]);
    var_dump($result);
} catch (AwsException $e) {
```

```
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Working with aliases using the AWS KMS API and the AWS SDK for PHP Version 3

AWS Key Management Service (AWS KMS) provides an optional display name for an AWS KMS key called an alias.

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using AWS Key Management Service (AWS KMS), see the AWS KMS Developer Guide.

Create an alias

To create an alias for a KMS key, use the CreateAlias operation. The alias must be unique in the account and AWS Region. If you create an alias for a KMS key that already has an alias, `CreateAlias` creates another alias to the same KMS key. It doesn't replace the existing alias.

Imports

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

```
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';
$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 in the caller's AWS account and AWS Region, use the ListAliases operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 KMS key, use the UpdateAlias operation.

Imports

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

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';
$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 KMS key.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$KmsClient = new Aws\Kms\KmsClient([
    'profile' => 'default',
    'version' => '2014-11-01',
    'region' => 'us-east-2'
]);

$aliasName = "alias/projectKey1";

try {
    $result = $KmsClient->deleteAlias([
        'AliasName' => $aliasName,
    ]);
    var_dump($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 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

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 in the Amazon Kinesis Developer Guide.

To create a Kinesis data stream, use the CreateStream operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$kinesisClient = new Aws\Kinesis\KinesisClient([
    'profile' => 'default',
    'version' => '2013-12-02',
    'region' => 'us-east-2'
]);

$name = "my_stream_name";

try {
    $result = $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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$kinesisClient = new Aws\Kinesis\KinesisClient([
    'profile' => 'default',
    'version' => '2013-12-02',
    'region' => 'us-east-2'
]);

try {
    $result = $kinesisClient->listStreams();
    var_dump($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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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([
        'Data' => $content,
        'StreamName' => $name,
        'PartitionKey' => $groupID
    ]);
    print("<p>ShardID = " . $result["ShardId"] . "</p>");
    var_dump($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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using Amazon Kinesis Data Streams, see the Amazon Kinesis Data Streams Developer Guide.

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 Firehose API and the AWS SDK for PHP Version 3

Amazon 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using Amazon 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 Firehose.

Imports

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

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";
$stream_type = "KinesisStreamAsSource";
$kinesis_stream = "arn:aws:kinesis:us-east-2:0123456789:stream/my_stream_name";
$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 Firehose access to your Amazon S3 bucket, as described in Grant Kinesis Data Firehose Access to an Amazon S3 Destination.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$firehoseClient = new Aws\Firehose\FirehoseClient([
    'profile' => 'default',
    'version' => '2015-08-04',
    'region' => 'us-east-2'
]);

$name = "my_S3_stream_name";
$stream_type = "DirectPut";
$s3bucket = 'arn:aws:s3:::bucket_name';
$s3Role = 'arn:aws:iam::0123456789:policy/Role';

try {
    $result = $firehoseClient->createDeliveryStream([
        'DeliveryStreamName' => $name,
        'DeliveryStreamType' => $stream_type,
        'S3DestinationConfiguration' => [
            'BucketARN' => $s3bucket,
            'CloudWatchLoggingOptions' => [
                'Enabled' => false,
            ],
            'RoleARN' => $s3Role
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Create a delivery stream using OpenSearch Service

To establish a 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 Firehose access to your OpenSearch Service cluster, as described in Grant Kinesis Data Firehose Access to an Amazon ES Destination.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 {
    $result = $firehoseClient->createDeliveryStream([
        'DeliveryStreamName' => $name,
        'DeliveryStreamType' => $stream_type,
        'ElasticsearchDestinationConfiguration' => [
            'DomainARN' => $esDomainARN,
```

```
            'IndexName' => $esIndex,
            'RoleARN' => $esRole,
            'S3Configuration' => [
                'BucketARN' => $s3bucket,
                'CloudWatchLoggingOptions' => [
                    'Enabled' => false,
                ],
                'RoleARN' => $s3Role,
            ],
            'TypeName' => $esType,
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Retrieve a delivery stream

To get the details about an existing Firehose delivery stream, use the DescribeDeliveryStream operation.

Imports

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

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->describeDeliveryStream([
        'DeliveryStreamName' => $name,
    ]);
    var_dump($result);
} 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 Firehose delivery streams sending data to Kinesis Data Streams, use the ListDeliveryStreams operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 Firehose delivery streams sending data to Amazon S3, OpenSearch Service, or Amazon Redshift, or to Splunk, use the ListDeliveryStreams operation.

Imports

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

use Aws\Exception\AwsException;
```

Sample Code

```
$firehoseClient = new Aws\Firehose\FirehoseClient([
    '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 Firehose delivery stream

To send data through a Firehose delivery stream to your specified destination, use the PutRecord operation after you create a Firehose delivery stream.

Before sending data to a Firehose delivery stream, use `DescribeDeliveryStream` to see if the delivery stream is active.

Imports

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

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";
}
```

Delete a Firehose delivery stream

To delete a 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\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.

Create and manage transcoding jobs in AWS Elemental MediaConvert

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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.

Imports

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

use Aws\Exception\AwsException;
use Aws\MediaConvert\MediaConvertClient;
```

Sample Code

```
$mediaConvertClient = new MediaConvertClient([
    'version' => '2017-08-29',
    'region' => 'us-east-2',
```

```
    'profile' => 'default'
]);
```

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 **Show job JSON** at the bottom of the **Job** section. This example shows the JSON for a simple job.

Sample Code

```
$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"
            ]
        ]
    ]
],
"AdAvailOffset" => 0,
"Inputs" => [
    [
        "AudioSelectors" => [
            "Audio Selector 1" => [
                "Offset" => 0,
                "DefaultSelection" => "NOT_DEFAULT",
                "ProgramSelection" => 1,
                "SelectorType" => "TRACK",
                "Tracks" => [
                    1
                ]
            ]
        ],
        "VideoSelector" => [
            "ColorSpace" => "FOLLOW"
        ],
        "FilterEnable" => "AUTO",
        "PsiControl" => "USE_PSI",
        "FilterStrength" => 0,
        "DeblockFilter" => "DISABLED",
        "DenoiseFilter" => "DISABLED",
        "TimecodeSource" => "EMBEDDED",
        "FileInput" => "s3://INPUT_BUCKET_AND_FILE_NAME"
```

```
        ]
    ],
    "TimecodeConfig" => [
        "Source" => "EMBEDDED"
    ]
];
```

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

```
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

```
$mediaConvertClient = new MediaConvertClient([
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default'
]);
```

```
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

```
$mediaConvertClient = new MediaConvertClient([
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default'
]);

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

```
$mediaConvertClient = new MediaConvertClient([
    'version' => '2017-08-29',
    'region' => 'us-east-2',
    'profile' => 'default'
]);

try {
    $result = $mediaConvertClient->listJobs([
        'MaxResults' => 20,
        'Order' => 'ASCENDING',
        'Queue' => 'QUEUE_ARN',
        'Status' => 'SUBMITTED',
        // 'NextToken' => '<string>', //OPTIONAL To retrieve the twenty next most
 recent jobs
    ]);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Topics

- Creating and using Amazon S3 buckets with the AWS SDK for PHP Version 3
- Managing Amazon S3 bucket access permissions with the AWS SDK for PHP Version 3
- Configuring Amazon S3 buckets with the AWS SDK for PHP Version 3

- Using Amazon S3 multipart uploads with AWS SDK for PHP Version 3
- Amazon S3 pre-signed URL with AWS SDK for PHP Version 3
- Amazon S3 pre-signed POSTs with AWS SDK for PHP Version 3
- Using an Amazon S3 bucket as a static web host with AWS SDK for PHP Version 3
- Working with Amazon S3 bucket policies with the AWS SDK for PHP Version 3
- Using S3 access point ARNs the AWS SDK for PHP Version 3
- Use Amazon S3 Multi-Region Access Points with the AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Imports

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

use Aws\S3\S3Client;
```

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

```
$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

```
function createBucket($s3Client, $bucketName)
{
    try {
        $result = $s3Client->createBucket([
            'Bucket' => $bucketName,
        ]);
        return 'The bucket\'s location is: ' .
            $result['Location'] . '. ' .
            'The bucket\'s effective URI is: ' .
            $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e->getAwsErrorMessage();
    }
}

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

```
$USAGE = "\n" .
    "To run this example, supply the name of an S3 bucket and a file to\n" .
    "upload to it.\n" .
    "\n" .
    "Ex: php PutObject.php <bucketname> <filename>\n";

if (count($argv) <= 2) {
    echo $USAGE;
    exit();
}

$bucket = $argv[1];
$file_Path = $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' => $file_Path,
    ]);
} 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Get and set an access control list policy

Imports

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

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

Sample Code

```
// 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";
}
```

Configuring Amazon S3 buckets with the AWS SDK for PHP Version 3

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support in Amazon S3, you can build rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

For more information about using CORS configuration with an Amazon S3 bucket, see Cross-Origin Resource Sharing (CORS).

The following examples show how to:

- Get the CORS configuration for a bucket using GetBucketCors.
- Set the CORS configuration for a bucket using PutBucketCors.

All the example code for the AWS SDK for PHP is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

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

Sample Code

```
$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

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

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

Sample Code

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

try {
    $result = $client->putBucketCors([
        'Bucket' => $bucketName, // REQUIRED
        'CORSConfiguration' => [ // REQUIRED
            'CORSRules' => [ // REQUIRED
                [
                    'AllowedHeaders' => ['Authorization'],
                    'AllowedMethods' => ['POST', 'GET', 'PUT'], // REQUIRED
                    'AllowedOrigins' => ['*'], // REQUIRED
                    'ExposeHeaders' => [],
                    'MaxAgeSeconds' => 3000
                ],
            ],
        ]
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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.

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

use Aws\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\ObjectUploader;
use Aws\S3\S3Client;
```

Sample Code

```
// Create an S3Client.
$s3Client = new S3Client([
    'profile' => 'default',
    'region' => 'us-east-2',
    'version' => '2006-03-01'
]);

$bucket = 'your-bucket';
$key = 'my-file.zip';

// Use a stream instead of a 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);
        // If the SDK chooses a multipart upload, try again if there is an exception.
        // Unlike PutObject calls, multipart upload calls are not automatically
 retried.
    } catch (MultipartUploadException $e) {
        rewind($source);
        $uploader = new MultipartUploader($s3Client, $source, [
            'state' => $e->getState(),
        ]);
    }
} while (!isset($result));

fclose($source);
```

Configuration

The `ObjectUploader` object constructor accepts the following arguments:

`$client`

The `Aws\ClientInterface` object to use for performing the transfers. This should be an instance of `Aws\S3\S3Client`.

`$bucket`

(`string`, *required*) Name of the bucket to which the object is being uploaded.

`$key`

(`string`, *required*) Key to use for the object being uploaded.

`$body`

(`mixed`, *required*) Object data to upload. Can be a `StreamInterface`, a PHP stream resource, or a string of data to upload.

`$acl`

(`string`) Access control list (ACL) to set on the object being upload. Objects are private by default.

$options

An associative array of configuration options for the multipart upload. The following configuration options are valid:

add_content_md5

(bool) Set to true to automatically calculate the MD5 checksum for the upload.

mup_threshold

(int, *default*: int(16777216)) The number of bytes for the file size. If the file size exceeds this limit, a multipart upload is used.

before_complete

(callable) Callback to invoke before the CompleteMultipartUpload operation. The callback should have a function signature similar to: function (Aws\Command $command) {...}.

before_initiate

(callable) Callback to invoke before the CreateMultipartUpload operation. The callback should have a function signature similar to: function (Aws\Command $command) {...}.

before_upload

(callable) Callback to invoke before any PutObject or UploadPart operations. The callback should have a function signature similar to: function (Aws\Command $command) {...}.

concurrency

(int, *default*: int(3)) Maximum number of concurrent UploadPart operations allowed during the multipart upload.

part_size

(int, *default*: int(5242880)) Part size, in bytes, to use when doing a multipart upload. The value 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 and $key arguments and the part_size option are ignored.

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

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

use Aws\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;
```

Sample Code

```
$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";
}
```

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

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

use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;
```

Sample Code

```
// 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) {
        // $command is a CreateMultipartUpload operation
        $command['CacheControl'] = 'max-age=3600';
    },
    'before_upload' => function (Command $command) {
        // $command is an UploadPart operation
        $command['RequestPayer'] = 'requester';
    },
    'before_complete' => function (Command $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.

```
$uploader = new MultipartUploader($client, $source, [
   'bucket' => 'your-bucket',
   'key' => 'your-key',
   'before_upload' => function(\Aws\Command $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

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

use Aws\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;
```

Sample Code

```
// 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',
]);

//Recover from errors
do {
    try {
        $result = $uploader->upload();
    } catch (MultipartUploadException $e) {
        $uploader = new MultipartUploader($s3Client, $source, [
            'state' => $e->getState(),
        ]);
    }
} while (!isset($result));

//Abort a multipart upload if failed
try {
    $result = $uploader->upload();
} catch (MultipartUploadException $e) {
    // State contains the "Bucket", "Key", and "UploadId"
    $params = $e->getState()->getId();
    $result = $s3Client->abortMultipartUpload($params);
}
```

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

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

use Aws\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;
```

Sample Code

```
// 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',
]);

do {
    try {
        $result = $uploader->upload();
    } catch (MultipartUploadException $e) {
        rewind($source);
        $uploader = new MultipartUploader($s3Client, $source, [
            'state' => $e->getState(),
        ]);
    }
} while (!isset($result));
```

```
fclose($source);
```

Aborting a multipart upload

A multipart upload can be aborted by retrieving the `UploadId` contained in the `UploadState` object and passing it to `abortMultipartUpload`.

```
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] for the multipart upload.

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

use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;
```

Sample Code

```
// 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\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.

add_content_md5

(boolean) Set to true to automatically calculate the MD5 checksum for the upload.

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\Exception\MultipartUploadException;
use Aws\S3\MultipartCopy;
use Aws\S3\S3Client;
```

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 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

```
use Aws\Exception\AwsException;
use AwsUtilities\PrintableLineBreak;
use AwsUtilities\TestableReadline;
use DateTime;

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

Sample Code

```
        $command = $s3Service->getClient()->getCommand('GetObject', [
            'Bucket' => $bucket,
            'Key' => $key,
        ]);
```

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

```
        try {
            $preSignedUrl = $s3Service->preSignedUrl($command, $expiration);
            echo "Your preSignedUrl is \n$preSignedUrl\nand will be good for the next
 20 minutes.\n";
            echo $linebreak;
            echo "Thanks for trying the Amazon S3 presigned URL demo.\n";
        } catch (AwsException $exception) {
            echo $linebreak;
            echo "Something went wrong: $exception";
            die();
        }
```

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

```
$preSignedUrl = $s3Service->preSignedUrl($command, $expiration);
```

> **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.

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 AwsS3PostObjectV4.

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 is available here on GitHub.

Credentials

> **Note**
>
> `PostObjectV4` does not work with credentials sourced from AWS IAM Identity Center.

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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 User Guide)
- expiration time string for the policy (optional, one hour by default).

Imports

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

use Aws\S3\PostObjectV4;
use Aws\S3\S3Client;
```

Sample Code

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

use Aws\S3\PostObjectV4;
use Aws\S3\S3Client;

$client = new S3Client([
    'profile' => 'default',
    'region' => 'us-east-1',
]);
$bucket = 'doc-example-bucket10';
$starts_with = 'user/eric/';
$client->listBuckets();

// Set 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', $starts_with],
];

// Set an expiration time (optional).
```

```
$expires = '+2 hours';

$postObject = new PostObjectV4(
    $client,
    $bucket,
    $formInputs,
    $options,
    $expires
);

// Get attributes for the HTML form, for example, action, method, enctype.
$formAttributes = $postObject->getFormAttributes();

// Get attributes for the HTML form values.
$formInputs = $postObject->getFormInputs();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <title>PHP</title>
</head>
<body>
<form action="<?php echo $formAttributes['action'] ?>" method="<?php echo
 $formAttributes['method'] ?>"
      enctype="<?php echo $formAttributes['enctype'] ?>">
    <label id="key">
        <input hidden type="text" name="key" value="<?php echo $starts_with ?><?php
 echo $formInputs['key'] ?>"/>
    </label>
    <h3>$formInputs:</h3>
    acl: <label id="acl">
        <input readonly type="text" name="acl" value="<?php echo $formInputs['acl'] ?
>"/>
    </label><br/>
    X-Amz-Credential: <label id="credential">
        <input readonly type="text" name="X-Amz-Credential" value="<?php echo
 $formInputs['X-Amz-Credential'] ?>"/>
    </label><br/>
    X-Amz-Algorithm: <label id="algorithm">
        <input readonly type="text" name="X-Amz-Algorithm" value="<?php echo
 $formInputs['X-Amz-Algorithm'] ?>"/>
    </label><br/>
    X-Amz-Date: <label id="date">
```

```
        <input readonly type="text" name="X-Amz-Date" value="<?php echo $formInputs['X-
Amz-Date'] ?>"/>
    </label><br/><br/><br/>
    Policy: <label id="policy">
        <input readonly type="text" name="Policy" value="<?php echo
 $formInputs['Policy'] ?>"/>
    </label><br/>
    X-Amz-Signature: <label id="signature">
        <input readonly type="text" name="X-Amz-Signature" value="<?php echo
 $formInputs['X-Amz-Signature'] ?>"/>
    </label><br/><br/>
    <h3>Choose file:</h3>
    <input type="file" name="file"/> <br/><br/>
    <h3>Upload file:</h3>
    <input type="submit" name="submit" value="Upload to Amazon S3"/>
</form>
</body>
</html>
```

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

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

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

Sample Code

```
$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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Get, delete, and replace a policy on a bucket

Imports

```
require "vendor/autoload.php";

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

Sample Code

```
$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 $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";
}

// Replaces a policy on the bucket
```

```
try {
    $resp = $s3Client->putBucketPolicy([
        'Bucket' => $bucket,
        'Policy' => 'foo policy',
    ]);
    echo "Succeed in put a policy on 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Imports

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

use Aws\S3\S3Client;
```

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

```
$s3 = new S3Client([
    'version'     => 'latest',
    'region'      => 'us-west-2',
]);
$result = $s3->getObject([
    'Bucket' => 'arn:aws:s3:us-west-2:123456789012:accesspoint:endpoint-name',
    'Key' => 'MyKey'
]);
```

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

```
$s3 = new S3Client([
    'version'     => 'latest',
    'region'      => 'us-west-2',
]);
$result = $s3->putObject([
    'Bucket' => 'arn:aws:s3:us-west-2:123456789012:accesspoint:endpoint-name',
    '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

```
$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`

```
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`).

```
[default]
s3_use_arn_region = true
```

Use Amazon S3 Multi-Region Access Points with the AWS SDK for PHP Version 3

Amazon Simple Storage Service (S3) Multi-Region Access Points provide a global endpoint for routing Amazon S3 request traffic between AWS Regions.

You can create Multi-Region Access Points using the SDK for PHP, another AWS SDK, the S3 console, or AWS CLI,

> **Important**
>
> To use Multi-Region Access Points with the SDK for PHP, your PHP environment must have the AWS Common Runtime (AWS CRT) extension installed.

When you create a Multi-Region Access Point, Amazon S3 generates an Amazon Resource Name (ARN) that has the following format:

`arn:aws:s3::account-id:accesspoint/MultiRegionAccessPoint_alias`

You can use the generated ARN in place of a bucket name for `getObject()` and `putObject()` methods.

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

use Aws\S3\S3Client;

// Assign the Multi-Region Access Point to a variable and use it place of a bucket
 name.
$mrap = 'arn:aws:s3::123456789012:accesspoint/mfzwi23gnjvgw.mrap';
$key = 'my-key';

$s3Client = new S3Client([
    'region' => 'us-east-1'
]);

$s3Client->putObject([
    'Bucket' => $mrap,
    'Key' => $key,
    'Body' => 'Hello World!'
]);

$result = $s3Client->getObject([
    'Bucket' => $mrap,
    'Key' => $key
]);

echo $result['Body'] . "\n";

// Clean up.
$result = $s3Client->deleteObject([
    'Bucket' => $mrap,
    'Key' => $key
]);

$s3Client->waitUntil('ObjectNotExists', ['Bucket' => $mrap, 'Key' => $key]);

echo "Object deleted\n";
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

```
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$client = new SecretsManagerClient([
    'profile' => 'default',
    'version' => '2017-10-17',
    'region' => 'us-west-2'
]);
$secretName = 'MySecretName';
$secret = json_encode([
    "username" => getenv("SMDEMO_USERNAME"),
    "password" => getenv("SMDEMO_PASSWORD"),
]);
$description = '<<Description>>';
try {
    $result = $client->createSecret([
        'Description' => $description,
        'Name' => $secretName,
        'SecretString' => $secret,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Retrieve a secret from Secrets Manager

To retrieve the value of a secret stored in Secrets Manager, use the GetSecretValue operation.

In the following example, `secret` is a string that contains the stored value. If the value for `username` is <<USERNAME>> and the value for `password` is <<PASSWORD>>, the output of `secret` is:

```
{"username":"<<USERNAME>>","password":"<<PASSWORD>>"}
```

Use `json_decode($secret, true)` to access the array values.

Imports

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$client = new SecretsManagerClient([
    'profile' => 'default',
    'version' => '2017-10-17',
    'region' => 'us-east-1',
]);

$secretName = 'MySecretName';

try {
    $result = $client->getSecretValue([
        'SecretId' => $secretName,
    ]);
} catch (AwsException $e) {
    $error = $e->getAwsErrorCode();
    if ($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 $e;
    }
    if ($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']);
}
print $secret;
$secretArray = json_decode($secret, true);
$username = $secretArray['username'];
$password = $secretArray['password'];

// 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

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$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

```
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$client = new SecretsManagerClient([
    'profile' => 'default',
    'version' => '2017-10-17',
    'region' => 'us-west-2'
]);
$secretName = 'MySecretName';
$secret = json_encode([
    "username" => getenv("SMDEMO_USERNAME"),
    "password" => getenv("SMDEMO_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";
}
```

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

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

Sample Code

```
$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";
}
```

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

```
$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

```
$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";
}
```

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.

AWS offers two versions of Amazon SES service and, correspondingly, the SDK for PHP offers two versions of the client: SesClient and SesV2Client. The functionality of the clients overlap in many cases although the way the methods are called or the results may differ. The two APIs also offer exclusive features, so you can use both clients to access all the functionality.

The examples in this section all use the original, `SesClient`.

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
- Creating custom email templates using the Amazon SES API and the AWS SDK for PHP Version 3
- Managing email filters using the Amazon SES API and the AWS SDK for PHP Version 3
- Creating and managing email rules using the Amazon SES API and the AWS SDK for PHP Version 3
- Monitoring your sending activity using the Amazon SES API and the AWS SDK for PHP Version 3
- Authorizing senders using the Amazon SES API and the AWS SDK for PHP Version 3

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using Amazon SES, see the Amazon SES Developer Guide.

Verify an 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

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

```
use Aws\Exception\AwsException;
```

Sample Code

```
$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\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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$SesClient = new Aws\Ses\SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-2'
]);

try {
    $result = $SesClient->listIdentities([
        'IdentityType' => 'EmailAddress',
    ]);
    var_dump($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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

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->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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

```
use Aws\Exception\AwsException;
```

Sample Code

```
$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 send with Amazon SES using the AWS SDK for PHP.';

try {
    $result = $SesClient->createTemplate([
        '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";
}
```

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$SesClient = new Aws\Ses\SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-2'
]);

$name = 'Template_Name';

try {
    $result = $SesClient->getTemplate([
        'TemplateName' => $name,
    ]);
    var_dump($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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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 send 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\Exception\AwsException;
```

Sample Code

```
$SesClient = new Aws\Ses\SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-2'
]);
```

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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' => $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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using Amazon SES, see the Amazon SES Developer Guide.

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$SesClient = new Aws\Ses\SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-2'
]);

$filter_name = 'FilterName';

try {
    $result = $SesClient->deleteReceiptFilter([
        'FilterName' => $filter_name,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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,
                    ],
```

```
                ],
            ],
            'Name' => $rule_name,
            'ScanEnabled' => true,
            'TlsPolicy' => 'Optional',
            'Recipients' => ['<string>']
        ],
        'RuleSetName' =>  $rule_set_name,

     ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
```

Sample Code

```
$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' => $rule_set_name,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

Monitoring 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

For more information about using Amazon SES, see the Amazon SES Developer Guide.

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.

Imports

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$SesClient = new SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-1'

]);

try {
    $result = $SesClient->getSendQuota();
    $send_limit = $result["Max24HourSend"];
    $sent = $result["SentLast24Hours"];
    $available = $send_limit - $sent;
```

```
    print("<p>You can send " . $available . " more messages in the next 24 hours.</
p>");
    var_dump($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

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$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":[
        "SES:SendEmail",
        "SES:SendRawEmail"
      ]
    }
  ]
}
EOT;
$name = "policyName";

try {
    $result = $SesClient->putIdentityPolicy([
        'Identity' => $identity,
        'Policy' => $policy,
```

```
        'PolicyName' => $name,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

Sample Code

```
$SesClient = new SesClient([
    'profile' => 'default',
    'version' => '2010-12-01',
    'region' => 'us-east-1'
]);

$identity = "arn:aws:ses:us-east-1:123456789012:identity/example.com";
$name = "policyName";

try {
    $result = $SesClient->deleteIdentityPolicy([
        'Identity' => $identity,
        'PolicyName' => $name,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
```

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

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.

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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\Exception\AwsException;
use Aws\Sns\SnsClient;
```

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';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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.

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$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 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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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.

Imports

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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->confirmSubscription([
        '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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$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.

Imports

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a 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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Get SMS attributes

To retrieve the default settings for SMS messages, use the GetSMSAttributes operation.

This example gets the `DefaultSMSType` 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

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

```
use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->getSMSAttributes([
        'attributes' => ['DefaultSMSType'],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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\Exception\AwsException;
use Aws\Sns\SnsClient;
```

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\Exception\AwsException;
use Aws\Sns\SnsClient;
```

Sample Code

```
$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

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Set attributes on a queue to enable long polling

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueUrl = "QUEUE_URL";

$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->setQueueAttributes([
        '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

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueUrl = "QUEUE_URL";

$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->receiveMessage([
        '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

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueName = "QUEUE_NAME";
```

```
$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->createQueue([
        'QueueName' => $queueName,
        'Attributes' => [
            'ReceiveMessageWaitTimeSeconds' => 20
        ],
    ]);
    var_dump($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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Change the visibility timeout of multiple messages

Imports

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

```
use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$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++) {
            $entries[] = [
                'Id' => 'unique_is_msg' . $i, // REQUIRED
                'ReceiptHandle' => $messages[$i]['ReceiptHandle'], // REQUIRED
                'VisibilityTimeout' => 3600
            ];
        }
        $result = $client->changeMessageVisibilityBatch([
            'Entries' => $entries,
            'QueueUrl' => $queueUrl
        ]);

        var_dump($result);
    } else {
        echo "No messages in queue \n";
    }
} catch (AwsException $e) {
```

```
    // output error message if fails
    error_log($e->getMessage());
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Send a message

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
```

```
]);

$params = [
    'DelaySeconds' => 10,
    'MessageAttributes' => [
        "Title" => [
            'DataType' => "String",
            'StringValue' => "The Hitchhiker's Guide to the Galaxy"
        ],
        "Author" => [
            'DataType' => "String",
            'StringValue' => "Douglas Adams."
        ],
        "WeeksOn" => [
            'DataType' => "Number",
            'StringValue' => "6"
        ]
    ],
    'MessageBody' => "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
    'QueueUrl' => 'QUEUE_URL'
];

try {
    $result = $client->sendMessage($params);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Receive and delete messages

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueUrl = "QUEUE_URL";

$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->receiveMessage([
        '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";
    }
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Enable a dead-letter queue

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$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 is available here on GitHub.

Credentials

Before running the example code, configure your AWS credentials, as described in Credentials. Then import the AWS SDK for PHP, as described in Basic usage.

Return a list of queues

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$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\n";
```

```
    }
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Create a queue

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueName = "SQS_QUEUE_NAME";

$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->createQueue([
        'QueueName' => $queueName,
        'Attributes' => [
            'DelaySeconds' => 5,
            'MaximumMessageSize' => 4096, // 4 KB
        ],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Return the URL of a queue

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$queueName = "SQS_QUEUE_NAME";

$client = new SqsClient([
    'profile' => 'default',
    'region' => 'us-west-2',
    'version' => '2012-11-05'
]);

try {
    $result = $client->getQueueUrl([
        'QueueName' => $queueName // REQUIRED
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

Delete a queue

Imports

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

Sample Code

```
$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());
}
```

Send events to Amazon EventBridge global endpoints

You can use Amazon EventBridge global endpoints to improve the availability and reliability of your event-driven applications.

After the EventBridge global endpoint is set up, you can send events to it by using the SDK for PHP.

> **Important**
>
> To use EventBridge global endpoints with the SDK for PHP, your PHP environment must have the AWS Common Runtime (AWS CRT) extension installed.

The following example uses the PutEvents method of the `EventBridgeClient` to send a single event to an EventBridge global endpoint.

```
<?php
/* Send a single event to an existing Amazon EventBridge global endpoint. */
require '../vendor/autoload.php';
```

```
use Aws\EventBridge\EventBridgeClient;

$evClient = new EventBridgeClient([
    'region' => 'us-east-1'
]);

$endpointId = 'xxxx123456.xxx';  // Existing EventBridge global endpointId.
$eventBusName = 'default';       // Existing event bus in the us-east-1 Region.

$event =  [
    'Source' => 'my-php-app',
    'DetailType' => 'test',
    'Detail' => json_encode(['foo' => 'bar']),
    'Time' => new DateTime(),
    'Resources' => ['php-script'],
    'EventBusName' => $eventBusName,
    'TraceHeader' => 'test'
];

$result = $evClient->putEvents([
    'EndpointId' => $endpointId,
    'Entries' => [$event]
]);
```

This blog post contains more information about EventBridge global endpoints.

SDK for PHP code examples

The code examples in this topic show you how to use the AWS SDK for PHP with AWS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples are sample applications that work across multiple AWS services.

Examples

- Actions and scenarios using SDK for PHP
- Cross-service examples using SDK for PHP

Actions and scenarios using SDK for PHP

The following code examples show how to perform actions and implement common scenarios by using the AWS SDK for PHP with AWS services.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Services

- API Gateway examples using SDK for PHP
- Auto Scaling examples using SDK for PHP
- Amazon Bedrock examples using SDK for PHP
- Amazon Bedrock Runtime examples using SDK for PHP
- Amazon DocumentDB examples using SDK for PHP

- DynamoDB examples using SDK for PHP
- AWS Glue examples using SDK for PHP
- IAM examples using SDK for PHP
- Kinesis examples using SDK for PHP
- Lambda examples using SDK for PHP
- Amazon RDS examples using SDK for PHP
- Amazon S3 examples using SDK for PHP
- Amazon SNS examples using SDK for PHP
- Amazon SQS examples using SDK for PHP

API Gateway examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with API Gateway.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`GetBasePathMapping`

The following code example shows how to use `GetBasePathMapping`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\ApiGateway\ApiGatewayClient;
use Aws\Exception\AwsException;


/* ////////////////////////////////////////////////////////////////////////
 * Purpose: Gets the base path mapping for a custom domain name in
 * Amazon API Gateway.
 *
 * Prerequisites: A custom domain name in API Gateway. For more information,
 * see "Custom Domain Names" in the Amazon API Gateway Developer Guide.
 *
 * Inputs:
 * - $apiGatewayClient: An initialized AWS SDK for PHP API client for
 *   API Gateway.
 * - $basePath: The base path name that callers must provide as part of the
 *   URL after the domain name.
 * - $domainName: The custom domain name for the base path mapping.
 *
 * Returns: The base path mapping, if available; otherwise, the error message.
 * //////////////////////////////////////////////////////////////////////// */

function getBasePathMapping($apiGatewayClient, $basePath, $domainName)
{
    try {
        $result = $apiGatewayClient->getBasePathMapping([
            'basePath' => $basePath,
            'domainName' => $domainName,
        ]);
        return 'The base path mapping\'s effective URI is: ' .
            $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e['message'];
```

```
    }
}

function getsTheBasePathMapping()
{
    $apiGatewayClient = new ApiGatewayClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => '2015-07-09'
    ]);

    echo getBasePathMapping($apiGatewayClient, '(none)', 'example.com');
}

// Uncomment the following line to run this code in an AWS account.
// getsTheBasePathMapping();
```

- For API details, see GetBasePathMapping in *AWS SDK for PHP API Reference*.

`ListBasePathMappings`

The following code example shows how to use `ListBasePathMappings`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\ApiGateway\ApiGatewayClient;
use Aws\Exception\AwsException;


/* ////////////////////////////////////////////////////////////////////////
 * Purpose: Lists the base path mapping for a custom domain name in
 * Amazon API Gateway.
```

```
 *
 * Prerequisites: A custom domain name in API Gateway. For more information,
 * see "Custom Domain Names" in the Amazon API Gateway Developer Guide.
 *
 * Inputs:
 * - $apiGatewayClient: An initialized AWS SDK for PHP API client for
 *   API Gateway.
 * - $domainName: The custom domain name for the base path mappings.
 *
 * Returns: Information about the base path mappings, if available;
 * otherwise, the error message.
 * ////////////////////////////////////////////////////////////////////// */

function listBasePathMappings($apiGatewayClient, $domainName)
{
    try {
        $result = $apiGatewayClient->getBasePathMappings([
            'domainName' => $domainName
        ]);
        return 'The base path mapping(s) effective URI is: ' .
            $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e['message'];
    }
}

function listTheBasePathMappings()
{
    $apiGatewayClient = new ApiGatewayClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => '2015-07-09'
    ]);

    echo listBasePathMappings($apiGatewayClient, 'example.com');
}

// Uncomment the following line to run this code in an AWS account.
// listTheBasePathMappings();
```

- For API details, see ListBasePathMappings in *AWS SDK for PHP API Reference*.

`UpdateBasePathMapping`

The following code example shows how to use `UpdateBasePathMapping`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\ApiGateway\ApiGatewayClient;
use Aws\Exception\AwsException;


/* ////////////////////////////////////////////////////////////////////////
 *
 * Purpose: Updates the base path mapping for a custom domain name
 * in Amazon API Gateway.
 *
 * Inputs:
 * - $apiGatewayClient: An initialized AWS SDK for PHP API client for
 *   API Gateway.
 * - $basePath: The base path name that callers must provide as part of the
 *   URL after the domain name.
 * - $domainName: The custom domain name for the base path mapping.
 * - $patchOperations: The base path update operations to apply.
 *
 * Returns: Information about the updated base path mapping, if available;
 * otherwise, the error message.
 * //////////////////////////////////////////////////////////////////////// */

function updateBasePathMapping(
    $apiGatewayClient,
    $basePath,
    $domainName,
    $patchOperations
) {
    try {
        $result = $apiGatewayClient->updateBasePathMapping([
```

```
            'basePath' => $basePath,
            'domainName' => $domainName,
            'patchOperations' => $patchOperations
        ]);
        return 'The updated base path\'s URI is: ' .
            $result['@metadata']['effectiveUri'];
    } catch (AwsException $e) {
        return 'Error: ' . $e['message'];
    }
}

function updateTheBasePathMapping()
{
    $patchOperations = array([
        'op' => 'replace',
        'path' => '/stage',
        'value' => 'stage2'
    ]);

    $apiGatewayClient = new ApiGatewayClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => '2015-07-09'
    ]);

    echo updateBasePathMapping(
        $apiGatewayClient,
        '(none)',
        'example.com',
        $patchOperations
    );
}

// Uncomment the following line to run this code in an AWS account.
// updateTheBasePathMapping();
```

- For API details, see UpdateBasePathMapping in *AWS SDK for PHP API Reference*.

Auto Scaling examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Auto Scaling.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Get started

Hello Auto Scaling

The following code examples show how to get started using Auto Scaling.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function helloService()
{
    $autoScalingClient = new AutoScalingClient([
        'region' => 'us-west-2',
        'version' => 'latest',
        'profile' => 'default',
    ]);

    $groups = $autoScalingClient->describeAutoScalingGroups([]);
    var_dump($groups);
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for PHP API Reference*.

Topics

Actions

`CreateAutoScalingGroup`

The following code example shows how to use `CreateAutoScalingGroup`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function createAutoScalingGroup(
        $autoScalingGroupName,
        $availabilityZones,
        $minSize,
        $maxSize,
        $launchTemplateId
    ) {
        return $this->autoScalingClient->createAutoScalingGroup([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'AvailabilityZones' => $availabilityZones,
            'MinSize' => $minSize,
            'MaxSize' => $maxSize,
            'LaunchTemplate' => [
                'LaunchTemplateId' => $launchTemplateId,
            ],
        ]);
    }
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for PHP API Reference*.

`DeleteAutoScalingGroup`

The following code example shows how to use `DeleteAutoScalingGroup`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function deleteAutoScalingGroup($autoScalingGroupName)
{
    return $this->autoScalingClient->deleteAutoScalingGroup([
        'AutoScalingGroupName' => $autoScalingGroupName,
        'ForceDelete' => true,
    ]);
}
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for PHP API Reference*.

`DescribeAutoScalingGroups`

The following code example shows how to use `DescribeAutoScalingGroups`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function describeAutoScalingGroups($autoScalingGroupNames)
{
    return $this->autoScalingClient->describeAutoScalingGroups([
        'AutoScalingGroupNames' => $autoScalingGroupNames
```

```
        ]);
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for PHP API Reference*.

`DescribeAutoScalingInstances`

The following code example shows how to use `DescribeAutoScalingInstances`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function describeAutoScalingInstances($instanceIds)
    {
        return $this->autoScalingClient->describeAutoScalingInstances([
            'InstanceIds' => $instanceIds
        ]);
    }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for PHP API Reference*.

`DescribeScalingActivities`

The following code example shows how to use `DescribeScalingActivities`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function describeScalingActivities($autoScalingGroupName)
    {
        return $this->autoScalingClient->describeScalingActivities([
            'AutoScalingGroupName' => $autoScalingGroupName,
        ]);
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for PHP API Reference*.

`DisableMetricsCollection`

The following code example shows how to use `DisableMetricsCollection`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function disableMetricsCollection($autoScalingGroupName)
    {
        return $this->autoScalingClient->disableMetricsCollection([
            'AutoScalingGroupName' => $autoScalingGroupName,
        ]);
    }
```

- For API details, see DisableMetricsCollection in *AWS SDK for PHP API Reference*.

`EnableMetricsCollection`

The following code example shows how to use `EnableMetricsCollection`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function enableMetricsCollection($autoScalingGroupName, $granularity)
{
    return $this->autoScalingClient->enableMetricsCollection([
        'AutoScalingGroupName' => $autoScalingGroupName,
        'Granularity' => $granularity,
    ]);
}
```

- For API details, see EnableMetricsCollection in *AWS SDK for PHP API Reference*.

`SetDesiredCapacity`

The following code example shows how to use `SetDesiredCapacity`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function setDesiredCapacity($autoScalingGroupName, $desiredCapacity)
{
    return $this->autoScalingClient->setDesiredCapacity([
        'AutoScalingGroupName' => $autoScalingGroupName,
        'DesiredCapacity' => $desiredCapacity,
    ]);
}
```

- For API details, see SetDesiredCapacity in *AWS SDK for PHP API Reference*.

`TerminateInstanceInAutoScalingGroup`

The following code example shows how to use `TerminateInstanceInAutoScalingGroup`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function terminateInstanceInAutoScalingGroup(
        $instanceId,
        $shouldDecrementDesiredCapacity = true,
        $attempts = 0
    ) {
        try {
            return $this->autoScalingClient->terminateInstanceInAutoScalingGroup([
                'InstanceId' => $instanceId,
                'ShouldDecrementDesiredCapacity' => $shouldDecrementDesiredCapacity,
            ]);
        } catch (AutoScalingException $exception) {
            if ($exception->getAwsErrorCode() == "ScalingActivityInProgress" &&
$attempts < 5) {
                error_log("Cannot terminate an instance while it is still pending.
Waiting then trying again.");
                sleep(5 * (1 + $attempts));
                return $this->terminateInstanceInAutoScalingGroup(
                    $instanceId,
                    $shouldDecrementDesiredCapacity,
                    ++$attempts
                );
            } else {
                throw $exception;
            }
        }
    }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for PHP API Reference*.

`UpdateAutoScalingGroup`

The following code example shows how to use `UpdateAutoScalingGroup`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function updateAutoScalingGroup($autoScalingGroupName, $args)
{
    if (array_key_exists('MaxSize', $args)) {
        $maxSize = ['MaxSize' => $args['MaxSize']];
    } else {
        $maxSize = [];
    }
    if (array_key_exists('MinSize', $args)) {
        $minSize = ['MinSize' => $args['MinSize']];
    } else {
        $minSize = [];
    }
    $parameters = ['AutoScalingGroupName' => $autoScalingGroupName];
    $parameters = array_merge($parameters, $minSize, $maxSize);
    return $this->autoScalingClient->updateAutoScalingGroup($parameters);
}
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for PHP API Reference*.

Scenarios

Manage groups and instances

The following code example shows how to:

- Create an Amazon EC2 Auto Scaling group with a launch template and Availability Zones, and get information about running instances.
- Enable Amazon CloudWatch metrics collection.
- Update the group's desired capacity and wait for an instance to start.
- Terminate an instance in the group.
- List scaling activities that occur in response to user requests and capacity changes.
- Get statistics for CloudWatch metrics, then clean up resources.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AutoScaling;

use Aws\AutoScaling\AutoScalingClient;
use Aws\CloudWatch\CloudWatchClient;
use Aws\Ec2\Ec2Client;
use AwsUtilities\AWSServiceClass;
use AwsUtilities\RunnableExample;

class GettingStartedWithAutoScaling implements RunnableExample
{
    protected Ec2Client $ec2Client;
    protected AutoScalingClient $autoScalingClient;
    protected AutoScalingService $autoScalingService;
    protected CloudWatchClient $cloudWatchClient;
    protected string $templateName;
    protected string $autoScalingGroupName;
    protected array $role;

    public function runExample()
    {
        echo("\n");
        echo("-------------------------------------\n");
```

```
        print("Welcome to the Amazon EC2 Auto Scaling getting started demo using
 PHP!\n");
        echo("--------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $this->autoScalingClient = new AutoScalingClient($clientArgs);
        $this->autoScalingService = new AutoScalingService($this-
>autoScalingClient);
        $this->cloudWatchClient = new CloudWatchClient($clientArgs);

        AWSServiceClass::$waitTime = 5;
        AWSServiceClass::$maxWaitAttempts = 20;

        /**
         * Step 0: Create an EC2 launch template that you'll use to create an Auto
 Scaling group.
         */
        $this->ec2Client = new EC2Client($clientArgs);
        $this->templateName = "example_launch_template_$uniqid";
        $instanceType = "t1.micro";
        $amiId = "ami-0ca285d4c2cda3300";
        $launchTemplate = $this->ec2Client->createLaunchTemplate(
            [
            'LaunchTemplateName' => $this->templateName,
            'LaunchTemplateData' => [
                'InstanceType' => $instanceType,
                'ImageId' => $amiId,
            ]
            ]
        );

        /**
         * Step 1: CreateAutoScalingGroup: pass it the launch template you created
 in step 0.
         */
        $availabilityZones[] = $this->ec2Client->describeAvailabilityZones([])
['AvailabilityZones'][1]['ZoneName'];
```

```
        $this->autoScalingGroupName = "demoAutoScalingGroupName_$uniqid";
        $minSize = 1;
        $maxSize = 1;
        $launchTemplateId = $launchTemplate['LaunchTemplate']['LaunchTemplateId'];
        $this->autoScalingService->createAutoScalingGroup(
            $this->autoScalingGroupName,
            $availabilityZones,
            $minSize,
            $maxSize,
            $launchTemplateId
        );

        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        $autoScalingGroup = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);

        /**
         * Step 2: DescribeAutoScalingInstances: show that one instance has
 launched.
         */
        $instanceIds = [$autoScalingGroup['AutoScalingGroups'][0]['Instances'][0]
['InstanceId']];
        $instances = $this->autoScalingService-
>describeAutoScalingInstances($instanceIds);
        echo "The Auto Scaling group {$this->autoScalingGroupName} was created
 successfully.\n";
        echo count($instances['AutoScalingInstances']) . " instances were created
 for the group.\n";
        echo $autoScalingGroup['AutoScalingGroups'][0]['MaxSize'] . " is the max
 number of instances for the group.\n";

        /**
         * Step 3: EnableMetricsCollection: enable all metrics or a subset.
         */
        $this->autoScalingService->enableMetricsCollection($this-
>autoScalingGroupName, "1Minute");

        /**
         * Step 4: UpdateAutoScalingGroup: update max size to 3.
         */
        echo "Updating the max number of instances to 3.\n";
        $this->autoScalingService->updateAutoScalingGroup($this-
>autoScalingGroupName, ['MaxSize' => 3]);
```

```
        /**
         * Step 5: DescribeAutoScalingGroups: show the current state of the group.
         */
        $autoScalingGroup = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
        echo $autoScalingGroup['AutoScalingGroups'][0]['MaxSize'];
        echo " is the updated max number of instances for the group.\n";

        $limits = $this->autoScalingService->describeAccountLimits();
        echo "Here are your account limits:\n";
        echo "MaxNumberOfAutoScalingGroups:
 {$limits['MaxNumberOfAutoScalingGroups']}\n";
        echo "MaxNumberOfLaunchConfigurations:
 {$limits['MaxNumberOfLaunchConfigurations']}\n";
        echo "NumberOfAutoScalingGroups: {$limits['NumberOfAutoScalingGroups']}\n";
        echo "NumberOfLaunchConfigurations:
 {$limits['NumberOfLaunchConfigurations']}\n";

        /**
         * Step 6: SetDesiredCapacity: set desired capacity to 2.
         */
        $this->autoScalingService->setDesiredCapacity($this->autoScalingGroupName,
 2);
        sleep(10); // Wait for the group to start processing the request.
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);

        /**
         * Step 7: DescribeAutoScalingInstances: show that two instances are
 launched.
         */
        $autoScalingGroups = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
        foreach ($autoScalingGroups['AutoScalingGroups'] as $autoScalingGroup) {
            echo "There is a group named:
 {$autoScalingGroup['AutoScalingGroupName']}";
            echo "with an ARN of {$autoScalingGroup['AutoScalingGroupARN']}.\n";
            foreach ($autoScalingGroup['Instances'] as $instance) {
                echo "{$autoScalingGroup['AutoScalingGroupName']} has an instance
 with id of: ";
                echo "{$instance['InstanceId']} and a lifecycle state of:
 {$instance['LifecycleState']}.\n";
            }
```

```
        }

        /**
         * Step 8: TerminateInstanceInAutoScalingGroup: terminate one of the
 instances in the group.
         */
        $this->autoScalingService-
>terminateInstanceInAutoScalingGroup($instance['InstanceId'], false);
        do {
            sleep(10);
            $instances = $this->autoScalingService-
>describeAutoScalingInstances([$instance['InstanceId']]);
        } while (count($instances['AutoScalingInstances']) > 0);
        do {
            sleep(10);
            $autoScalingGroups = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
            $instances = $autoScalingGroups['AutoScalingGroups'][0]['Instances'];
        } while (count($instances) < 2);
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        foreach ($autoScalingGroups['AutoScalingGroups'] as $autoScalingGroup) {
            echo "There is a group named:
 {$autoScalingGroup['AutoScalingGroupName']}";
            echo "with an ARN of {$autoScalingGroup['AutoScalingGroupARN']}.\n";
            foreach ($autoScalingGroup['Instances'] as $instance) {
                echo "{$autoScalingGroup['AutoScalingGroupName']} has an instance
 with id of: ";
                echo "{$instance['InstanceId']} and a lifecycle state of:
 {$instance['LifecycleState']}.\n";
            }
        }

        /**
         * Step 9: DescribeScalingActivities: list the scaling activities that have
 occurred for the group so far.
         */
        $activities = $this->autoScalingService-
>describeScalingActivities($autoScalingGroup['AutoScalingGroupName']);
        echo "We found " . count($activities['Activities']) . " activities.\n";
        foreach ($activities['Activities'] as $activity) {
            echo "{$activity['ActivityId']} - {$activity['StartTime']} -
 {$activity['Description']}\n";
        }
```

```
        /**
         * Step 10: Use the Amazon CloudWatch API to get and show some metrics
 collected for the group.
         */
        $metricsNamespace = 'AWS/AutoScaling';
        $metricsDimensions = [
            [
                'Name' => 'AutoScalingGroupName',
                'Value' => $autoScalingGroup['AutoScalingGroupName'],
            ],
        ];
        $metrics = $this->cloudWatchClient->listMetrics(
            [
            'Dimensions' => $metricsDimensions,
            'Namespace' => $metricsNamespace,
            ]
        );
        foreach ($metrics['Metrics'] as $metric) {
            $timespan = 5;
            if ($metric['MetricName'] != 'GroupTotalCapacity' &&
 $metric['MetricName'] != 'GroupMaxSize') {
                continue;
            }
            echo "Over the last $timespan minutes, {$metric['MetricName']} recorded:
\n";
            $stats = $this->cloudWatchClient->getMetricStatistics(
                [
                'Dimensions' => $metricsDimensions,
                'EndTime' => time(),
                'StartTime' => time() - (5 * 60),
                'MetricName' => $metric['MetricName'],
                'Namespace' => $metricsNamespace,
                'Period' => 60,
                'Statistics' => ['Sum'],
                ]
            );
            foreach ($stats['Datapoints'] as $stat) {
                echo "{$stat['Timestamp']}: {$stat['Sum']}\n";
            }
        }

        return $instances;
    }
```

```
    public function cleanUp()
    {
        /**
         * Step 11: DisableMetricsCollection: disable all metrics.
         */
        $this->autoScalingService->disableMetricsCollection($this-
>autoScalingGroupName);

        /**
         * Step 12: DeleteAutoScalingGroup: to delete the group you must stop all
 instances.
         * - UpdateAutoScalingGroup with MinSize=0
         * - TerminateInstanceInAutoScalingGroup for each instance,
         *     specify ShouldDecrementDesiredCapacity=True. Wait for instances to
 stop.
         * - Now you can delete the group.
         */
        $this->autoScalingService->updateAutoScalingGroup($this-
>autoScalingGroupName, ['MinSize' => 0]);
        $this->autoScalingService->terminateAllInstancesInAutoScalingGroup($this-
>autoScalingGroupName);
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        $this->autoScalingService->deleteAutoScalingGroup($this-
>autoScalingGroupName);

        /**
         * Step 13: Delete launch template.
         */
        $this->ec2Client->deleteLaunchTemplate(
            [
            'LaunchTemplateName' => $this->templateName,
            ]
        );
    }

    public function helloService()
    {
        $autoScalingClient = new AutoScalingClient([
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ]);
```

```
        $groups = $autoScalingClient->describeAutoScalingGroups([]);
        var_dump($groups);
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Amazon Bedrock examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon Bedrock.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`ListFoundationModels`

The following code example shows how to use `ListFoundationModels`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the available Amazon Bedrock foundation models.

```
public function listFoundationModels()
{
    $result = $this->bedrockClient->listFoundationModels();
    return $result;
}
```

- For API details, see ListFoundationModels in *AWS SDK for PHP API Reference*.

Amazon Bedrock Runtime examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon Bedrock Runtime.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- AI21 Labs Jurassic-2
- Amazon Titan Image Generator
- Anthropic Claude
- Meta Llama
- Scenarios
- Stable Diffusion

AI21 Labs Jurassic-2

InvokeModel

The following code example shows how to send a text message to AI21 Labs Jurassic-2, using the Invoke Model API.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use the Invoke Model API to send a text message.

```
    public function invokeJurassic2($prompt)
    {
        # The different model providers have individual request and response
 formats.
        # For the format, ranges, and default values for AI21 Labs Jurassic-2, refer
 to:
        # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html

        $completion = "";

        try {
            $modelId = 'ai21.j2-mid-v1';

            $body = [
                'prompt' => $prompt,
```

```
                'temperature' => 0.5,
                'maxTokens' => 200,
            ];

            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => json_encode($body),
                'modelId' => $modelId,
            ]);

            $response_body = json_decode($result['body']);

            $completion = $response_body->completions[0]->data->text;
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $completion;
    }
```

- For API details, see InvokeModel in *AWS SDK for PHP API Reference*.

Amazon Titan Image Generator

InvokeModel

The following code example shows how to invoke Amazon Titan Image on Amazon Bedrock to generate an image.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an image with the Amazon Titan Image Generator.

```
    public function invokeTitanImage(string $prompt, int $seed)
    {
```

```
        # The different model providers have individual request and response
 formats.
        # For the format, ranges, and default values for Titan Image models refer
 to:
        # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
titan-image.html

        $base64_image_data = "";

        try {
            $modelId = 'amazon.titan-image-generator-v1';

            $request = json_encode([
                'taskType' => 'TEXT_IMAGE',
                'textToImageParams' => [
                    'text' => $prompt
                ],
                'imageGenerationConfig' => [
                    'numberOfImages' => 1,
                    'quality' => 'standard',
                    'cfgScale' => 8.0,
                    'height' => 512,
                    'width' => 512,
                    'seed' => $seed
                ]
            ]);

            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => $request,
                'modelId' => $modelId,
            ]);

            $response_body = json_decode($result['body']);

            $base64_image_data = $response_body->images[0];
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $base64_image_data;
    }
```

- For API details, see InvokeModel in *AWS SDK for PHP API Reference*.

Anthropic Claude

InvokeModel

The following code example shows how to send a text message to Anthropic Claude, using the Invoke Model API.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Invoke the Anthropic Claude 2 foundation model to generate text.

```
    public function invokeClaude($prompt)
    {
        # The different model providers have individual request and response
 formats.
        # For the format, ranges, and default values for Anthropic Claude, refer to:
        # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
claude.html

        $completion = "";

        try {
            $modelId = 'anthropic.claude-v2';

            # Claude requires you to enclose the prompt as follows:
            $prompt = "\n\nHuman: {$prompt}\n\nAssistant:";

            $body = [
                'prompt' => $prompt,
                'max_tokens_to_sample' => 200,
                'temperature' => 0.5,
                'stop_sequences' => ["\n\nHuman:"],
            ];
```

```
            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => json_encode($body),
                'modelId' => $modelId,
            ]);

            $response_body = json_decode($result['body']);

            $completion = $response_body->completion;
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $completion;
    }
```

- For API details, see InvokeModel in *AWS SDK for PHP API Reference*.

Meta Llama

InvokeModel: Llama 2

The following code example shows how to send a text message to Meta Llama 2, using the Invoke Model API.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use the Invoke Model API to send a text message.

```
    public function invokeLlama2($prompt)
    {
        # The different model providers have individual request and response
formats.
```

```
        # For the format, ranges, and default values for Meta Llama 2 Chat, refer
 to:
        # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
meta.html

        $completion = "";

        try {
            $modelId = 'meta.llama2-13b-chat-v1';

            $body = [
                'prompt' => $prompt,
                'temperature' => 0.5,
                'max_gen_len' => 512,
            ];

            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => json_encode($body),
                'modelId' => $modelId,
            ]);

            $response_body = json_decode($result['body']);

            $completion = $response_body->generation;
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $completion;
    }
```

- For API details, see InvokeModel in *AWS SDK for PHP API Reference*.

Scenarios

Invoke multiple foundation models on Amazon Bedrock

The following code example shows how to prepare and send a prompt to a variety of large-language models (LLMs) on Amazon Bedrock

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Invoke multiple LLMs on Amazon Bedrock.

```
namespace BedrockRuntime;

class GettingStartedWithBedrockRuntime
{
    protected BedrockRuntimeService $bedrockRuntimeService;

    public function runExample()
    {
        echo "\n";
        echo "---------------------------------------------------------------------
\n";
        echo "Welcome to the Amazon Bedrock Runtime getting started demo using PHP!
\n";
        echo "---------------------------------------------------------------------
\n";

        $clientArgs = [
            'region' => 'us-east-1',
            'version' => 'latest',
            'profile' => 'default',
        ];

        $bedrockRuntimeService = new BedrockRuntimeService($clientArgs);

        $prompt = 'In one paragraph, who are you?';

        echo "\nPrompt: " . $prompt;

        echo "\n\nAnthropic Claude:";
        echo $bedrockRuntimeService->invokeClaude($prompt);

        echo "\n\nAI21 Labs Jurassic-2: ";
        echo $bedrockRuntimeService->invokeJurassic2($prompt);
```

```
        echo "\n\nMeta Llama 2 Chat: ";
        echo $bedrockRuntimeService->invokeLlama2($prompt);

        echo
"\n----------------------------------------------------------------------\n";

        $image_prompt = 'stylized picture of a cute old steampunk robot';

        echo "\nImage prompt: " . $image_prompt;

        echo "\n\nStability.ai Stable Diffusion XL:\n";
        $diffusionSeed = rand(0, 4294967295);
        $style_preset = 'photographic';
        $base64 = $bedrockRuntimeService->invokeStableDiffusion($image_prompt,
$diffusionSeed, $style_preset);
        $image_path = $this->saveImage($base64, 'stability.stable-diffusion-xl');
        echo "The generated images have been saved to $image_path";

        echo "\n\nAmazon Titan Image Generation:\n";
        $titanSeed = rand(0, 2147483647);
        $base64 = $bedrockRuntimeService->invokeTitanImage($image_prompt,
$titanSeed);
        $image_path = $this->saveImage($base64, 'amazon.titan-image-generator-v1');
        echo "The generated images have been saved to $image_path";
    }

    private function saveImage($base64_image_data, $model_id): string
    {
        $output_dir = "output";

        if (!file_exists($output_dir)) {
            mkdir($output_dir);
        }

        $i = 1;
        while (file_exists("$output_dir/$model_id" . '_' . "$i.png")) {
            $i++;
        }

        $image_data = base64_decode($base64_image_data);

        $file_path = "$output_dir/$model_id" . '_' . "$i.png";
```

```
        $file = fopen($file_path, 'wb');
        fwrite($file, $image_data);
        fclose($file);

        return $file_path;
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - InvokeModel
  - InvokeModelWithResponseStream

Stable Diffusion

InvokeModel

The following code example shows how to invoke Stability.ai Stable Diffusion XL on Amazon Bedrock to generate an image.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an image with Stable Diffusion.

```
    public function invokeStableDiffusion(string $prompt, int $seed, string
 $style_preset)
    {
        # The different model providers have individual request and response
 formats.
        # For the format, ranges, and available style_presets of Stable Diffusion
 models refer to:
        # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
stability-diffusion.html
```

```
        $base64_image_data = "";

        try {
            $modelId = 'stability.stable-diffusion-xl';

            $body = [
                'text_prompts' => [
                    ['text' => $prompt]
                ],
                'seed' => $seed,
                'cfg_scale' => 10,
                'steps' => 30
            ];

            if ($style_preset) {
                $body['style_preset'] = $style_preset;
            }

            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => json_encode($body),
                'modelId' => $modelId,
            ]);

            $response_body = json_decode($result['body']);

            $base64_image_data = $response_body->artifacts[0]->base64;
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $base64_image_data;
    }
```

- For API details, see InvokeModel in *AWS SDK for PHP API Reference*.

Amazon DocumentDB examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon DocumentDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

Serverless examples

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Amazon DocumentDB event with Lambda using PHP.

```
<?php

require __DIR__.'/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
```

```
{
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
 "\n";
    }
}
return new DocumentDBEventHandler();
```

DynamoDB examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with DynamoDB.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

Actions

`BatchExecuteStatement`

The following code example shows how to use `BatchExecuteStatement`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array $parameters)
```

```
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- For API details, see BatchExecuteStatement in *AWS SDK for PHP API Reference*.

`BatchWriteItem`

The following code example shows how to use `BatchWriteItem`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
items or increase depth.");
        }

        $marshal = new Marshaler();
        $total = 0;
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' => ['Item'
=> $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.\n";
                break;
            }
        }
    }
```

- For API details, see BatchWriteItem in *AWS SDK for PHP API Reference*.

`CreateTable`

The following code example shows how to use `CreateTable`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a table.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName, 'AttributeType'
=> $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
```

```
        ]);
    }
```

- For API details, see CreateTable in *AWS SDK for PHP API Reference*.

`DeleteItem`

The following code example shows how to use `DeleteItem`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- For API details, see DeleteItem in *AWS SDK for PHP API Reference*.

`DeleteTable`

The following code example shows how to use `DeleteTable`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function deleteTable(string $TableName)
{
    $this->customWaiter(function () use ($TableName) {
        return $this->dynamoDbClient->deleteTable([
            'TableName' => $TableName,
        ]);
    });
}
```

- For API details, see DeleteTable in *AWS SDK for PHP API Reference*.

`ExecuteStatement`

The following code example shows how to use `ExecuteStatement`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function insertItemByPartiQL(string $statement, array $parameters)
```

```
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this->buildStatementAndParameters("SELECT",
$tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- For API details, see ExecuteStatement in *AWS SDK for PHP API Reference*.

`GetItem`

The following code example shows how to use `GetItem`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- For API details, see GetItem in *AWS SDK for PHP API Reference*.

`ListTables`

The following code example shows how to use `ListTables`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
```

```
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- For API details, see ListTables in *AWS SDK for PHP API Reference*.

`PutItem`

The following code example shows how to use `PutItem`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    echo "What's the name of the last movie you watched?\n";
    while (empty($movieName)) {
        $movieName = testable_readline("Movie name: ");
    }
    echo "And what year was it released?\n";
    $movieYear = "year";
    while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
        $movieYear = testable_readline("Year released: ");
    }

    $service->putItem([
        'Item' => [
            'year' => [
                'N' => "$movieYear",
            ],
            'title' => [
                'S' => $movieName,
            ],
        ],
        'TableName' => $tableName,
    ]);
```

```
    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- For API details, see PutItem in *AWS SDK for PHP API Reference*.

`Query`

The following code example shows how to use `Query`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
```

```
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- For API details, see Query in *AWS SDK for PHP API Reference*.

Scan

The following code example shows how to use `Scan`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
```

```
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- For API details, see Scan in *AWS SDK for PHP API Reference*.

`UpdateItem`

The following code example shows how to use `UpdateItem`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
$rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
```

```
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
$rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- For API details, see UpdateItem in *AWS SDK for PHP API Reference*.

Scenarios

Get started with tables, items, and queries

The following code example shows how to:

- Create a table that can hold movie data.
- Put, get, and update a single movie in the table.
- Write movie data to the table from a sample JSON file.
- Query for movies that were released in a given year.
- Scan for movies that were released in a range of years.
- Delete a movie from the table, then delete the table.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
```

```
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
$rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
```

```
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
 $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a {$movie['Item']
['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something lighter,
 in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
```

```
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were born:
\n";
        $oops = "Oops! There were no movies released in that year (that we know of).
\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Query a table by using batches of PartiQL statements

The following code example shows how to:

- Get a batch of items by running multiple SELECT statements.
- Add a batch of items by running multiple INSERT statements.
- Update a batch of items by running multiple UPDATE statements.
- Delete a batch of items by running multiple DELETE statements.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
```

```
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo using
 PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
```

```
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
 $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
 $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
```

```
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something lighter,
 in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were born:
\n";
        $oops = "Oops! There were no movies released in that year (that we know of).
\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
```

```
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
```

```
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- For API details, see BatchExecuteStatement in *AWS SDK for PHP API Reference*.

Query a table using PartiQL

The following code example shows how to:

- Get an item by running a SELECT statement.
- Add an item by running an INSERT statement.
- Update an item by running an UPDATE statement.
- Delete an item by running a DELETE statement.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo using
 PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
```

```
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
 $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";
```

```
        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1 ||
 $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something lighter,
 in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
```

```
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were born:
\n";
        $oops = "Oops! There were no movies released in that year (that we know of).
\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
```

```
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this->buildStatementAndParameters("SELECT",
$tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- For API details, see ExecuteStatement in *AWS SDK for PHP API Reference*.

Serverless examples

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a DynamoDB event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
```

```
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be marked
 as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting DynamoDB batch item failures with Lambda using PHP.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
```

```
        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

AWS Glue examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with AWS Glue.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios

Actions

`CreateCrawler`

The following code example shows how to use `CreateCrawler`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);

    public function createCrawler($crawlerName, $role, $databaseName, $path): Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
$databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }
```

- For API details, see CreateCrawler in *AWS SDK for PHP API Reference*.

`CreateJob`

The following code example shows how to use `CreateJob`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $jobName = 'test-job-' . $uniqid;

        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

    public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
    {
        return $this->glueClient->createJob([
            'Name' => $jobName,
            'Role' => $role,
            'Command' => [
                'Name' => 'glueetl',
                'ScriptLocation' => $scriptLocation,
                'PythonVersion' => $pythonVersion,
            ],
            'GlueVersion' => $glueVersion,
        ]);
    }
```

- For API details, see CreateJob in *AWS SDK for PHP API Reference*.

`DeleteCrawler`

The following code example shows how to use `DeleteCrawler`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- For API details, see DeleteCrawler in *AWS SDK for PHP API Reference*.

`DeleteDatabase`

The following code example shows how to use `DeleteDatabase`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
```

```
    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }
```

- For API details, see DeleteDatabase in *AWS SDK for PHP API Reference*.

`DeleteJob`

The following code example shows how to use `DeleteJob`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }
```

- For API details, see DeleteJob in *AWS SDK for PHP API Reference*.

`DeleteTable`

The following code example shows how to use `DeleteTable`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

    public function deleteTable($tableName, $databaseName)
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }
```

- For API details, see DeleteTable in *AWS SDK for PHP API Reference*.

`GetCrawler`

The following code example shows how to use `GetCrawler`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
```

```
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }
```

- For API details, see GetCrawler in *AWS SDK for PHP API Reference*.

`GetDatabase`

The following code example shows how to use `GetDatabase`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $databaseName = "doc-example-database-$uniqid";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }
```

- For API details, see GetDatabase in *AWS SDK for PHP API Reference*.

`GetJobRun`

The following code example shows how to use `GetJobRun`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $jobName = 'test-job-' . $uniqid;

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']], ['SUCCEEDED',
'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }
```

- For API details, see GetJobRun in *AWS SDK for PHP API Reference*.

`GetJobRuns`

The following code example shows how to use `GetJobRuns`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $jobName = 'test-job-' . $uniqid;

        $jobRuns = $glueService->getJobRuns($jobName);

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''): Result
    {
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }
```

- For API details, see GetJobRuns in *AWS SDK for PHP API Reference*.

`GetTables`

The following code example shows how to use `GetTables`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    $databaseName = "doc-example-database-$uniqid";

    $tables = $glueService->getTables($databaseName);

public function getTables($databaseName): Result
{
    return $this->glueClient->getTables([
        'DatabaseName' => $databaseName,
    ]);
}
```

- For API details, see GetTables in *AWS SDK for PHP API Reference*.

`ListJobs`

The following code example shows how to use `ListJobs`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    $jobs = $glueService->listJobs();
    echo "Current jobs:\n";
    foreach ($jobs['JobNames'] as $jobsName) {
        echo "{$jobsName}\n";
    }
```

```
    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
    {
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }
```

- For API details, see ListJobs in *AWS SDK for PHP API Reference*.

`StartCrawler`

The following code example shows how to use `StartCrawler`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $glueService->startCrawler($crawlerName);

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
```

```
    }
```

- For API details, see StartCrawler in *AWS SDK for PHP API Reference*.

`StartJobRun`

The following code example shows how to use `StartJobRun`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        $jobName = 'test-job-' . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

   public function startJobRun($jobName, $databaseName, $tables, $outputBucketUrl):
Result
   {
       return $this->glueClient->startJobRun([
           'JobName' => $jobName,
           'Arguments' => [
               'input_database' => $databaseName,
               'input_table' => $tables['TableList'][0]['Name'],
               'output_bucket_url' => $outputBucketUrl,
               '--input_database' => $databaseName,
               '--input_table' => $tables['TableList'][0]['Name'],
               '--output_bucket_url' => $outputBucketUrl,
           ],
       ]);
   }
```

- For API details, see StartJobRun in *AWS SDK for PHP API Reference*.

Scenarios

Get started with crawlers and jobs

The following code example shows how to:

- Create a crawler that crawls a public Amazon S3 bucket and generates a database of CSV-formatted metadata.
- List information about databases and tables in your AWS Glue Data Catalog.
- Create a job to extract CSV data from the S3 bucket, transform the data, and load JSON-formatted output into another S3 bucket.
- List information about job runs, view transformed data, and clean up resources.

For more information, see Tutorial: Getting started with AWS Glue Studio.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace Glue;

use Aws\Glue\GlueClient;
use Aws\S3\S3Client;
use AwsUtilities\AWSServiceClass;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithGlue
{
    public function run()
    {
```

```
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the AWS Glue getting started demo using PHP!\n");
        echo("--------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $glueClient = new GlueClient($clientArgs);
        $glueService = new GlueService($glueClient);
        $iamService = new IAMService();
        $crawlerName = "example-crawler-test-" . $uniqid;

        AWSServiceClass::$waitTime = 5;
        AWSServiceClass::$maxWaitAttempts = 20;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $databaseName = "doc-example-database-$uniqid";
        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);
        $glueService->startCrawler($crawlerName);

        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

        //Upload job script
        $s3client = new S3Client($clientArgs);
        $bucketName = "test-glue-bucket-" . $uniqid;
        $s3client->createBucket([
            'Bucket' => $bucketName,
```

```
            'CreateBucketConfiguration' => ['LocationConstraint' => 'us-west-2'],
        ]);

        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => 'run_job.py',
            'SourceFile' => __DIR__ . '/flight_etl_job_script.py'
        ]);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => 'setup_scenario_getting_started.yaml',
            'SourceFile' => __DIR__ . '/setup_scenario_getting_started.yaml'
        ]);

        $tables = $glueService->getTables($databaseName);

        $jobName = 'test-job-' . $uniqid;
        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']], ['SUCCEEDED',
'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

        $jobRuns = $glueService->getJobRuns($jobName);

        $objects = $s3client->listObjects([
            'Bucket' => $bucketName,
        ])['Contents'];

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

```
        echo "Downloading " . $objects[1]['Key'] . "\n";
        /** @var Stream $downloadObject */
        $downloadObject = $s3client->getObject([
            'Bucket' => $bucketName,
            'Key' => $objects[1]['Key'],
        ])['Body']->getContents();
        echo "Here is the first 1000 characters in the object.";
        echo substr($downloadObject, 0, 1000);

        $jobs = $glueService->listJobs();
        echo "Current jobs:\n";
        foreach ($jobs['JobNames'] as $jobsName) {
            echo "{$jobsName}\n";
        }

        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        echo "Delete all objects in the bucket.\n";
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
```

```
        ]);
        echo "Delete the bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);

        echo "This job was brought to you by the number $uniqid\n";
    }
}

namespace Glue;

use Aws\Glue\GlueClient;
use Aws\Result;

use function PHPUnit\Framework\isEmpty;

class GlueService extends \AwsUtilities\AWSServiceClass
{
    protected GlueClient $glueClient;

    public function __construct($glueClient)
    {
        $this->glueClient = $glueClient;
    }

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }

    public function createCrawler($crawlerName, $role, $databaseName, $path): Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
 $databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
```

```
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }

    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
            'DatabaseName' => $databaseName,
        ]);
    }

    public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
    {
        return $this->glueClient->createJob([
            'Name' => $jobName,
            'Role' => $role,
            'Command' => [
                'Name' => 'glueetl',
                'ScriptLocation' => $scriptLocation,
                'PythonVersion' => $pythonVersion,
            ],
            'GlueVersion' => $glueVersion,
        ]);
    }
```

```
    public function startJobRun($jobName, $databaseName, $tables, $outputBucketUrl):
Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
        ]);
    }

    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
    {
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''): Result
    {
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }
```

```
    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
 Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }

    public function deleteTable($tableName, $databaseName)
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }

    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.

- CreateCrawler
- CreateJob
- DeleteCrawler
- DeleteDatabase
- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

IAM examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with IAM.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

- Scenarios

Actions

`AttachRolePolicy`

The following code example shows how to use `AttachRolePolicy`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";
```

```
$service->attachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);

    public function attachRolePolicy($roleName, $policyArn)
    {
        return $this->customWaiter(function () use ($roleName, $policyArn) {
            $this->iamClient->attachRolePolicy([
                'PolicyArn' => $policyArn,
                'RoleName' => $roleName,
            ]);
        });
    }
```

- For API details, see AttachRolePolicy in *AWS SDK for PHP API Reference*.

`CreatePolicy`

The following code example shows how to use `CreatePolicy`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";
```

```
    public function createPolicy(string $policyName, string $policyDocument)
    {
        $result = $this->customWaiter(function () use ($policyName, $policyDocument)
{
            return $this->iamClient->createPolicy([
                'PolicyName' => $policyName,
                'PolicyDocument' => $policyDocument,
            ]);
        });
        return $result['Policy'];
    }
```

- For API details, see CreatePolicy in *AWS SDK for PHP API Reference*.

`CreateRole`

The following code example shows how to use `CreateRole`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";
```

```
    /**
     * @param string $roleName
     * @param string $rolePolicyDocument
     * @return array
     * @throws AwsException
     */
    public function createRole(string $roleName, string $rolePolicyDocument)
    {
        $result = $this->customWaiter(function () use ($roleName,
$rolePolicyDocument) {
            return $this->iamClient->createRole([
                'AssumeRolePolicyDocument' => $rolePolicyDocument,
                'RoleName' => $roleName,
            ]);
        });
        return $result['Role'];
    }
```

- For API details, see CreateRole in *AWS SDK for PHP API Reference*.

`CreateServiceLinkedRole`

The following code example shows how to use `CreateServiceLinkedRole`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function createServiceLinkedRole($awsServiceName, $customSuffix = "",
 $description = "")
    {
        $createServiceLinkedRoleArguments = ['AWSServiceName' => $awsServiceName];
        if ($customSuffix) {
```

```
            $createServiceLinkedRoleArguments['CustomSuffix'] = $customSuffix;
        }
        if ($description) {
            $createServiceLinkedRoleArguments['Description'] = $description;
        }
        return $this->iamClient-
>createServiceLinkedRole($createServiceLinkedRoleArguments);
    }
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for PHP API Reference*.

`CreateUser`

The following code example shows how to use `CreateUser`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";


    /**
     * @param string $name
     * @return array
     * @throws AwsException
     */
    public function createUser(string $name): array
    {
        $result = $this->iamClient->createUser([
            'UserName' => $name,
        ]);
```

```
        return $result['User'];
    }
```

- For API details, see CreateUser in *AWS SDK for PHP API Reference*.

`GetAccountPasswordPolicy`

The following code example shows how to use `GetAccountPasswordPolicy`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getAccountPasswordPolicy()
    {
        return $this->iamClient->getAccountPasswordPolicy();
    }
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for PHP API Reference*.

`GetPolicy`

The following code example shows how to use `GetPolicy`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getPolicy($policyArn)
    {
        return $this->customWaiter(function () use ($policyArn) {
            return $this->iamClient->getPolicy(['PolicyArn' => $policyArn]);
        });
    }
```

- For API details, see GetPolicy in *AWS SDK for PHP API Reference*.

`GetRole`

The following code example shows how to use `GetRole`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getRole($roleName)
    {
        return $this->customWaiter(function () use ($roleName) {
            return $this->iamClient->getRole(['RoleName' => $roleName]);
        });
    }
```

- For API details, see GetRole in *AWS SDK for PHP API Reference*.

`ListAttachedRolePolicies`

The following code example shows how to use `ListAttachedRolePolicies`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listAttachedRolePolicies($roleName, $pathPrefix = "", $marker =
 "", $maxItems = 0)
    {
        $listAttachRolePoliciesArguments = ['RoleName' => $roleName];
        if ($pathPrefix) {
            $listAttachRolePoliciesArguments['PathPrefix'] = $pathPrefix;
        }
        if ($marker) {
            $listAttachRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listAttachRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->iamClient-
>listAttachedRolePolicies($listAttachRolePoliciesArguments);
    }
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for PHP API Reference*.

`ListGroups`

The following code example shows how to use `ListGroups`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listGroups($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listGroupsArguments = [];
        if ($pathPrefix) {
            $listGroupsArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listGroupsArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listGroupsArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listGroups($listGroupsArguments);
    }
```

- For API details, see ListGroups in *AWS SDK for PHP API Reference*.

`ListPolicies`

The following code example shows how to use `ListPolicies`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listPolicies($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listPoliciesArguments = [];
        if ($pathPrefix) {
            $listPoliciesArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listPoliciesArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listPoliciesArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listPolicies($listPoliciesArguments);
    }
```

- For API details, see ListPolicies in *AWS SDK for PHP API Reference*.

`ListRolePolicies`

The following code example shows how to use `ListRolePolicies`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listRolePolicies($roleName, $marker = "", $maxItems = 0)
    {
        $listRolePoliciesArguments = ['RoleName' => $roleName];
        if ($marker) {
```

```
            $listRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->customWaiter(function () use ($listRolePoliciesArguments) {
            return $this->iamClient->listRolePolicies($listRolePoliciesArguments);
        });
    }
```

- For API details, see ListRolePolicies in *AWS SDK for PHP API Reference*.

`ListRoles`

The following code example shows how to use `ListRoles`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    /**
     * @param string $pathPrefix
     * @param string $marker
     * @param int $maxItems
     * @return Result
     * $roles = $service->listRoles();
     */
    public function listRoles($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listRolesArguments = [];
        if ($pathPrefix) {
            $listRolesArguments["PathPrefix"] = $pathPrefix;
        }
```

```
        if ($marker) {
            $listRolesArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listRolesArguments["MaxItems"] = $maxItems;
        }
        return $this->iamClient->listRoles($listRolesArguments);
    }
```

- For API details, see ListRoles in *AWS SDK for PHP API Reference*.

`ListSAMLProviders`

The following code example shows how to use `ListSAMLProviders`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listSAMLProviders()
    {
        return $this->iamClient->listSAMLProviders();
    }
```

- For API details, see ListSAMLProviders in *AWS SDK for PHP API Reference*.

`ListUsers`

The following code example shows how to use `ListUsers`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listUsers($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listUsersArguments = [];
        if ($pathPrefix) {
            $listUsersArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listUsersArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listUsersArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listUsers($listUsersArguments);
    }
```

- For API details, see ListUsers in *AWS SDK for PHP API Reference*.

Scenarios

Create a user and assume a role

The following code example shows how to create a user and assume a role.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace Iam\Basics;

require 'vendor/autoload.php';

use Aws\Credentials\Credentials;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;
use Iam\IAMService;

echo("\n");
echo("---------------------------------------\n");
print("Welcome to the IAM getting started demo using PHP!\n");
echo("---------------------------------------\n");

$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
```

```
echo "Created user with the arn: {$user['Arn']}\n";

$key = $service->createAccessKey($user['UserName']);
$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);

$inlinePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"sts:AssumeRole\",
                    \"Resource\": \"{$assumeRoleRole['Arn']}\"}]
}";
$inlinePolicy = $service->createUserPolicy("iam_demo_inline_policy_$uuid",
 $inlinePolicyDocument, $user['UserName']);
//First, fail to list the buckets with the user
$credentials = new Credentials($key['AccessKeyId'], $key['SecretAccessKey']);
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
try {
    $s3Client->listBuckets([
```

```
    ]);
    echo "this should not run";
} catch (S3Exception $exception) {
    echo "successfully failed!\n";
}

$stsClient = new StsClient(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
sleep(10);
$assumedRole = $stsClient->assumeRole([
    'RoleArn' => $assumeRoleRole['Arn'],
    'RoleSessionName' => "DemoAssumeRoleSession_$uuid",
]);
$assumedCredentials = [
    'key' => $assumedRole['Credentials']['AccessKeyId'],
    'secret' => $assumedRole['Credentials']['SecretAccessKey'],
    'token' => $assumedRole['Credentials']['SessionToken'],
];
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $assumedCredentials]);
try {
    $s3Client->listBuckets([]);
    echo "this should now run!\n";
} catch (S3Exception $exception) {
    echo "this should now not fail\n";
}

$service->detachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);
$deletePolicy = $service->deletePolicy($listAllBucketsPolicy['Arn']);
echo "Delete policy: {$listAllBucketsPolicy['PolicyName']}\n";
$deletedRole = $service->deleteRole($assumeRoleRole['Arn']);
echo "Deleted role: {$assumeRoleRole['RoleName']}\n";
$deletedKey = $service->deleteAccessKey($key['AccessKeyId'], $user['UserName']);
$deletedUser = $service->deleteUser($user['UserName']);
echo "Delete user: {$user['UserName']}\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - AttachRolePolicy
  - CreateAccessKey

- CreatePolicy
- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Kinesis examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Kinesis.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Serverless examples

Serverless examples

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an Kinesis event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data
```

```
            // Any exception thrown will be logged and the invocation will be marked
 as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting Kinesis batch item failures with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';
```

```
class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
```

```
}

$logger = new StderrLogger();
return new Handler($logger);
```

Lambda examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Lambda.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios
- Serverless examples

Actions

`CreateFunction`

The following code example shows how to use `CreateFunction`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function createFunction($functionName, $role, $bucketName, $handler)
    {
        //This assumes the Lambda function is in an S3 bucket.
        return $this->customWaiter(function () use ($functionName, $role,
$bucketName, $handler) {
            return $this->lambdaClient->createFunction([
                'Code' => [
                    'S3Bucket' => $bucketName,
                    'S3Key' => $functionName,
                ],
                'FunctionName' => $functionName,
                'Role' => $role['Arn'],
                'Runtime' => 'python3.9',
                'Handler' => "$handler.lambda_handler",
            ]);
        });
    }
```

- For API details, see CreateFunction in *AWS SDK for PHP API Reference*.

`DeleteFunction`

The following code example shows how to use `DeleteFunction`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- For API details, see DeleteFunction in *AWS SDK for PHP API Reference*.

`GetFunction`

The following code example shows how to use `GetFunction`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function getFunction($functionName)
{
    return $this->lambdaClient->getFunction([
        'FunctionName' => $functionName,
    ]);
}
```

- For API details, see GetFunction in *AWS SDK for PHP API Reference*.

`Invoke`

The following code example shows how to use `Invoke`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public function invoke($functionName, $params, $logType = 'None')
{
    return $this->lambdaClient->invoke([
```

```
            'FunctionName' => $functionName,
            'Payload' => json_encode($params),
            'LogType' => $logType,
        ]);
    }
```

- For API details, see Invoke in *AWS SDK for PHP API Reference*.

`ListFunctions`

The following code example shows how to use `ListFunctions`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function listFunctions($maxItems = 50, $marker = null)
    {
        if (is_null($marker)) {
            return $this->lambdaClient->listFunctions([
                'MaxItems' => $maxItems,
            ]);
        }

        return $this->lambdaClient->listFunctions([
            'Marker' => $marker,
            'MaxItems' => $maxItems,
        ]);
    }
```

- For API details, see ListFunctions in *AWS SDK for PHP API Reference*.

`UpdateFunctionCode`

The following code example shows how to use `UpdateFunctionCode`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
    {
        return $this->lambdaClient->updateFunctionCode([
            'FunctionName' => $functionName,
            'S3Bucket' => $s3Bucket,
            'S3Key' => $s3Key,
        ]);
    }
```

- For API details, see UpdateFunctionCode in *AWS SDK for PHP API Reference*.

`UpdateFunctionConfiguration`

The following code example shows how to use `UpdateFunctionConfiguration`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
    {
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
            'Environment' => $environment,
        ]);
```

```
    }
```

- For API details, see UpdateFunctionConfiguration in *AWS SDK for PHP API Reference*.

Scenarios

Get started with functions

The following code example shows how to:

- Create an IAM role and Lambda function, then upload handler code.
- Invoke the function with a single parameter and get results.
- Update the function code and configure with an environment variable.
- Invoke the function with new parameters and get results. Display the returned execution log.
- List the functions for your account, then clean up resources.

For more information, see Create a Lambda function with the console.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace Lambda;

use Aws\S3\S3Client;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithLambda
{
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
```

```
print("Welcome to the AWS Lambda getting started demo using PHP!\n");
echo("-------------------------------------\n");

$clientArgs = [
    'region' => 'us-west-2',
    'version' => 'latest',
    'profile' => 'default',
];
$uniqid = uniqid();

$iamService = new IAMService();
$s3client = new S3Client($clientArgs);
$lambdaService = new LambdaService();

echo "First, let's create a role to run our Lambda code.\n";
$roleName = "test-lambda-role-$uniqid";
$rolePolicyDocument = "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Principal\": {
                \"Service\": \"lambda.amazonaws.com\"
            },
            \"Action\": \"sts:AssumeRole\"
        }
    ]
}";
$role = $iamService->createRole($roleName, $rolePolicyDocument);
echo "Created role {$role['RoleName']}.\n";

$iamService->attachRolePolicy(
    $role['RoleName'],
    "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
);
echo "Attached the AWSLambdaBasicExecutionRole to {$role['RoleName']}.\n";

echo "\nNow let's create an S3 bucket and upload our Lambda code there.\n";
$bucketName = "test-example-bucket-$uniqid";
$s3client->createBucket([
    'Bucket' => $bucketName,
]);
echo "Created bucket $bucketName.\n";
```

```
        $functionName = "doc_example_lambda_$uniqid";
        $codeBasic = __DIR__ . "/lambda_handler_basic.zip";
        $handler = "lambda_handler_basic";
        $file = file_get_contents($codeBasic);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "Uploaded the Lambda code.\n";

        $createLambdaFunction = $lambdaService->createFunction($functionName, $role,
 $bucketName, $handler);
        // Wait until the function has finished being created.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['State'] == "Pending");
        echo "Created Lambda function {$getLambdaFunction['Configuration']
['FunctionName']}.\n";

        sleep(1);

        echo "\nOk, let's invoke that Lambda code.\n";
        $basicParams = [
            'action' => 'increment',
            'number' => 3,
        ];
        /** @var Stream $invokeFunction */
        $invokeFunction = $lambdaService->invoke($functionName, $basicParams)
['Payload'];
        $result = json_decode($invokeFunction->getContents())->result;
        echo "After invoking the Lambda code with the input of
 {$basicParams['number']} we received $result.\n";

        echo "\nSince that's working, let's update the Lambda code.\n";
        $codeCalculator = "lambda_handler_calculator.zip";
        $handlerCalculator = "lambda_handler_calculator";
        echo "First, put the new code into the S3 bucket.\n";
        $file = file_get_contents($codeCalculator);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
```

```
        ]);
        echo "New code uploaded.\n";

        $lambdaService->updateFunctionCode($functionName, $bucketName,
 $functionName);
        // Wait for the Lambda code to finish updating.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "New Lambda code uploaded.\n";

        $environment = [
            'Variable' => ['Variables' => ['LOG_LEVEL' => 'DEBUG']],
        ];
        $lambdaService->updateFunctionConfiguration($functionName,
 $handlerCalculator, $environment);
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "Lambda code updated with new handler and a LOG_LEVEL of DEBUG for more
 information.\n";

        echo "Invoke the new code with some new data.\n";
        $calculatorParams = [
            'action' => 'plus',
            'x' => 5,
            'y' => 4,
        ];
        $invokeFunction = $lambdaService->invoke($functionName, $calculatorParams,
 "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "Indeed, {$calculatorParams['x']} + {$calculatorParams['y']} does equal
 $result.\n";
        echo "Here's the extra debug info: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nBut what happens if you try to divide by zero?\n";
        $divZeroParams = [
            'action' => 'divide',
            'x' => 5,
```

```
            'y' => 0,
        ];
        $invokeFunction = $lambdaService->invoke($functionName, $divZeroParams,
 "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "You get a |$result| result.\n";
        echo "And an error message: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nHere's all the Lambda functions you have in this Region:\n";
        $listLambdaFunctions = $lambdaService->listFunctions(5);
        $allLambdaFunctions = $listLambdaFunctions['Functions'];
        $next = $listLambdaFunctions->get('NextMarker');
        while ($next != false) {
            $listLambdaFunctions = $lambdaService->listFunctions(5, $next);
            $next = $listLambdaFunctions->get('NextMarker');
            $allLambdaFunctions = array_merge($allLambdaFunctions,
 $listLambdaFunctions['Functions']);
        }
        foreach ($allLambdaFunctions as $function) {
            echo "{$function['FunctionName']}\n";
        }

        echo "\n\nAnd don't forget to clean up your data!\n";

        $lambdaService->deleteFunction($functionName);
        echo "Deleted Lambda function.\n";
        $iamService->deleteRole($role['RoleName']);
        echo "Deleted Role.\n";
        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Deleted all objects from the S3 bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);
        echo "Deleted the bucket.\n";
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;
```

```
require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];
```

```
        // Create the connection to the DB
        $conn = new PDO(

 "mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an Kinesis event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
```

```
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be marked
 as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a DynamoDB event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be marked
 as failed
        }

        $totalRecords = count($records);
```

```
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Amazon DocumentDB event with Lambda using PHP.

```
<?php

require __DIR__.'/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }
```

```
    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
 "\n";
    }
}
return new DocumentDBEventHandler();
```

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an S3 event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';


class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' . $bucket .
 '. Make sure they exist and your bucket is in the same region as this function.' .
 "\n";
                throw $e;
            }
        }
    }
}
```

```
$logger = new StderrLogger();
return new Handler($logger);
```

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-
custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
```

```
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be marked
 as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting Kinesis batch item failures with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
```

```
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting DynamoDB batch item failures with Lambda using PHP.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
```

```
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from an SQS queue. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
```

```
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Amazon RDS examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon RDS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

Actions

`CreateDBInstance`

The following code example shows how to use `CreateDBInstance`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;
```

```
$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

$dbIdentifier = '<<{{db-identifier}}>>';
$dbClass = 'db.t2.micro';
$storage = 5;
$engine = 'MySQL';
$username = 'MyUser';
$password = 'MyPassword';

try {
    $result = $rdsClient->createDBInstance([
        'DBInstanceIdentifier' => $dbIdentifier,
        'DBInstanceClass' => $dbClass,
        'AllocatedStorage' => $storage,
        'Engine' => $engine,
        'MasterUsername' => $username,
        'MasterUserPassword' => $password,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see CreateDBInstance in *AWS SDK for PHP API Reference*.

`CreateDBSnapshot`

The following code example shows how to use `CreateDBSnapshot`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;



$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

$dbIdentifier = '<<{{db-identifier}}>>';
$snapshotName = '<<{{backup_2018_12_25}}>>';

try {
    $result = $rdsClient->createDBSnapshot([
        'DBInstanceIdentifier' => $dbIdentifier,
        'DBSnapshotIdentifier' => $snapshotName,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see CreateDBSnapshot in *AWS SDK for PHP API Reference*.

`DeleteDBInstance`

The following code example shows how to use `DeleteDBInstance`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;


//Create an RDSClient
$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-1'
]);

$dbIdentifier = '<<{{db-identifier}}>>';

try {
    $result = $rdsClient->deleteDBInstance([
        'DBInstanceIdentifier' => $dbIdentifier,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see DeleteDBInstance in *AWS SDK for PHP API Reference*.

`DescribeDBInstances`

The following code example shows how to use `DescribeDBInstances`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';
```

```
use Aws\Exception\AwsException;


//Create an RDSClient
$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

try {
    $result = $rdsClient->describeDBInstances();
    foreach ($result['DBInstances'] as $instance) {
        print('<p>DB Identifier: ' . $instance['DBInstanceIdentifier']);
        print('<br />Endpoint: ' . $instance['Endpoint']["Address"]
            . ':' . $instance['Endpoint']["Port"]);
        print('<br />Current Status: ' . $instance["DBInstanceStatus"]);
        print('</p>');
    }
    print(" Raw Result ");
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see DescribeDBInstances in *AWS SDK for PHP API Reference*.

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
```

```
        $generator = new AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
```

```
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Amazon S3 examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon S3.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon S3

The following code examples show how to get started using Amazon S3.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
use Aws\S3\S3Client;

$client = new S3Client(['region' => 'us-west-2']);
$results = $client->listBuckets();
var_dump($results);
```

- For API details, see ListBuckets in *AWS SDK for PHP API Reference*.

Topics

- Actions
- Scenarios
- Serverless examples

Actions

`CopyObject`

The following code example shows how to use `CopyObject`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Simple copy of an object.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $folder = "copied-folder";
            $this->s3client->copyObject([
                'Bucket' => $this->bucketName,
                'CopySource' => "$this->bucketName/$fileName",
                'Key' => "$folder/$fileName-copy",
            ]);
            echo "Copied $fileName to $folder/$fileName-copy.\n";
        } catch (Exception $exception) {
            echo "Failed to copy $fileName with error: " . $exception->getMessage();
            exit("Please fix error with object copying before continuing.");
        }
```

- For API details, see CopyObject in *AWS SDK for PHP API Reference*.

`CreateBucket`

The following code example shows how to use `CreateBucket`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a bucket.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $this->s3client->createBucket([
                'Bucket' => $this->bucketName,
                'CreateBucketConfiguration' => ['LocationConstraint' => $region],
            ]);
            echo "Created bucket named: $this->bucketName \n";
        } catch (Exception $exception) {
```

```
            echo "Failed to create bucket $this->bucketName with error: " .
 $exception->getMessage();
            exit("Please fix error with bucket creation before continuing.");
        }
```

- For API details, see CreateBucket in *AWS SDK for PHP API Reference*.

`DeleteBucket`

The following code example shows how to use `DeleteBucket`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete an empty bucket.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $this->s3client->deleteBucket([
                'Bucket' => $this->bucketName,
            ]);
            echo "Deleted bucket $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to delete $this->bucketName with error: " . $exception-
>getMessage();
            exit("Please fix error with bucket deletion before continuing.");
        }
```

- For API details, see DeleteBucket in *AWS SDK for PHP API Reference*.

`DeleteObjects`

The following code example shows how to use `DeleteObjects`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a set of objects from a list of keys.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $objects = [];
            foreach ($contents['Contents'] as $content) {
                $objects[] = [
                    'Key' => $content['Key'],
                ];
            }
            $this->s3client->deleteObjects([
                'Bucket' => $this->bucketName,
                'Delete' => [
                    'Objects' => $objects,
                ],
            ]);
            $check = $this->s3client->listObjectsV2([
                'Bucket' => $this->bucketName,
            ]);
            if (count($check) <= 0) {
                throw new Exception("Bucket wasn't empty.");
            }
            echo "Deleted all objects and folders from $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to delete $fileName from $this->bucketName with error: " .
$exception->getMessage();
            exit("Please fix error with object deletion before continuing.");
        }
```

- For API details, see DeleteObjects in *AWS SDK for PHP API Reference*.

`GetObject`

The following code example shows how to use `GetObject`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get an object.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $file = $this->s3client->getObject([
                'Bucket' => $this->bucketName,
                'Key' => $fileName,
            ]);
            $body = $file->get('Body');
            $body->rewind();
            echo "Downloaded the file and it begins with: {$body->read(26)}.\n";
        } catch (Exception $exception) {
            echo "Failed to download $fileName from $this->bucketName with error:
" . $exception->getMessage();
            exit("Please fix error with file downloading before continuing.");
        }
```

- For API details, see GetObject in *AWS SDK for PHP API Reference*.

`ListObjectsV2`

The following code example shows how to use `ListObjectsV2`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List objects in a bucket.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        try {
            $contents = $this->s3client->listObjectsV2([
                'Bucket' => $this->bucketName,
            ]);
            echo "The contents of your bucket are: \n";
            foreach ($contents['Contents'] as $content) {
                echo $content['Key'] . "\n";
            }
        } catch (Exception $exception) {
            echo "Failed to list objects in $this->bucketName with error: " .
$exception->getMessage();
            exit("Please fix error with listing objects before continuing.");
        }
```

- For API details, see ListObjectsV2 in *AWS SDK for PHP API Reference*.

`PutObject`

The following code example shows how to use `PutObject`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Upload an object to a bucket.

```
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);

        $fileName = __DIR__ . "/local-file-" . uniqid();
        try {
            $this->s3client->putObject([
                'Bucket' => $this->bucketName,
                'Key' => $fileName,
                'SourceFile' => __DIR__ . '/testfile.txt'
            ]);
            echo "Uploaded $fileName to $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to upload $fileName with error: " . $exception-
>getMessage();
            exit("Please fix error with file upload before continuing.");
        }
```

- For API details, see PutObject in *AWS SDK for PHP API Reference*.

Scenarios

Create a presigned URL

The following code example shows how to create a presigned URL for Amazon S3 and upload an object.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace S3;
use Aws\Exception\AwsException;
use AwsUtilities\PrintableLineBreak;
use AwsUtilities\TestableReadline;
use DateTime;
```

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

class PresignedURL
{
    use PrintableLineBreak;
    use TestableReadline;

    public function run()
    {
        $s3Service = new S3Service();

        $expiration = new DateTime("+20 minutes");
        $linebreak = $this->getLineBreak();

        echo $linebreak;
        echo ("Welcome to the Amazon S3 presigned URL demo.\n");
        echo $linebreak;

        $bucket = $this->testable_readline("First, please enter the name of the S3
 bucket to use: ");
        $key = $this->testable_readline("Next, provide the key of an object in the
 given bucket: ");
        echo $linebreak;
        $command = $s3Service->getClient()->getCommand('GetObject', [
            'Bucket' => $bucket,
            'Key' => $key,
        ]);
        try {
            $preSignedUrl = $s3Service->preSignedUrl($command, $expiration);
            echo "Your preSignedUrl is \n$preSignedUrl\nand will be good for the
 next 20 minutes.\n";
            echo $linebreak;
            echo "Thanks for trying the Amazon S3 presigned URL demo.\n";
        } catch (AwsException $exception) {
            echo $linebreak;
            echo "Something went wrong: $exception";
            die();
        }
    }
}

$runner = new PresignedURL();
$runner->run();
```

Get started with buckets and objects

The following code example shows how to:

- Create a bucket and upload a file to it.
- Download an object from a bucket.
- Copy an object to a subfolder in a bucket.
- List the objects in a bucket.
- Delete the bucket objects and the bucket.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the Amazon S3 getting started demo using PHP!\n");
        echo("---------------------------------------\n");

        $region = 'us-west-2';

        $this->s3client = new S3Client([
                'region' => $region,
        ]);
        /* Inline declaration example
        $s3client = new Aws\S3\S3Client(['region' => 'us-west-2']);
        */

        $this->bucketName = "doc-example-bucket-" . uniqid();

        try {
            $this->s3client->createBucket([
```

```
                'Bucket' => $this->bucketName,
                'CreateBucketConfiguration' => ['LocationConstraint' => $region],
            ]);
            echo "Created bucket named: $this->bucketName \n";
        } catch (Exception $exception) {
            echo "Failed to create bucket $this->bucketName with error: " .
 $exception->getMessage();
            exit("Please fix error with bucket creation before continuing.");
        }

        $fileName = __DIR__ . "/local-file-" . uniqid();
        try {
            $this->s3client->putObject([
                'Bucket' => $this->bucketName,
                'Key' => $fileName,
                'SourceFile' => __DIR__ . '/testfile.txt'
            ]);
            echo "Uploaded $fileName to $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to upload $fileName with error: " . $exception-
>getMessage();
            exit("Please fix error with file upload before continuing.");
        }

        try {
            $file = $this->s3client->getObject([
                'Bucket' => $this->bucketName,
                'Key' => $fileName,
            ]);
            $body = $file->get('Body');
            $body->rewind();
            echo "Downloaded the file and it begins with: {$body->read(26)}.\n";
        } catch (Exception $exception) {
            echo "Failed to download $fileName from $this->bucketName with error:
 " . $exception->getMessage();
            exit("Please fix error with file downloading before continuing.");
        }

        try {
            $folder = "copied-folder";
            $this->s3client->copyObject([
                'Bucket' => $this->bucketName,
                'CopySource' => "$this->bucketName/$fileName",
                'Key' => "$folder/$fileName-copy",
```

```
            ]);
            echo "Copied $fileName to $folder/$fileName-copy.\n";
        } catch (Exception $exception) {
            echo "Failed to copy $fileName with error: " . $exception->getMessage();
            exit("Please fix error with object copying before continuing.");
        }

        try {
            $contents = $this->s3client->listObjectsV2([
                'Bucket' => $this->bucketName,
            ]);
            echo "The contents of your bucket are: \n";
            foreach ($contents['Contents'] as $content) {
                echo $content['Key'] . "\n";
            }
        } catch (Exception $exception) {
            echo "Failed to list objects in $this->bucketName with error: " .
$exception->getMessage();
            exit("Please fix error with listing objects before continuing.");
        }

        try {
            $objects = [];
            foreach ($contents['Contents'] as $content) {
                $objects[] = [
                    'Key' => $content['Key'],
                ];
            }
            $this->s3client->deleteObjects([
                'Bucket' => $this->bucketName,
                'Delete' => [
                    'Objects' => $objects,
                ],
            ]);
            $check = $this->s3client->listObjectsV2([
                'Bucket' => $this->bucketName,
            ]);
            if (count($check) <= 0) {
                throw new Exception("Bucket wasn't empty.");
            }
            echo "Deleted all objects and folders from $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to delete $fileName from $this->bucketName with error: " .
$exception->getMessage();
```

```
            exit("Please fix error with object deletion before continuing.");
        }

        try {
            $this->s3client->deleteBucket([
                'Bucket' => $this->bucketName,
            ]);
            echo "Deleted bucket $this->bucketName.\n";
        } catch (Exception $exception) {
            echo "Failed to delete $this->bucketName with error: " . $exception-
>getMessage();
            exit("Please fix error with bucket deletion before continuing.");
        }

        echo "Successfully ran the Amazon S3 with PHP demo.\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Serverless examples

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an S3 event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';


class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
```

```
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' . $bucket .
 '. Make sure they exist and your bucket is in the same region as this function.' .
 "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Amazon SNS examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon SNS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios
- Serverless examples

Actions

`CheckIfPhoneNumberIsOptedOut`

The following code example shows how to use `CheckIfPhoneNumberIsOptedOut`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Indicates whether the phone number owner has opted out of receiving SMS messages
 from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see CheckIfPhoneNumberIsOptedOut in *AWS SDK for PHP API Reference*.

`ConfirmSubscription`

The following code example shows how to use `ConfirmSubscription`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Verifies an endpoint owner's intent to receive messages by
 * validating the token sent to the endpoint by an earlier Subscribe action.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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->confirmSubscription([
        'Token' => $subscription_token,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For API details, see ConfirmSubscription in *AWS SDK for PHP API Reference*.

`CreateTopic`

The following code example shows how to use `CreateTopic`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Create a Simple Notification Service topics in your AWS account at the requested
 region.
 *
 * This code expects that you have AWS credentials set up per:
```

```
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see CreateTopic in *AWS SDK for PHP API Reference*.

`DeleteTopic`

The following code example shows how to use `DeleteTopic`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
```

```
use Aws\Sns\SnsClient;


/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For API details, see DeleteTopic in *AWS SDK for PHP API Reference*.

`GetSMSAttributes`

The following code example shows how to use `GetSMSAttributes`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Get the type of SMS Message sent by default from the AWS SNS service.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->getSMSAttributes([
        'attributes' => ['DefaultSMSType'],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see GetSMSAttributes in *AWS SDK for PHP API Reference*.

`GetTopicAttributes`

The following code example shows how to use `GetTopicAttributes`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$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());
}
```

- For API details, see GetTopicAttributes in *AWS SDK for PHP API Reference*.

`ListPhoneNumbersOptedOut`

The following code example shows how to use `ListPhoneNumbersOptedOut`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages from
 your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see ListPhoneNumbersOptedOut in *AWS SDK for PHP API Reference*.

`ListSubscriptions`

The following code example shows how to use `ListSubscriptions`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Returns a list of Amazon SNS subscriptions in the requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For API details, see ListSubscriptions in *AWS SDK for PHP API Reference*.

`ListTopics`

The following code example shows how to use `ListTopics`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Returns a list of the requester's topics from your AWS SNS account in the region
 specified.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For API details, see ListTopics in *AWS SDK for PHP API Reference*.

`Publish`

The following code example shows how to use `Publish`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a 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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

`SetSMSAttributes`

The following code example shows how to use `SetSMSAttributes`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see SetSMSAttributes in *AWS SDK for PHP API Reference*.

`SetTopicAttributes`

The following code example shows how to use `SetTopicAttributes`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For API details, see SetTopicAttributes in *AWS SDK for PHP API Reference*.

`Subscribe`

The following code example shows how to use `Subscribe`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Subscribe an email address to a topic.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
```

```
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$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 an HTTP endpoint to a topic.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For API details, see Subscribe in *AWS SDK for PHP API Reference*.

`Unsubscribe`

The following code example shows how to use `Unsubscribe`.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Deletes a subscription to an Amazon SNS topic.
```

```
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Unsubscribe in *AWS SDK for PHP API Reference*.

Scenarios

Publish an SMS text message

The following code example shows how to publish SMS messages using Amazon SNS.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;


/**
 * Sends a text message (SMS message) directly to a phone number using Amazon SNS.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/guide_credentials.html
 */

$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());
}
```

- For more information, see AWS SDK for PHP Developer Guide.
- For API details, see Publish in *AWS SDK for PHP API Reference*.

Serverless examples

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-
custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
```

```
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be marked
 as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Amazon SQS examples using SDK for PHP

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for PHP with Amazon SQS.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

Topics

- Serverless examples

Serverless examples

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
```

```
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from an SQS queue. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';
```

```
class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Cross-service examples using SDK for PHP

The following sample applications use the AWS SDK for PHP to work across multiple AWS services.

Cross-service examples target an advanced level of experience to help you start building applications.

Examples

- Create a photo asset management application that lets users manage photos using labels
- Create an Aurora Serverless work item tracker

Create a photo asset management application that lets users manage photos using labels

SDK for PHP

Shows how to develop a photo asset management application that detects labels in images using Amazon Rekognition and stores them for later retrieval.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

For a deep dive into the origin of this example see the post on AWS Community.

Services used in this example

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Create an Aurora Serverless work item tracker

SDK for PHP

Shows how to use the AWS SDK for PHP to create a web application that tracks work items in an Amazon RDS database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with a RESTful PHP backend.

- Integrate a React.js web application with AWS services.

- List, add, update, and delete items in an Amazon RDS table.
- Send an email report of filtered work items using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

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

The AWS shared responsibility model applies to data protection in . 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. You are also responsible for 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 users with AWS IAM Identity Center or AWS Identity and Access Management (IAM).

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 require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 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) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS SDK for PHP or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Identity and Access Management

AWS Identity and Access Management (IAM) is an 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 AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS services work with IAM
- Troubleshooting AWS identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

Service user – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see Troubleshooting AWS identity and access or the user guide of the AWS service you are using.

Service administrator – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If

you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating

IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource

(instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that

support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics

- I am not authorized to perform an action in AWS
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS resources

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `awes:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *my-example-widget* resource by using the `awes:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see How AWS services work with IAM.

- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map

the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service 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.

Resilience for this AWS Product or Service

The 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 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.

This AWS product or service 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 this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service 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.

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

```
use Aws\S3\Crypto\S3EncryptionClient;
use Aws\S3\S3Client;

$encryptionClient = new S3EncryptionClient(
    new S3Client([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ])
);
```

Post-migration

```
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 AWS KMS with kms-key-id

> **Note**
>
> These examples use imports and variables defined in Example 1. For example, `$encryptionClient`.

Pre-migration

```
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,
]);
```

Post-migration

```
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.

```
// 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.

```
$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 `S3PutObject` operation using the `Aws\Middleware::mapRequest` helper method.

See mapRequest for more information.

How can I sign an arbitrary request?

You can sign an arbitrary :aws-php-class: *PSR-7 request <class-Psr.Http.Message.RequestInterface.html>* using the SDK's :aws-php-class: *SignatureV4 class <class-Aws.Signature.SignatureV4.html>*.

See Signing Custom Amazon CloudSearch Domain Requests with AWS SDK for PHP Version 3 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.

See mapCommand for more information.

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.

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 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 in the AWS General Reference for a list of regions, endpoints, and the supported schemes.

```
$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.

If you encounter a parsing error, check your system and be sure it fulfills the SDK’s Requirements and Recommendations for the AWS SDK for PHP Version 3.

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 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 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.

```
$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`, `ProvisionedThroughputExceededException`, `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 AwsExceptionAwsException. 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`.

```
$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 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 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 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 guide for more details.

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 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 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 let's you provide a set of global configuration values 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 guide for more details.

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.

Document history

The following tables describe the important changes since the last release of the AWS SDK for PHP Developer Guide.

Most recent changes:

Change	Description	Date
Amazon EventBridge global endpoints	Add code example that shows how to use Amazon EventBridge global endpoints	December 22, 2023
AWS Common Runtime (AWS CRT)	Add a topic that discusses the use of the AWS Common Runtime (AWS CRT) by the SDK for PHP.	November 17, 2023
StreamWrapper mkdir() updates	Add information about working with buckets and folder objects by using `mkdir()`.	November 2, 2023
Service client creation	Update code snippets by removing 'version' parameter since the 'latest' is the default.	August 31, 2023
Table of contents	Updated table of contents to make code examples more accessible.	June 1, 2023
IAM best practices updates	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM. Updates to Getting started.	May 20, 2023

Amazon S3 transfer manager	Added the `add_content_md5` transfer option.	April 13, 2023
Amazon S3 multipart uploads	Included configuration information for synchronous uploads. Added the `add_content_md5` upload option for asynchronous uploads.	April 13, 2023
Reference information	Added multiple links to relevant detail content in the AWS SDKs and Tools Reference Guide. Updated guide formatting.	September 14, 2022
General cleanup	Added references to the AWS SDKs and Tools Reference Guide. Updated AWS Key Management Service sections to reflect terminology updates.	August 23, 2022
Working with AWS services	Included lists of the code examples that are available on GitHub.	April 1, 2022
Enabling SDK metrics	Removed information about enabling SDK metrics, which was deprecated on December 20, 2021.	January 27, 2022
Amazon S3 encryption client migration	Added topic on Amazon S3 encryption client migration	August 7, 2020

Older changes:

Change	Description	Release date
Secrets Manager examples	Add more service examples	Mar 27th, 2019
Endpoint discovery	Configuration of endpoint discovery	Feb 15th, 2019
Amazon CloudFront	Add more service examples	Jan 25th, 2019
Service features	SDK metrics	Jan 11th, 2018
Amazon Kinesis, Amazon SNS	Add more service examples	Dec 14th, 2018
Amazon SES examples	Add more service examples	Oct 5th, 2018
AWS KMS examples	Add more service examples	Aug 8th, 2018
Credentials	Clarify and simplify credentia ls guide	June 30th, 2018
MediaConvert examples	Add more service examples	June 15th, 2018
New web layout	Documentation switched to AWS style	May 9th, 2018
Amazon S3 encryption	Client side encryption	Nov 17th, 2017
Amazon S3, Amazon SQS	Add more service examples	Mar 26th, 2017
Amazon S3, IAM, Amazon EC2	Add more service examples	Mar 17th, 2017
Add credentials	Adds support for AssumeRole and ini	Jan 17th, 2017
S3 Examples	S3 Multi-Region and presigned posts	Mar 18th, 2016
OpenSearch Service and Amazon CloudSearch	Add more service examples	Dec 28th, 2015
Command line	Add command parameters	Aug 13th, 2015

Change	Description	Release date
Service features	Adds service features for S3 and AWS	Apr 30th, 2015
New SDK version	Version 3 of the AWS SDK for PHP released.	May 26th, 2015