aws

版本 3 开发人员指南

适用于 PHP 的 Amazon SDK

适用于 PHP 的 Amazon SDK: 版本 3 开发人员指南

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 中国的 Amazon Web Services 服务入门 (PDF)。

Table of Contents

适用于 PHP 的 Amazon SDK 版本3是什么？

适用于 PHP 的 Amazon SDK 版本 3 使 PHP 开发人员能够在 PHP 代码中使用亚马逊 Web Services，并使用亚马逊 S3、亚马逊 DynamoDB 和 Amazon Glacier 等服务构建强大的应用程序和软件。通过 Composer 安装 SDK（需要 `aws/aws-sdk-php` 包），在几分钟之内就可以开始使用，也可以下载单独的 `aws.zip` 或 `aws.phar` 文件进行安装。

开发工具包中的服务并不是全部直接可用的。要了解目前支持哪些服务 适用于 PHP 的 Amazon SDK，请参阅服务名称和 API 版本。

> **Note**
>
> 如果您要将使用 SDK 版本 2 的代码迁移到版本 3，请务必阅读从 适用于 PHP 的 Amazon SDK版本 2 升级。

开始使用 SDK

如果您已准备动手操作 SDK，请遵循章节 适用于 PHP 的 Amazon SDK 版本 3 入门。它会指导您完成身份验证 Amazon、设置开发环境以及使用 Amazon S3 创建您的第一个基本应用程序。

其他资源

- 常见问题解答
- 术语表
- Amazon SDKs 和《工具参考指南》：包含设置、功能和其他常见的基本概念。 Amazon SDKs
- Guzzle 文档
- awsdocs 适用于 PHP 的 Amazon SDK aws-doc-sdk-examples/存储库中提供了使用的代码示例。
- Gitter 上的 PHP SDK 社区。
- Amazon Web Services re:Post.

GitHub:

- 的源代码可在 适用于 PHP 的 Amazon SDK aws/ aws-sdk-php 存储库中找到。
- 为开发工具包做出贡献

- 报告错误或请求功能

API 文档

请查找 SDK 的 API 文档https://docs.aws.amazon.com/sdk-for-php/最新/reference/。

SDK 主要版本的维护和支持

有关 SDK 主要版本及其底层依赖项的维护和支持的信息，请参阅《Amazon SDKs 和工具参考指南》中的以下内容：

- Amazon SDKs 和工具维护政策
- Amazon SDKs 和 “工具” 版本支持矩阵

适用于 PHP 的 Amazon SDK 版本 3 入门

本章专门帮助您启动和运行适用于 PHP 的 Amazon SDK版本 3。

主题

适用于 PHP 的 Amazon SDK 版本 3 的要求和建议

要获得最佳效果 适用于 PHP 的 Amazon SDK，请确保您的环境支持以下要求和建议。

要求

要使用 适用于 PHP 的 Amazon SDK，必须使用启用了 SimplexML PHP 扩展的 PHP 版本 8.1 或更高版本。如果你需要签署私有亚马逊 CloudFront URLs，你还需要使用 OpenSSL PHP 扩展程序。

建议

除了最低要求之外，我们还建议您安装、卸载并使用以下内容。

安装 cURL 7.16.2 或更高版本	使用使用 OpenSSL/NSS 和 zlib 编译的 curl 的最新版本。如果您的系统中没有安装 cURL，并且您也没有为客户端配置自定义 http_handler，开发工具包将使用 PHP 流封装程序。
使用 OPCache	使用该 OPcache 扩展通过将预编译的脚本字节码存储在共享内存中来提高 PHP 性能。这样 PHP 就不需要在每次请求时加载并解析脚本。默认情况下，此扩展通常为启用状态。

	运行亚马逊 Linux 时，你需要安装 php56-opcache 或 php55-opcache yum 包才能使用该扩展程序。 OPCache
在生产环境中卸载 Xdebug	Xdebug 有助于发现性能瓶颈。但如果性能对于您的应用程序至关重要，请不要在您的生产环境中安装 Xdebug 扩展。加载此扩展会显著降低开发工具包的性能。
使用 Composer 类映射自动加载工具	自动加载工具可加载 PHP 脚本需要的类。Composer 生成自动加载工具，自动加载您的应用程序的 PHP 脚本，以及您的应用程序需要的其他所有 PHP 脚本，包括 适用于 PHP 的 Amazon SDK。 我们建议您针对生产环境使用类映射自动加载工具，以提升自动加载工具的性能。将 `-o` 或 `==optimize-autoloader` 选项传递到 Composer 的安装命令，可生成类映射自动加载工具。

兼容性测试

运行 SDK 代码库中的 `compatibility-test.php` 文件，验证系统是否可以运行 SDK。兼容性测试除了可以检查是否满足开发工具包的最低系统要求外，还可以检查是否有可选设置，并提供有助于提升性能的建议。兼容性测试可将结果输出到命令行或 Web 浏览器。如果在浏览器中检查测试结果，成功完成的检查是绿色的，警告是紫色，失败是红色。如果从命令行运行，每项检查的结果是单独的一行。

如果要报告开发工具包的问题，提供兼容性测试的输出有助于发现潜在原因。

安装 适用于 PHP 的 Amazon SDK 版本 3

你可以安装 适用于 PHP 的 Amazon SDK 版本 3：

- 作为依赖项（通过 Composer）
- 作为开发工具包的自带 phar
- 作为开发工具包的 ZIP 文件

在安装 适用于 PHP 的 Amazon SDK 版本 3 之前，请确保您的环境使用的是 PHP 版本 8.1 或更高版本。了解有关环境要求和建议的更多信息。

> **Note**
>
> 通过 .phar 和 .zip 方法安装 SDK，需要单独安装和启用多字节字符串 PHP 扩展。

通过 Composer 适用于 PHP 的 Amazon SDK 作为依赖项安装

Composer 是安装 适用于 PHP 的 Amazon SDK的推荐方式。Composer 是一款 PHP 工具，用于管理和安装项目的依赖项。

有关如何安装 Composer、配置自动加载并遵循定义依赖关系的其他最佳实践的更多信息，请参阅 getcomposer.org。

安装 Composer

如果 Composer 未在您的项目中，请从 Download Composer 页面下载并安装 Composer。

- 对于 Windows，请按照 Windows 安装程序说明进行操作。
- 对于 Linux，请按照命令行安装说明进行操作。

通过 Composer 添加 适用于 PHP 的 Amazon SDK 为依赖项

如果您的系统上已经全局安装了 Composer，请在项目的基目录中运行以下命令以 适用于 PHP 的 Amazon SDK 作为依赖项进行安装：

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

否则，请键入此 Composer 命令以安装最新版本的 适用于 PHP 的 Amazon SDK 作为依赖项。

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

将自动加载工具添加到 php 脚本

Installing Composer 在环境中创建多个文件夹和文件。您将使用的主要文件为 `autoload.php`，位于环境中的 `vendor` 文件夹中。

要在脚本 适用于 PHP 的 Amazon SDK 中使用，请在脚本中加入自动加载器，如下所示。

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

使用打包的 Phar 进行安装

的每个版本都 适用于 PHP 的 Amazon SDK 包含一个预打包的 phar（PHP 存档），其中包含运行 SDK 所需的所有类和依赖项。此外，phar 会自动为 适用于 PHP 的 Amazon SDK 及其所有依赖项注册一个类自动加载器。

您可以下载打包的 phar 并把它包含到您的脚本中。

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

> **Note**
>
> 不建议使用带有 Suhosin 补丁的 PHP，但这对于 Ubuntu 和 Debian 分发版是很常见的。在这种情况下，您可能需要在 suhosin.ini 中启用 phar。否则，在您的代码中包含 phar 文件将导致无错误提示的故障。要修改 suhosin.ini，请添加以下行。
>
> ```
> suhosin.executor.include.whitelist = phar
> ```

使用 ZIP 文件进行安装

适用于 PHP 的 Amazon SDK 包括一个 ZIP 文件，其中包含运行 SDK 所需的所有类和依赖关系。此外，这个 ZIP 文件还包含适用于 适用于 PHP 的 Amazon SDK 及其依赖项的类自动加载工具。

要安装开发工具包，请下载 .zip 文件，然后在项目中的选定位置进行解压缩。然后将自动加载工具包含到您的脚本中，如下所示。

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

Amazon 使用 适用于 PHP 的 Amazon SDK 版本 3 进行身份验证

使用开发 Amazon 时，您必须确定您的代码是如何进行身份验证的。 Amazon Web Services 服务您可以根据环境和可用的访问权限以不同的方式配置对 Amazon 资源的编程 Amazon 访问权限。

要选择您的身份验证方法并针对 SDK 进行配置，请参阅和工具参考指南中的身份验证Amazon SDKs和访问。

使用 IAM Identity Center 设置身份验证以进行本地开发

我们建议在本地开发且雇主未向其提供身份验证方法的新用户进行设置 Amazon IAM Identity Center。此方法包括安装 Amazon CLI 以便于配置和定期登录 Amazon 访问门户。如果您选择此方法，则在完成Amazon SDKs 和工具参考指南中的 IAM Identity Center 身份验证程序后，您的环境应包含以下元素：

- Amazon CLI，用于在运行应用程序之前启动 Amazon 访问门户会话。
- 共享 Amazonconfig 文件，其 [default] 配置文件包含一组可由 SDK 引用的配置值。要查找此文件的位置，请参阅Amazon SDKs 和工具参考指南中的共享文件的位置。
- 共享 config 文件包含 region 设置。这将设置 SDK 用于请求的默认值 Amazon Web Services 区域 。此区域用于未显式配置 region 属性的 SDK 服务请求。
- 在向 Amazon发送请求之前，SDK 使用配置文件的 SSO 令牌提供程序配置来获取凭证。该sso_role_name值是与 IAM 身份中心权限集关联的 IAM 角色，允许访问您的应用程序中的用户。 Amazon Web Services 服务

  以下示例 config 文件展示了使用 SSO 令牌提供程序配置来设置的默认配置文件。配置文件的 sso_session 设置是指所指定的 sso-session 节。该sso-session部分包含启动 Amazon 访问门户会话的设置。

```
[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
```

适用于 PHP 的 Amazon SDK 无需向您的应用程序添加其他软件包（例如SSO和SSOOIDC）即可使用 IAM Identity Center 身份验证。

启动 Amazon 访问门户会话

在运行可访问的应用程序之前 Amazon Web Services 服务，您需要为软件开发工具包进行有效的 Amazon 访问门户会话，才能使用 IAM Identity Center 身份验证来解析证书。根据配置的会话时长，访问权限最终将过期，并且开发工具包将遇到身份验证错误。要登录 Amazon 访问门户，请在中运行以下命令 Amazon CLI。

```
aws sso login
```

如果遵循引导并具有默认的配置文件设置，则无需使用 `--profile` 选项来调用该命令。如果您的 SSO 令牌提供程序配置在使用指定的配置文件，则命令为 `aws sso login --profile named-profile`。

要选择性地测试是否已有活动会话，请运行以下 Amazon CLI 命令。

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

如果会话是活动的，则对此命令的响应会报告共享 `config` 文件中配置的 IAM Identity Center 账户和权限集。

> **Note**
>
> 如果您已经有一个有效的 Amazon 访问门户会话并且`aws sso login`正在运行，则无需提供凭据。
>
> 登录过程可能会提示您允许 Amazon CLI 访问您的数据。由于 Amazon CLI 是在适用于 Python 的 SDK 之上构建的，因此权限消息可能包含`botocore`名称的变体。

了解有关身份验证的更多信息

- 有关使用 IAM Identity Center 进行身份验证的更多详细信息，请参阅工具参考指南中的Amazon SDKs 了解 IAM 身份中心身份验证
- 有关最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。
- 要创建短期 Amazon 证书，请参阅 IAM 用户指南中的临时安全证书。

- 要了解其他 适用于 PHP 的 Amazon SDK 可以使用的凭证提供商，请参阅《工具参考指南》Amazon SDKs 中的标准化凭证提供商。

使用适用于 PHP 的 Amazon SDK 版本 3 创建简单的应用程序

使用 适用于 PHP 的 Amazon SDK 来给 Amazon S3 打招呼。以下示例显示了 Amazon S3 存储桶的列表。

先决条件

- 下载并安装 SDK
- 使用适用于 PHP 的 Amazon SDK 之前，必须先使用 Amazon 设置身份验证。有关设置身份验证的信息，请参阅 Amazon 使用 适用于 PHP 的 Amazon SDK 版本 3 进行身份验证

在您的代码中包含 SDK

无论您使用哪种方式安装开发工具包，都可以通过单独的 `require` 语句在您的代码中包含开发工具包。请参阅以下 PHP 代码表，了解符合您的安装方式的代码。请使用系统的实际路径替换 `/path/to/` 的任何实例。

安装方法	所需语句
使用 Composer	`require '/path/to/vendor/autoload.php';`
使用 phar	`require '/path/to/aws.phar';`
使用 ZIP	`require '/path/to/aws-autoloader.php';`

在此主题中，我们假设了 Composer 安装方法。如果您使用其他安装方法，可以回到这一部分来查找应使用的正确 `require` 代码。

编写代码

确保可以进行身份验证。

复制并在新的源文件中粘贴以下代码。保存并将文件命名为 hello-s3.php。

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

/**
 * List your Amazon S3 buckets.
 */

//Create a S3Client
// snippet-start:[s3.php.list_buckets.main]
$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";
}
```

运行程序

打开命令提示符以运行 PHP 程序。运行 PHP 程序的典型命令语法是：

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

此示例代码不使用任何参数。要运行此代码，请在命令提示符下输入以下内容：

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

后续步骤

要测试更多其他 Amazon S3 操作，请查看 GitHub 上的 Amazon 代码示例存储库。

Amazon Cloud9 与 适用于 PHP 的 Amazon SDK 版本 3 一起使用

> **Note**
>
> Amazon Cloud9 不再向新客户提供。的现有客户 Amazon Cloud9 可以继续照常使用该服务。了解更多。

Amazon Cloud9 是一个基于 Web 的集成开发环境 (IDE)，其中包含用于在云中编码、构建、运行、测试、调试和发布软件的一系列工具。您可以使用 Amazon Cloud9 浏览器编写和运行 PHP 代码。适用于 PHP 的 Amazon SDK Amazon Cloud9 包括代码编辑器和终端等工具。由于 Amazon Cloud9 IDE 基于云端，因此您可以使用联网的计算机在办公室、家中或任何地方处理项目。有关的一般信息 Amazon Cloud9，请参阅《Amazon Cloud9 用户指南》。

按照以下说明 Amazon Cloud9 进行设置 适用于 PHP 的 Amazon SDK：

- 第 1 步：设置 Amazon Web Services 账户 要使用 Amazon Cloud9
- 第 2 步：设置 Amazon Cloud9 开发环境
- 第 3 步：设置 适用于 PHP 的 Amazon SDK
- 步骤 4：下载示例代码
- 步骤 5：运行示例代码

第 1 步：设置 Amazon Web Services 账户 要使用 Amazon Cloud9

要使用 Amazon Cloud9，请从登录到 Amazon Cloud9 控制台 Amazon Web Services 管理控制台。

> **Note**
>
> 如果您使用 Amazon IAM Identity Center 进行身份验证，则可能需要在 IAM 控制台中`iam:ListInstanceProfilesForRole`向用户附加的策略中添加所需的权限。

要在您的 Amazon 账户中设置 IAM 实体以访问 Amazon Cloud9 和登录 Amazon Cloud9 控制台，请参阅Amazon Cloud9 用户指南 Amazon Cloud9中的团队设置。

第 2 步：设置 Amazon Cloud9 开发环境

登录 Amazon Cloud9 控制台后，使用控制台创建 Amazon Cloud9 开发环境。创建环境后， Amazon Cloud9 打开该环境的 IDE。

有关详细信息，请参阅《Amazon Cloud9 用户指南》中的在 Amazon Cloud9中创建环境。

> **Note**
>
> 在控制台中首次创建环境之后，我们建议您选择创建新的环境实例 (EC2)。此选项指示 Amazon Cloud9 创建环境，启动 Amazon EC2 实例，然后将新实例连接到新环境。这是开始使用的最快方法 Amazon Cloud9。

如果终端未在 IDE 中打开，请打开它。在 IDE 中的菜单栏上，选择窗口、新终端。您可以使用终端窗口来安装工具和构建应用程序。

第 3 步：设置 适用于 PHP 的 Amazon SDK

Amazon Cloud9 打开开发环境的 IDE 后，使用终端窗口 适用于 PHP 的 Amazon SDK 在您的环境中进行设置。

Composer 是安装 适用于 PHP 的 Amazon SDK的推荐方式。Composer 是一款 PHP 工具，用于管理和安装项目的依赖项。

有关如何安装 Composer、配置自动加载并遵循定义依赖关系的其他最佳实践的更多信息，请参阅 getcomposer.org。

安装 Composer

如果 Composer 未在您的项目中，请从 Download Composer 页面下载并安装 Composer。

- 对于 Windows，请按照 Windows 安装程序说明进行操作。
- 对于 Linux，请按照命令行安装说明进行操作。

通过 Composer 添加 适用于 PHP 的 Amazon SDK 为依赖项

如果您的系统上已经全局安装了 Composer，请在项目的基目录中运行以下命令以 适用于 PHP 的 Amazon SDK 作为依赖项进行安装：

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

否则，请键入此 Composer 命令以安装最新版本的 适用于 PHP 的 Amazon SDK 作为依赖项。

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

将自动加载工具添加到 php 脚本

Installing Composer 在环境中创建多个文件夹和文件。您将使用的主要文件为 `autoload.php`，位于环境中的 `vendor` 文件夹中。

要在脚本 适用于 PHP 的 Amazon SDK 中使用，请在脚本中加入自动加载器，如下所示。

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

步骤 4：下载示例代码

使用终端窗口将的示例代码下载 适用于 PHP 的 Amazon SDK 到 Amazon Cloud9 开发环境中。

要将官方 Amazon SDK 文档中使用的所有代码示例的副本下载到您环境的根目录中，请运行以下命令：

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

的代码示例位于`ENVIRONMENT_NAME/aws-doc-sdk-examples/php`目录，其中`ENVIRONMENT_NAME`是您的开发环境的名称。 适用于 PHP 的 Amazon SDK

要继续使用 Amazon S3 示例，我们建议从代码示例 `ENVIRONMENT_NAME/aws-doc-sdk-examples/php/example_code/s3/ListBuckets.php` 开始。此示例将列出 Amazon S3 桶。使用终端窗口导航到 `s3` 目录并列出文件。

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

要在中打开文件 Amazon Cloud9，可以`ListBuckets.php`直接在终端窗口中单击。

有关理解代码示例的更多支持，请参阅 适用于 PHP 的 Amazon SDK 代码示例。

步骤 5：运行示例代码

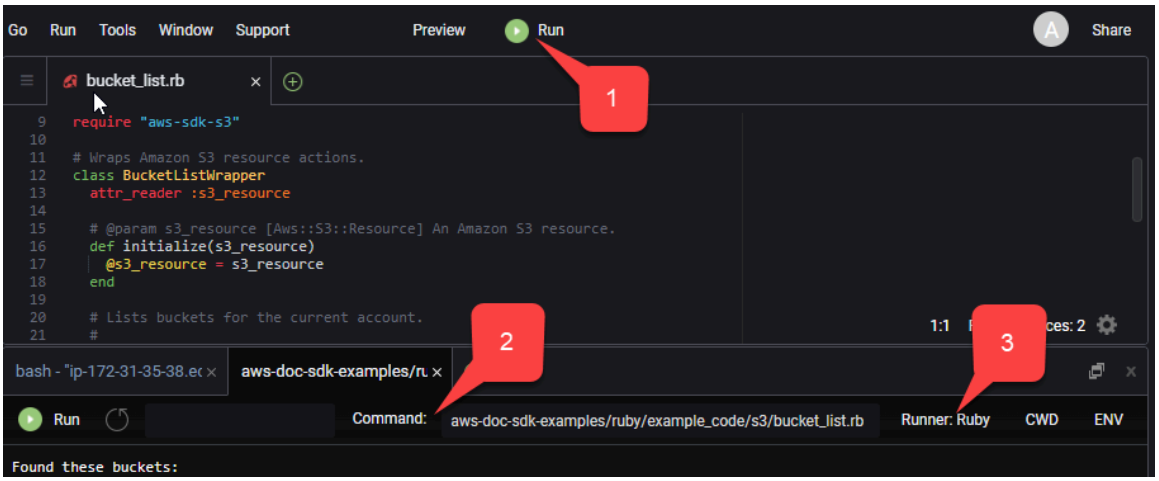
要在 Amazon Cloud9 开发环境中运行代码，请选择顶部菜单栏中的 “运行” 按钮。 Amazon Cloud9 自动检测`.php`文件扩展名并使用 PHP（内置 Web 服务器）运行程序来运行代码。但是，在此示例中，我们实际上需要 PHP（**`cli`**）选项。有关在 Amazon Cloud9中运行代码的更多信息，请参阅 Amazon Cloud9 用户指南中的运行代码。

在下面的屏幕截图中，请注意以下基本区域：

- 1：运行。运行按钮位于顶部菜单栏中。这会为结果打开新选项卡。

> **Note**
>
> 还可以手动创建新的运行配置。在菜单栏上依次选择运行、运行配置和新建运行配置。

- 2：命令。 Amazon Cloud9 使用您运行的文件的路径和文件名填充命令文本框。如果代码需要传入任何命令行参数，则可以将这些参数添加到命令行中，方法与通过终端窗口运行代码时相同。
- 3：跑步者。 Amazon Cloud9 检测到您的文件扩展名为，`.php`然后选择 PHP（内置 Web 服务器）运行程序来运行您的代码。选择 PHP (**`cli`**) 以改为运行此示例。



运行代码生成的任何输出都显示在选项卡中。

在适用于 PHP 的 Amazon SDK 版本 3 中配置服务客户端

为了以编程方式访问 Amazon Web Services 服务，适用于 PHP 的 Amazon SDK 版本 3 为每项 Amazon Web Services 服务使用一个客户端对象。例如，如果您的应用程序需要访问 Amazon EC2，则创建一个 Amazon EC2 客户端对象（Ec2Client 类的一个实例）来与该服务交互。然后，您可以使用服务客户端向该 Amazon Web Services 服务 发出请求。

配置 SDK 行为的方法有很多，但归根结底，一切都与服务客户端的行为有关。在您的代码创建使用配置的服务客户端之前，任何配置都不会生效。

您提供的配置示例包括：

- 当您调用服务时代码向 Amazon 进行身份验证的方式
- 您希望服务客户端使用的 Amazon Web Services 区域
- 服务调用的重试和超时设置
- HTTP 代理配置

请参阅 Amazon SDK 和工具参考指南，以了解许多 Amazon SDK 中常见的设置、功能和其他基础概念。

在外部配置 适用于 PHP 的 Amazon SDK 版本 3 的服务客户端

许多配置设置可以通过代码以外的方式来处理。大多数配置设置可以设置为环境变量，也可以在单独的共享 Amazon `config`文件中设置。 Amazon 共享 `config` 文件可以维护多组独立的设置（称为配置文件），以便为不同的环境或测试提供不同的配置。有关 Amazon 共享`credentials`文件`config`和文件的完整讨论，请参阅Amazon SDKs 和工具参考指南。

大多数环境变量和共享`config`文件设置都是标准化的，并且是跨 Amazon SDKs 工具共享的，以支持不同编程语言和应用程序之间的一致功能。

要查看 SDK 可以从环境变量或配置文件中解析的所有设置，请参阅Amazon SDKs 和工具参考指南中的设置参考。

用于客户端配置的配置提供程序链

SDK 会检查几个位置（或来源）以查找配置值。

1. 在代码中或服务客户端本身设置的任何显式设置均优先于其他任何设置。
2. 环境变量
   - 有关设置环境变量的详细信息，请参阅《工具参考指南》Amazon SDKs 和《工具参考指南》中的环境变量。
   - 请注意，您可以在不同作用域层级为 shell 配置环境变量：系统级、用户级，以及特定终端会话级。
3. 共享 `config` 文件和 `credentials` 文件
   - 有关设置这些文件的详细信息，请参阅《工具参考指南》config和《工具参考指南》中的 “共享 Amazon SDKs 和*credentials*文件”。
4. 最后使用 SDK 源代码本身提供的任何默认值。
   - 某些属性（例如“Region”）没有默认值。您必须在代码、环境设置或共享 `config` 文件中明确指定这些属性。如果 SDK 无法解析所需的配置，API 请求在运行时会失败。

除了这个通用配置链外， 适用于 PHP 的 Amazon SDK 版本 3 还使用专门的提供者链，包括凭证提供程序链和Amazon Web Services 区域 解析链。这些专门的链会添加其他提供程序，这些提供程序会考虑 SDK 的运行环境。例如，在容器或 EC2 实例中。

创建使用外部设置配置的服务客户端

您需要在应用程序中创建一个服务客户端才能与 Amazon Web Services 服务通信。服务客户是您与之建立联系的重要 Amazon Web Services 服务纽带，可以处理所有复杂的沟通细节，因此您不必担心。它们会自动处理安全性、错误处理和重试等重要任务，让您可以专注于构建应用程序，而不是处理复杂的技术问题。

使用不带参数的构造函数构造服务客户端

如果您需要的所有配置设置都来自外部来源，则可以使用空构造函数来创建服务客户端：

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

use Aws\S3\S3Client;

$s3 = new S3Client([]);
```

前面的代码段创建了一个 `S3Client` 实例。在创建过程中，SDK 会按照配置提供程序链依次查找所需的设置。SDK 找到设置值后，就会使用该值。

创建过程中还将使用默认提供 Amazon Web Services 区域 商链和默认凭证提供者链。在链的某个地方，SDK 必须解析要使用的 Amazon Web Services 区域 ，并找到使其能够为签名请求检索凭证的设置。如果 SDK 无法找到这些值，则客户端创建失败。

适用于 PHP 的 Amazon SDK 版本 3 环境变量

除了大多数人支持的交叉 SDK 设置外 Amazon SDKs， 适用于 PHP 的 Amazon SDK 版本 3 还可使用以下环境变量：

AWS_SDK_加载_非默认_配置

设置后，此环境变量会指示 SDK 除 Amazon `config`文件之外还从`credentials`文件 (`~/.aws/config`~/.aws/credential) 加载凭证。

AWS_SDK_UA_APP_ID

设置自定义应用程序标识符，以包含在 SDK 发出的请求的 User-Agent 标头中。

AWS_SUPPRESS_PHP_DEPRECATION_WARNING

当设置为 true 时，会抑制 SDK 可能生成的 PHP 弃用警告。

在 适用于 PHP 的 Amazon SDK 版本 3 的代码中配置服务客户端

除了（或作为替代方案）通过外部方式配置服务客户端之外，您还可以在代码中以编程方式对其进行配置。

通过在代码中配置服务客户端，您可以精细地控制许多可用的选项。可以通过外部方式设置的大多数配置也可以在代码中进行设置。

代码中的基本配置

您可以向客户端的构造函数传递选项的关联数组，从而在代码中创建和配置服务客户端。在以下示例中，关联数组仅包含客户端使用的“区域”选项：

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

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

//Create an S3Client
$s3 = new S3Client([
```

```
    'region' => 'eu-south-2'
]);
```

有关可选“版本”参数的信息，请参阅配置选项主题。

请注意，我们并未向客户端显式提供凭证。这是因为 SDK 使用默认凭证提供程序链来查找凭证信息。

在 适用于 PHP 的 Amazon SDK 版本 3 的客户端构造器选项 中详细介绍了所有通用的客户端配置选项。创建的客户端不同，提供的选项数组也不同。每个客户端的 API 文档中介绍了这些自定义客户端配置选项。

使用 Sdk 类

Aws\Sdk 类可作为客户端工厂，用于管理多个客户端的共享配置选项。许多可提供给特定客户端构造函数的选项也可提供给 Aws\Sdk 类。这些选项会应用于每个客户端构造函数。

导入

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

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

示例代码

```
// 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();
```

所有客户端均可共享的选项置于根级别的键值对中。服务特定的配置数据可以在关联数组中提供，其密钥与服务的命名空间相同（例如，“S3”、“DynamoDb” 等）。

```
$sdk = new Aws\Sdk([
```

```
    'region'   => 'us-west-2',
    'DynamoDb' => [
        'region' => 'eu-central-1'
    ]
]);

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

特定于服务的配置值结合了特定于服务的值和根级值（即特定于服务的值浅合并到根级值）。

> **Note**
>
> 如果您在应用程序中使用多个客户端实例，强烈建议您使用 Sdk 类来创建客户端。Sdk 类会自动针对每个开发工具包客户端使用同一 HTTP 客户端，允许不同服务的开发工具包客户端执行非阻塞 HTTP 请求。如果开发工具包客户端未使用同一 HTTP 客户端，则开发工具包客户端发送的 HTTP 请求可能会阻塞服务之间的 Promise 协调。

适用于 PHP 的 Amazon SDK 版本 3 的客户端构造器选项

客户端构造函数选项可在客户端构造函数中提供，也可以提供给 `Aws\Sdk` 类。为特定类型客户端提供的选项数组会因您创建的客户端而异。每个客户端的 API 文档中都介绍了这些自定义客户端配置选项。

如果您没有明确提供客户端所需的客户端构造器选项，则 SDK for PHP 会从环境变量或 Amazon 配置文件中查找值。所有客户端都需要一个凭证提供程序值和一个 Amazon Web Services 区域 值，因此您必须将这些值设置为构造函数选项或在外部设置。

默认情况下，所检查的配置文件是位于主目录中的 `.aws/config`，通常为 `~/.aws/config`。但是，您可以使用环境变量 `AWS_CONFIG_FILE` 来设置默认配置文件所在的位置。例如，如果您使用 `open_basedir` 来限制对特定目录的文件访问，这可能特别有用。

有关共享 Amazon `config`文件和`credentials`文件的位置和格式的更多信息，请参阅《Amazon SDKs 和工具参考指南》中的配置。

有关可以在配置文件中设置或作为环境变量设置的所有全局 Amazon 配置设置的详细信息，请参阅《和工具参考指南》中的配置Amazon SDKs 和身份验证设置参考。

配置选项

- api_provider
- 凭证
- debug
- stats
- 端点
- endpoint_provider
- endpoint_discovery
- handler
- http
- http_handler
- 配置文件
- region
- 重试
- scheme
- 服务
- signature_provider
- signature_version
- ua_append
- use_aws_shared_config_files
- 验证
- 版本

以下示例展示了如何将选项传入 Amazon S3 客户端构造函数。

```
use Aws\S3\S3Client;

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

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

有关构造客户端的更多信息，请参阅the section called “创建基本服务客户端”。

api_provider

Type

```
callable
```

可调用的 PHP，接受类型、服务和版本参数，并返回一组相应的配置数据。类型值可为 `api`、`waiter` 或 `paginator` 之一。

默认情况下，开发工具包使用可从开发工具包的 `Aws\Api\FileSystemApiProvider` 文件夹加载 API 文件的 `src/data` 实例。

凭证

Type

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

传递 `Aws\Credentials\CredentialsInterface` 对象以使用特定凭证实例。以下内容指定应使用 IAM Identity Center 凭证提供程序。此提供程序也称为 SSO 凭证提供程序。

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

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

如果使用命名配置文件，请在上一个示例中用配置文件名称来替换“`default`”。要了解有关设置命名配置文件的更多信息，请参阅config和工具参考指南中的共享Amazon SDKs 和*credentials*文件。

如果没有指定待使用的凭证提供程序，而是依赖凭证提供程序链，则由于身份验证失败而产生的错误消息通常是通用的。其由正在检查有效凭证的源列表中的最后一个提供程序生成，该提供程序可能不是您尝试使用的提供程序。当您指定要使用哪个凭证提供程序时，任何由此产生的错误消息都更加有用和相关，因为其仅来自该提供程序。要详细了解已检查凭证的来源链，请参阅Amazon SDKs 和工具参考指南中的凭证提供者链。

传递 `false` 以使用空凭证并且不对请求签名。

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

传递可调用的凭证提供程序函数，以使用函数创建凭证。

```
use Aws\Credentials\CredentialProvider;

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

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

将缓存的凭证传递到 `Aws\CacheInterface` 的实例，以缓存跨多个流程的默认提供程序链返回的值。

```
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
]);
```

您可以在适用于 PHP 的 Amazon SDK 版本 3 的凭据指南中找到有关向客户提供凭证的更多信息。

> **Note**
>
> 加载凭证并在使用时延时验证。

debug

Type

```
bool|array
```

输出有关每次传输的调试信息。调试信息包含在准备和通过线路发送事务时事务的每个状态变更的信息。调试输出中还包括有关客户端所使用的特定 HTTP 处理程序的信息（例如，调试 cURL 输出）。

设置为 `true` 以便在发送请求时显示调试信息。

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

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

您也可以提供包含以下键的关联数组。

logfn（可调用）

通过日志消息调用的函数。默认情况下，会使用 PHP 的 `echo` 函数。

stream_size (int)

当流的大小大于此数字时，将不记录流数据。设置为 `0`，以不记录任何流数据。

scrub_auth (bool)

设置`false`为可禁用从记录的消息中清理身份验证数据（这意味着您的 Amazon 访问密钥 ID 和签名将传递到）。`logfn`

http (bool)

设置为 `false`，以禁用低级 HTTP 处理程序的“调试”功能（如完整 cURL 输出）。

auth_headers（数组）

设置为您要替换的标头映射到要将标头替换为的值的键/值映射。除非将 `scrub_auth` 设置为 `true`，否则不会使用这些值。

auth_strings（数组）

设置为要映射到替代项的正则表达式的键/值映射。如果将 `scrub_auth` 设置为 `true`，身份验证数据清除器将使用这些值。

```
$s3 = new Aws\S3\S3Client([
    'region'  => 'us-west-2',
    'debug'   => [
        'logfn'        => function ($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**
>
> 此选项还输出由 `http` 调试选项生成的底层 HTTP 处理程序信息。调试输出对于在 适用于 PHP 的 Amazon SDK中诊断问题极其有用。在开发工具包中打开问题时，请为隔离的故障案例提供调试输出。

stats

Type

`bool|array`

将传输统计数据绑定到开发工具包操作返回的错误和结果。

设置为 `true` 以收集有关所发送请求的传输统计数据。

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

// Perform an operation
$result = $s3->listBuckets();
// Inspect the stats
$stats = $result['@metadata']['transferStats'];
```

您也可以提供包含以下键的关联数组。

retries (bool)

设置为 `false` 以对尝试的重试禁用报告。默认情况下，会收集并返回重试统计数据。

http (bool)

设置`true`为可以从较低级别的 HTTP 适配器收集统计信息（例如，中返回的值 GuzzleHttpTransferStats）。HTTP 处理程序必须支持 __on_transfer_stats 选项，此设置才会生效。HTTP 统计数据以关联数组的索引数组的形式返回；每个关联数组都包含客户端的 HTTP 处理程序为请求返回的传输统计数据。默认情况下禁用。

如果重试了某个请求，则会返回每次请求的传输统计数据，`$result['@metadata']['transferStats']['http'][0]` 包含第一次请求的统计数据，`$result['@metadata']['transferStats']['http'][1]` 包含第二次请求的统计数据，以此类推。

timer (bool)

设置为 `true` 以启用命令计时器，可报告某一操作所用的总挂钟时间（以秒为单位）。默认情况下禁用。

```
$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'];
```

端点

Type

`string`

Web 服务的完整 URI。这是 AWS Elemental MediaConvert 等服务所必需的，这些服务使用账户特定的端点。对于这些服务，请使用 `describeEndpoints` 方法请求此端点。

仅在连接到自定义端点（例如，Amazon S3 的本地版本或本地 Amazon DynamoDB）时需要。

以下是连接到本地 Amazon DynamoDB 的示例：

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

有关可用Amazon 区域和终端节点的列表，请参阅 Amazon 区域和终端节点。

endpoint_provider

Type

`Aws\EndpointV2\EndpointProviderV2|callable`

EndpointProviderV2 或 PHP 的可调用实例，它接受选项的哈希值，包括 “服务” 和 “区域” 密钥。它会返回 NULL 或端点数据的哈希值，其中“端点”键是必需的。

以下示例演示如何创建最简单的端点提供程序。

```
$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
```

端点发现识别并连接到支持端点发现的服务 API 的正确端点。对于支持但不需要端点发现的服务，请在客户端创建期间启用 `endpoint_discovery`。如果服务不支持端点发现，则忽略此配置。

`Aws\EndpointDiscovery\ConfigurationInterface`

一个可选配置提供程序，可针对服务指定的操作实现自动连接到服务 API 的相应端点。

`Aws\EndpointDiscovery\Configuration` 对象接受两个选项，包括布尔值“enabled”（指示是否启用端点发现）和整数“cache_limit”（指示端点缓存中的最大键数量）。

对于创建的每个客户端，传递一个 `Aws\EndpointDiscovery\Configuration` 对象以使用端点发现的特定配置。

```
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,
```

```
]);
```

传递 Aws\CacheInterface 的实例以缓存端点发现在多个流程中返回的值。

```
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),
]);
```

将数组传递到端点发现。

```
use Aws\S3\S3Client;

$s3 = new S3Client([
    'region'      => 'us-west-2',
    'endpoint_discovery' => [
        'enabled' => true,
        'cache_limit' => 1000
    ],
]);
```

handler

Type

```
callable
```

一个处理程序，接受命令对象和请求对象，并返回已完成并显示 GuzzleHttp\Promise\PromiseInterface 对象或者已拒绝并显示 Aws\ResultInterface 的承诺 (Aws\Exception\AwsException)。处理程序不接受下一个处理程序，因为它是终端，预期应执行命令。如果未提供任何处理程序，则使用默认的 Guzzle 处理程序。

您可以使用 Aws\MockHandler 返回模拟结果或引发模拟异常。您将结果或异常入队，然后将按照 FIFO 顺序 MockHandler 将它们出队。

```
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
```

设置为一组 HTTP 选项，这些选项将应用于开发工具包所创建的 HTTP 请求和传输。

开发工具包支持以下配置选项：

cert

Type

```
string|array
```

指定 PEM 格式的客户端证书。

- 设置为一个字符串，此字符串表示仅到证书文件的路径。

```
use Aws\S3\S3Client;

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

- 设置为包含路径和密码的数组。

```
use Aws\S3\S3Client;

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

connect_timeout

一个浮点数，描述在尝试连接服务器时要等待的秒数。使用 `0` 无限期等待（默认行为）。

```
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
```

指示基础 HTTP 处理程序输出调试信息。不同 HTTP 处理程序提供的调试信息会有所不同。

- 传递 `true` 以将调试输出写入 STDOUT。
- 传递 `resource` 返回的 `fopen`，以将调试输出写入特定的 PHP 流资源。

decode_content

Type

```
bool
```

指示基础 HTTP 处理程序扩大压缩响应的正文。如果不启用，压缩的响应正文可能用 `GuzzleHttp\Psr7\InflateStream` 来扩大。

> **Note**
>
> 默认情况下，在开发工具包的默认 HTTP 处理程序中会启用内容解码。为了实现向后兼容性，此默认设置无法更改。如果在 Amazon S3 中存储压缩文件，我们建议您在 S3 客户端级别禁用内容解码。

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

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

$result = $client->getObject([
    'Bucket' => 'amzn-s3-demo-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`

在发送请求之前要延迟的毫秒数。这通常用于重试请求之前的延迟。

expect

Type

`bool|string`

此选项将传递至基础 HTTP 处理程序。默认情况下，将在请求的正文超过 1 MB 时设置“Expect: 100-Continue”标头。`true` 或 `false` 在所有请求上启用或禁用该标头。如果使用整数，则仅其正文超出此设置的请求将使用该标头。用作整数时，如果正文大小未知，则将发送 Expect 标头。

> **Warning**
>
> 禁用 Expect 标头可防止服务返回身份验证或其他错误。应小心地配置此选项。

进度

Type

`callable`

定义要在有了传输进度时调用的函数。该函数接受以下参数：

1. 预计要下载的总字节数。
2. 到目前为止下载的字节数。
3. 预计要上传的字节数。
4. 到目前为止上传的字节数。

```
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 = $client->getObject([
    'Bucket' => 'amzn-s3-demo-bucket',
    'Key'    => 'large.mov',
    '@http' => [
        'progress' => function ($expectedDl, $dl, $expectedUl, $ul) {
            printf(
                "%s of %s downloaded, %s of %s uploaded.\n",
                $expectedDl,
                $dl,
                $expectedUl,
                $ul
            );
        }
    ]
]);
```

proxy

Type

```
string|array
```

您可以使用proxy选项通过代理连接到 Amazon 服务。

- 提供一个字符串值以连接到所有类型的代理 URIs。该代理字符串值可以包含方案、用户名和密码。例如 "http://username:password@192.168.16.1:10"。
- 提供代理设置的关联数组，其中键是 URI 的方案，值是给定 URI 的代理（即，您可以为“http”和“https”端点提供不同的代理）。

```
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',
        ]
    ]
]);
```

您可以使用 HTTP_PROXY 环境变量来配置“http”协议特定的代理，使用 HTTPS_PROXY 环境变量来配置“https”特定的代理。

sink

Type

```
resource|string|Psr\Http\Message\StreamInterface
```

`sink` 选项控制要将操作的响应数据下载到的位置。

- 提供 `resource` 返回的 `fopen`，以将响应正文下载到 PHP 流。
- 以 `string` 值的形式提供文件在磁盘上的路径，以将响应正文下载到磁盘上的特定文件。
- 提供 `Psr\Http\Message\StreamInterface`，以将响应正文下载到特定的 PSR 流对象。

> **Note**
>
> 默认情况下，开发工具包会将响应正文下载到 PHP 临时流。这意味着，数据会一直保留在内存中，直到正文大小达到 2 MB，此时数据将被写入磁盘上的临时文件中。

synchronous

Type

bool

synchronous 选项会告知基础 HTTP 处理程序，您打算阻止结果。

流

Type

bool

设置为 true，以告知基础 HTTP 处理程序，您要从 Web 服务流式传输响应的响应正文，而不是预先进行下载。例如，Amazon S3 流包装类中依赖此选项来确保对数据进行流式传输。

timeout

Type

float

一个浮点数，描述请求超时（以秒为单位）。使用 0 无限期等待 (默认行为)。

```
use Aws\DynamoDb\DynamoDbClient;

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

确认

Type

bool|string

您可以使用verifyhttp选项自定义 SDK 的对等 SSL/TLS 证书验证行为。

- 设置true为可启用 SSL/TLS 对等证书验证并使用操作系统提供的默认 CA 捆绑包。
- 设置为 false 以禁用对等证书验证。（这样不安全！）
- 设置为一个字符串，以提供 CA 证书捆绑包的路径，从而使用自定义 CA 捆绑包启用验证。

如果找不到系统的 CA 捆绑包并且收到错误，请向开发工具包提供 CA 捆绑包的路径。如果您不需要特定的 CA 捆绑包，Mozilla 提供了可从此处下载的常用 CA 捆绑包（由 cURL 的维护者维护）。一旦您的磁盘中具有 CA 捆绑包，则可将 openssl.cafile PHP.ini 设置为指向该文件的路径，从而忽略 verify 请求选项。您可以在 cURL 网站上了解有关 SSL 证书的更多详细信息。

```
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
```

http_handler 选项用于将开发工具包与其他 HTTP 客户端集成。http_handler 选项是一个函数，接受 Psr\Http\Message\RequestInterface 对象和一组应用于命令的 http 选项，并返回已完成并显示 GuzzleHttp\Promise\PromiseInterface 对象或已拒绝并显示以下一组异常数据的 Psr\Http\Message\ResponseInterface 对象：

- `exception` - (\Exception) 遇到的异常。
- `response` - (Psr\Http\Message\ResponseInterface) 收到的响应（如果有）。
- `connection_error` - (bool) 设置为 `true` 以将错误标记为连接错误。将此值设置为 `true` 还允许开发工具包自动重试该操作（如果需要）。

开发工具包通过用 `http_handler` 对象包装所提供的 `handler` 来将给定的 `http_handler` 转换为正常 `Aws\WrappedHttpHandler` 选项。

默认情况下，开发工具包使用 Guzzle 作为其 HTTP 处理程序。您可以在这里提供不同的 HTTP 处理程序，或者为 Guzzle 客户端提供您自己的自定义选项。

设置 TLS 版本

一种使用情形是使用 Curl 来设置 Guzzle 使用的 TLS 版本，前提是您的环境中已安装了 Curl。请注意 Curl 对于所支持的 TLS 版本的版本约束。默认情况下会使用最新版本。如果明确设置了 TLS 版本，但是远程服务器不支持此版本，则会产生错误，而不是使用更低的 TLS 版本。

您可以通过将 debug 客户端选项设置为 true 并检查 SSL 连接输出，确定用于给定客户端操作的 TLS 版本。此行看起来类似于：`SSL connection using TLSv1.2`

设置 Guzzle 6 使用 TLS 1.2 的示例：

```
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**
>
> `http_handler` 选项会取代所提供的任何 `handler` 选项。

配置文件

Type

`string`

“profile” 选项指定从主目录中的凭据文件创建 Amazon 凭据时要使用的配置文件（通常`~/.aws/credentials`）。此设置将覆盖 `AWS_PROFILE` 环境变量。

> **Note**
>
> 指定 “配置文件” 选项时，将忽略该`"credentials"`选项，并忽略 Amazon 配置文件中与凭据相关的设置（通常`~/.aws/config`）。

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

有关配置凭据和.ini 文件格式的更多信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 的凭据。

region

Type

`string`

必需

true

Amazon 要连接的区域。有关可用区域的列表，请参阅 Amazon 区域和端点。

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

重试

Type

```
int|array|Aws\CacheInterface|Aws\Retry\ConfigurationInterface|callable
```

默认

```
int(3)
```

配置重试模式和对客户端允许的最大重试次数。传递 `0` 以禁用重试。

三种重试模式是：

- `legacy`：默认的旧版重试实现
- `standard`：添加了重试配额系统，以防止不太可能成功的重试
- `adaptive`：构建在标准模式上，添加了客户端速率限制器。请注意，此模式被认为是实验性的。

重试的配置由每个请求使用的模式和最大尝试次次数组成。配置可以按以下优先顺序在几个不同的位置进行设置。

优先顺序

重试配置的优先级顺序如下所示（1 覆盖 2-3，等等）：

1. 客户端配置选项
2. 环境变量
3. Amazon 共享配置文件

环境变量

- `AWS_RETRY_MODE` – 设置为 `legacy`、`standard` 或 `adaptive`。
- `AWS_MAX_ATTEMPTS` – 设置为整数值，表示每个请求的最大尝试次数

共享的配置文件密钥

- retry_mode – 设置为 legacy、standard 或 adaptive。
- max_attempts – 设置为整数值，表示每个请求的最大尝试次数

客户端配置

下面的示例将对 Amazon DynamoDB 客户端禁用重试。

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

以下示例传入一个整数，这将默认为 legacy 模式并使用传入整数作为重试次数

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

Aws\Retry\Configuration 对象接受两个参数，即重试模式

和表示每个请求的最大尝试次数的整数。此示例传递一个

用于重试配置的 Aws\Retry\Configuration 对象。

```
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,
]);
```

此示例传入一个用于重试配置的数组。

```
use Aws\S3\S3Client;

$s3 = new S3Client([
    'region' => 'us-west-2',
    'retries' => [
        'mode' => 'standard',
        'max_attempts' => 7
    ],
]);
```

此示例传递 Aws\CacheInterface 的实例以缓存默认重试配置提供程序返回的值。

```
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
```

默认

```
string(5) "https"
```

连接时要使用的 URI 方案。默认情况下，开发工具包会使用“https”端点（即使用 SSL/TLS 连接）。您可以通过将 scheme 设置为“http”，尝试通过未加密的“http”端点来连接服务。

```
$s3 = new Aws\S3\S3Client([
    'version' => '2006-03-01',
```

```
    'region'  => 'us-west-2',
    'scheme'  => 'http'
]);
```

有关端点列表以及服务是否支持 http 方案，请参阅 Amazon 区域和端点。

服务

Type

`string`

必需

`true`

要使用的服务的名称。在使用开发工具包提供的客户端时，默认情况下会提供此值（即 `Aws\S3\S3Client`）。在测试开发工具包中尚未发布但磁盘中已提供的服务时，此选项很有用。

signature_provider

Type

`callable`

一种可调用对象，它接受签名版本名称（例如v4）、服务名称和 Amazon 区域并返回Aws\Signature\SignatureInterface对象，NULL或者提供者能够为给定参数创建签名者。此提供程序用于创建供客户端使用的签署人。

开发工具包在 `Aws\Signature\SignatureProvider` 类中提供了可用于创建自定义签名提供程序的不同函数。

signature_version

Type

`string`

一个字符串，表示要用于服务的自定义签名版本（例如 v4 等）。如果需要，每个操作签名版本均可能覆盖此请求的签名版本。

以下示例展示了如何配置要使用签名版本 4 的 Amazon S3 客户端：

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

> **Note**
>
> 客户端使用的 `signature_provider` 必须能够创建您提供的 `signature_version` 选项。开发工具包使用的默认 `signature_provider` 可以创建“v4”和“anonymous”签名版本的签名对象。

ua_append

Type

`string|string[]`

默认

`[]`

添加到传递给 HTTP 处理程序的用户代理字符串的字符串或字符串数组。

use_aws_shared_config_files

Type

`bool|array`

默认

`bool(true)`

设置为 false 可禁用在 '~/ 中检查共享配置文件。 aws/config’ and ‘~/.aws/credentials'。这将覆盖 AWS_CONFIG_FILE 环境变量。

验证

Type

```
bool|array
```

默认

```
bool(true)
```

设置为 `false` 以禁用客户端参数验证。您可能会发现，关闭验证会略微提高性能，但区别可以忽略不计。

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

设置为关联的验证选项数组，以启用特定的验证约束：

- `required` - 验证必需的参数是否存在（默认启用）。
- `min` - 验证值的最小长度（默认启用）。
- `max` - 验证值的最大长度。
- `pattern` - 验证值是否与正则表达式匹配。

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

版本

Type

```
string
```

必需

false

此选项指定要使用的 Web 服务的版本（例如，`2006-03-01`）。

从 SDK 的版本 3.277.10 开始，“版本”选项不再是必需的。如果没有指定“版本”选项，则 SDK 使用服务客户端的最新版本。

在创建服务客户端时，有两种情况需要“版本”参数。

- 您使用的是 3.277.10 之前的 PHP SDK 版本。
- 您使用的是 3.277.10 或更高版本，并且想要使用不是“最新”版本的服务客户端版本。

例如，以下代码段使用了 SDK 的版本 3.279.7，但不是 `Ec2Client` 的最新版本。

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

指定版本约束可确保对服务所做的重大更改不会影响您的代码。

在每个客户端的 API 文档页面上均提供了可用 API 版本的列表。如果您无法加载特定 API 版本，则可能需要更新开发工具包副本。

Amazon Web Services 区域 为 适用于 PHP 的 Amazon SDK 版本 3 进行设置

SDK 客户端会 Amazon Web Services 服务 连接到您在创建客户端时指定的特定 Amazon Web Services 区域 内容。此配置允许您的应用程序与该地理区域中的 Amazon 资源进行交互。当您在未明确设置区域的情况下创建服务客户端时，SDK 将使用外部配置中的默认区域。

区域解析链

适用于 PHP 的 Amazon SDK 版本 3 使用以下顺序来确定服务客户端使用哪个区域：

1. 代码中提供的区域 — 如果您在客户端构造函数选项中显式设置了区域，则该区域优先于所有其他来源。

```
$s3Client = new Aws\S3\S3Client([
    'region' => 'us-west-2'
]);
```

2. 环境变量 — 如果代码中未提供区域，SDK 将按顺序检查以下环境变量：
   - AWS_REGION
   - AWS_DEFAULT_REGION

```
# Example of setting Region through environment variables.
export AWS_REGION=us-east-1
```

3. Amazon 配置文件-如果未设置区域环境变量，SDK 将检查 Amazon 配置文件：
   a. SDK 在 ~/.aws/config（或 AWS_CONFIG_FILE 环境变量指定的位置）中查找
   b. SDK 在 AWS_PROFILE 环境变量指定的配置文件中检查区域设置
   c. 如果未指定 AWS_PROFILE，则 SDK 使用“default”配置文件

   例如，假设我们有以下配置文件设置：

```
# Example ~/.aws/config file.
[default]
region = eu-west-1

[profile production]
region = eu-central-1
```

   如果将 AWS_PROFILE 环境变量的值设置为“production”，则客户端使用 eu-central-1 Region。如果不存在 AWS_PROFILE 环境变量，则客户端使用 eu-west-1 区域。

4. 如果 SDK 在上述任何来源中均未找到区域值，则会引发异常，因为区域值是服务客户端的一项必需设置。

最佳实践

在 适用于 PHP 的 Amazon SDK 版本 3 中使用区域时，请考虑以下最佳实践：

在生产代码中显式设置区域

对于生产应用程序，我们建议在代码中显式设置区域，而不是依赖环境变量或 `config`。这使得您的代码更具可预测性，并且减少了对外部配置的依赖。

在开发和测试环境中使用环境变量

对于开发和测试环境，使用环境变量可以提高灵活性，而无需更改代码。

在多个环境中使用配置文件

如果您的应用程序需要在多个 Amazon 环境中运行，请考虑在文件中使用不同的配置 Amazon `config`文件并根据需要在它们之间切换。

使用 适用于 PHP 的 Amazon SDK 版本 3 凭证提供商

有关可用凭证机制的参考信息 Amazon SDKs，请参阅和工具参考指南中的凭证Amazon SDKs 和访问权限。

> **Important**
>
> 出于安全考虑，我们强烈建议您不要使用 root 账户进行 Amazon 访问。请务必参阅 IAM 用户指南中的 IAM 中的安全最佳实践，了解最新安全建议。

在 适用于 PHP 的 Amazon SDK 版本 3 中，凭证提供者的角色是为 SDK 的 Amazon Web Services 服务 客户端获取和提供凭证。SDK 使用其获取的凭证对每个请求进行加密签名，从而向服务进行身份验证。凭证通常包含访问密钥（一个访问密钥 ID 和一个秘密访问密钥）。

当您使用临时证书时，例如设置 IAM Identity Center 身份验证或将运行时配置为担任 IAM 角色时，会话令牌会添加到访问密钥中，从而提供对 Amazon 资源的限时访问权限。

适用于 PHP 的 Amazon SDK 版本 3 中的凭证提供者是什么？

凭证提供程序是一个函数，该函数返回一个用 `GuzzleHttp\Promise\PromiseInterface` 实例执行或因 `Aws\Credentials\CredentialsInterface` 而被拒绝的 `Aws\Exception\CredentialsException`。SDK 提供多种凭证提供程序函数的实现方式，或者您可以实现自己的自定义逻辑来创建凭证或优化凭证加载过程。

凭证提供程序将被传入 `credentials` 客户端构造函数选项。凭证提供程序是异步的，因此每次调用 API 操作时都会强制对其进行延迟评估。因此，将凭证提供程序函数传递给开发工具包

客户端构造函数不会立即验证凭证。如果凭证提供程序未返回凭证对象，将因 Aws\Exception\CredentialsException 拒绝 API 操作。

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

// Use the ECS credential provider.
$provider = CredentialProvider::ecsCredentials();
// Be sure to memoize the credentials.
$memoizedProvider = CredentialProvider::memoize($provider);

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

了解 适用于 PHP 的 Amazon SDK 版本 3 中的默认凭证提供者链

默认凭证提供程序链由 SDK 调用的一系列内置凭证提供程序组成。它由 defaultProvider 凭证提供程序函数实现，不带任何参数。找到有效凭证后，搜索即告停止。

按以下 适用于 PHP 的 Amazon SDK 顺序执行凭证提供程序：

- **env 提供程序** - SDK 搜索已设置为环境变量的Amazon 访问密钥。
- **assumeRoleWithWebIdentityCredentialProvider 提供程序** - SDK 搜索 IAM 角色和 Web 身份令牌文件设置。
- 在链的此时刻，SDK 在共享 Amazon config和credentials文件中查找配置。SDK 在“默认”配置文件下查找配置，但如果设置了 AWS_PROFILE 环境变量，SDK 将使用其命名配置文件值。
  - **sso 提供程序** - SDK 在共享 config 文件中查找 IAM Identity Center 配置设置。
  - **login provider**-SDK 在共享config文件中查找 Amazon 控制台登录会话配置设置。
  - **process 提供程序** - SDK 在共享 credentials 文件中查找 credential_process 设置。
  - **ini提供商**-软件开发工具包在共享credentials文件中查找 Amazon 证书或 IAM 角色信息。
  - **process 提供程序** - SDK 在共享 config 文件中查找 credential_process 设置。
  - **ini提供商**-软件开发工具包在共享config文件中查找 Amazon 证书或 IAM 角色信息。

- **ecsCredentials 提供程序** - SDK 查找环境变量 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 或 `AWS_CONTAINER_CREDENTIALS_FULL_URI`，这些变量会提供获取临时凭证所需的信息。
- **instanceProfile 提供程序** - SDK 使用 EC2 实例元数据服务来获取实例配置文件中指定的 IAM 角色。SDK 根据角色信息获取临时凭证。

> **Note**
>
> 系统会自动记住默认提供程序的结果。

你可以在 GitHub 源代码中查看链的代码。

适用于 PHP 的 Amazon SDK 版本 3 中的内置凭证提供商

SDK 提供了多个内置凭证提供程序，您可以单独使用这些提供程序，也可以在自定义凭证提供程序链中结合使用。

当您在创建服务客户端期间指定凭证提供程序时，SDK 会尝试仅使用指定的凭证提供程序来加载凭证。它不使用默认凭证提供程序链。如果您知道服务客户端需要使用 `instanceProfile` 提供程序，可通过在服务客户端构造函数中指定 `instanceProfile` 提供程序来绕过默认链：

```
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',
    'credentials' => $memoizedProvider  // The default credential provider chain is not
 used.
]);
```

> **Important**
>
> 每次执行 API 操作时均会调用凭证提供程序。如果加载凭证是一项代价高昂的任务（例如从磁盘或网络资源加载）或者凭证未由提供程序缓存，请考虑将您的凭证提供程序包装在 Aws

`\Credentials\CredentialProvider::memoize` 函数中。系统会自动记住开发工具包使用的默认凭证提供程序。

主题

- 适用于 PHP 的 SDK 中的 login 提供程序
- 适用于 PHP 的 SDK 中的 assumeRole 提供程序
- 适用于 PHP 的 SDK 中的 sso 提供程序
- 适用于 PHP 的 SDK 中的 defaultProvider 提供程序
- 适用于 PHP 的 SDK 中的 ecsCredentials 提供程序
- 适用于 PHP 的 SDK 中的 env 提供程序
- 适用于 PHP 的 SDK 中的 assumeRoleWithWebIdentityCredentialProvider 提供程序
- 适用于 PHP 的 SDK 中的 ini 提供程序
- 适用于 PHP 的 SDK 中的 process 提供程序
- 适用于 PHP 的 SDK 中的 instanceProfile 提供程序

适用于 PHP 的 SDK 中的 **`login`** 提供程序

`Aws\Credentials\CredentialProvider::login`尝试加载由基于浏览器的登录会话配置的凭据，这些会话由 CLI Amazon 等工具提供便利。身份验证后， Amazon 生成适用于本地 Amazon SDKs 和工具的临时证书。

通过此流程，您可以使用在初始账户设置期间创建的根证书、IAM 用户或身份提供商提供的联合身份进行身份验证，而 Amazon SDK for PHP 会自动为您管理临时证书。这种方法无需在本地存储长期凭证，从而增强了安全性。

运行 `aws login` 命令时，您可以从活动控制台会话中进行选择，也可以通过基于浏览器的身份验证流程登录，这将自动生成临时凭证。 Amazon 适用于 PHP 的 SDK 将使用登录服务自动刷新这些凭证，最长 12 小时。

登录提供商会尝试根据提供的配置文件加载由前面提到的登录会话工作流程生成的访问令牌。如果在调用提供程序时未提供配置文件，它将尝试通过首先检查`AWS_PROFILE`环境变量来解析配置文件，然后再回退到配置文件`default`。代码内配置可以传递给提供商，在那里它会寻找用于刷新凭据的登录服务客户端的`region`值。如果配置数组中未提供区域，则提供程序将尝试通过检查`AWS_REGION`环境变

量来解析区域，然后检查已解析的配置文件中设置的区域值。如果找不到区域，则提供商将返回被拒绝的承诺，其中包含有关如何配置区域的说明。

提供者作为默认链的一部分被调用，可以直接调用。

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

$provider = CredentialProvider::login(<profile_name>, ['region' => <region>]);
// 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',
    'credentials' => $provider
]);
```

默认情况下，如果您要使用的服务客户端上未提供凭据配置，则该提供程序将作为`defaultProvider()`证书链的一部分进行调用。在这种情况下，服务客户端的区域将自动传递给`login()`提供商。同样在这种情况下，在回退到配置文件之前，将通过检查`AWS_PROFILE`环境变量来解析传递给登录提供者的配置文件值`default`。

适用于 PHP 的 SDK 中的 **`assumeRole`** 提供程序

如果您使用 `Aws\Credentials\AssumeRoleCredentialProvider` 通过代入角色创建凭证，则需要按所示方式使用 `'client'` 对象和 `StsClient` 详细信息来提供 `'assume_role_params'` 信息。

> **Note**
>
> 为避免在每个 API 操作中不必要地获取 Amazon STS 凭证，您可以使用该`memoize`函数来处理证书过期时自动刷新凭证的问题。请参阅下面的示例代码。

```
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
]);
```

有关更多信息`'assume_role_params'`，请参阅AssumeRole。

适用于 PHP 的 SDK 中的 **sso** 提供程序

`Aws\Credentials\CredentialProvider::sso` 是单点登录凭证提供程序。该提供商也称为 Amazon IAM Identity Center 凭证提供商。

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

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

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

```
    'credentials' => $credentials
]);
```

如果使用命名配置文件，请在上一个示例中用配置文件名称来替换“default”。要了解有关设置命名配置文件的更多信息，请参阅《工具参考指南》config和《工具参考指南》中的共享Amazon SDKs和*credentials*文件。或者，您可以使用 AWS_PROFILE 环境变量来指定要使用的配置文件设置。

要进一步了解 IAM 身份中心提供商的工作原理，请参阅Amazon SDKs 和工具参考指南中的了解 IAM 身份中心身份验证。

适用于 PHP 的 SDK 中的 **defaultProvider** 提供程序

Aws\Credentials\CredentialProvider::defaultProvider 是默认凭证提供程序，也称为默认凭证提供程序链。如果您在创建客户端时省略 credentials 选项，则会使用此提供程序。例如，如果您创建一个 S3Client（如以下代码段所示），则 SDK 将使用默认提供程序：

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

如果您想向链中的特定凭证提供程序提供参数，也可以在代码中使用 defaultProvider。例如，如果使用 ecsCredentials 提供程序函数，则以下示例提供自定义连接超时和重试设置。

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

$provider = CredentialProvider::defaultProvider([
    'timeout' => '1.5',
    'retries' => 5
]);

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

适用于 PHP 的 SDK 中的 **ecsCredentials** 提供程序

Aws\Credentials\CredentialProvider::ecsCredentials 尝试通过 GET 请求加载凭证，其 URI 由容器中的环境变量 AWS_CONTAINER_CREDENTIALS_RELATIVE_URI 指定。

```
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
]);
```

适用于 PHP 的 SDK 中的 **env** 提供程序

使用环境变量来包含您的凭据可以防止您意外共享您的 Amazon 私有访问密钥。我们建议您不要在任何生产文件中将 Amazon 访问密钥直接添加到客户端。

要对 Amazon Web Services 进行身份验证，SDK 首先检查环境变量中的凭证。开发工具包会使用 `getenv()` 函数来查找 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN` 环境变量。这些凭证称为环境凭证。有关如何获取这些值的说明，请参阅《工具参考指南》Amazon SDKs 和《工具参考指南》中的使用短期凭证进行身份验证。

如果您在上托管应用程序 Amazon Elastic Beanstalk，则可以通过Amazon Elastic Beanstalk 控制台设置`AWS_ACCESS_KEY_IDAWS_SECRET_KEY`、和`AWS_SESSION_TOKEN`环境变量，这样 SDK 就可以自动使用这些凭证。

有关如何设置环境变量的更多信息，请参阅《工具参考指南》Amazon SDKs 和《工具参考指南》中的环境变量支持。另外，有关大多数支持的所有环境变量的列表 Amazon SDKs，请参阅环境变量列表。

您也可以在命令行中设置环境变量，如下所示。

Linux

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   # The access key for your Amazon Web Services ##.
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   # The secret access key for your Amazon Web Services ##.
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
   # The temporary session key for your Amazon Web Services ##.
```

```
   # 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 Amazon SDKs other than PHP.
```

Windows

```
C:\> SET  AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   # The access key for your Amazon Web Services ##.
C:\> SET  AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   # The secret access key for your Amazon Web Services ##.
C:\> SET AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
   # The temporary session key for your Amazon Web Services ##.
   # 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 Amazon SDKs besides PHP.
```

Aws\Credentials\CredentialProvider::env 尝试从环境变量中加载凭证。

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

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

适用于 PHP 的 SDK 中的 **assumeRoleWithWebIdentityCredentialProvider** 提供程序

Aws\Credentials
\CredentialProvider::assumeRoleWithWebIdentityCredentialProvider 尝试通过代入角色来加载凭证。如果存在环境变量 AWS_ROLE_ARN 和 AWS_WEB_IDENTITY_TOKEN_FILE，则提供商将尝试使用磁盘上的令牌（位于在 AWS_WEB_IDENTITY_TOKEN_FILE 中指定的完整路径上），代入在 AWS_ROLE_ARN 上指定的角色。如果使用环境变量，则提供商将尝试从 AWS_ROLE_SESSION_NAME 环境变量设置会话。

如果未设置环境变量，则提供商将使用默认配置文件或者设置为 AWS_PROFILE 的配置文件。默认情况下，提供商从 ~/.aws/credentials 和 ~/.aws/config 读取配置文件，并可读取 filename 配置选项中指定的配置文件。提供商将代入配置文件的 role_arn 中的角色，从

web_identity_token_file 中设置的完整路径读取令牌。如果已在配置文件上设置，则将使用 role_session_name。

提供商作为默认链的一部分调用，并可以直接调用。

```
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
]);
```

默认情况下，该凭证提供者将继承配置的区域，该区域将 StsClient 用于担任该角色。（可选）StsClient 可以提供完整版。凭证应按照提供的任何false方式进行设置 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
```

```
]);
```

适用于 PHP 的 SDK 中的 **ini** 提供程序

Aws\Credentials\CredentialProvider::ini 尝试从共享的 config 和 credentials 文件中加载凭证。默认情况下，SDK 会尝试从位于的共享 Amazon credentials文件中加载 “默认” 配置文件~/.aws/credentials。如果 SDK 找到了AWS_SDK_LOAD_NONDEFAULT_CONFIG环境变量，它还会在位于的共享 Amazon config文件中检查 “默认” 配置文件~/.aws/config。

```
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
]);
```

您可以通过向创建提供程序的函数提供参数来使用自定义配置文件或 .ini 文件位置。

```
$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
]);
```

适用于 PHP 的 SDK 中的 **process** 提供程序

Aws\Credentials\CredentialProvider::process尝试通过执行共享 Amazon 配置文件中配置文件中指定的credential_process值来加载凭证。

默认情况下，SDK 会先尝试从位于的共享 Amazon credentials文件中加载 “默认” 配置文件~/.aws/credentials。如果在共享 credentials 文件中找不到“默认”配置文件，SDK 将在共享 config 文件中查找默认配置文件。下面是共享 credentials 文件的配置示例。

```
[default]
credential_process = /path/to/file/credential_returning_executable.sh --custom-command
 custom_parameter
```

SDK 将通过使用 PHP 的 shell_exec 函数完全调用给定的 credential_process 命令，然后从 stdout 中读取 JSON 数据。credential_process 必须采用以下格式将凭证写入 stdout：

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

SessionToken 和 Expiration 是可选的。如果存在，凭证将被视为临时的。

```
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
]);
```

您可以通过向创建提供程序的函数提供参数来使用自定义配置文件或 .ini 文件位置。

```
$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
]);
```

适用于 PHP 的 SDK 中的 **instanceProfile** 提供程序

Aws\Credentials\CredentialProvider::instanceProfile 尝试为 Amazon EC2 实例配置文件中指定的 IAM 角色加载凭证。

```
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
]);
```

默认情况下，提供商最多重新尝试提取凭证三次。可以使用 retries 选项设置重试次数，如以下代码所示，将该选项设置为 0 可以完全禁用重试。

```
use Aws\Credentials\CredentialProvider;

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

如果环境变量 AWS_METADATA_SERVICE_NUM_ATTEMPTS 可用，则其值优先于前面显示的“重试”选项。

> **Note**
>
> 可以通过将 AWS_EC2_METADATA_DISABLED 环境变量设置为 true 来禁用从 Amazon EC2 实例配置文件进行加载的尝试。

在适用于 PHP 的 SDK 中链接凭证提供程序

可以使用 Aws\Credentials\CredentialProvider::chain() 函数将凭证提供程序链接起来。此函数接受可变数量的参数，每个参数都是凭证提供程序函数。然后，此函数会返回一个由提供的函数构成的新函数，这样便可以一个接一个地调用这些函数，直至其中一个提供程序返回已成功执行的 Promise。

defaultProvider 在失败之前使用此组合来检查多个提供程序。defaultProvider 的源代码演示了 chain 函数的使用。

```
// 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)
        )
    );
}
```

创建自定义凭证提供程序以与适用于 PHP 的 SDK 结合使用

凭证提供程序只是在调用时返回承诺 (GuzzleHttp\Promise\PromiseInterface) 的函数，该承诺用 Aws\Credentials\CredentialsInterface 对象执行或因 Aws\Exception\CredentialsException 而被拒绝。

创建提供程序的最佳实践是创建一个函数，通过调用该函数来创建实际的凭证提供程序。例如，此处是 env 提供程序的源代码（为了举例，略微进行了修改）。请注意，它是可返回实际提供程序函数的函数。这样，您便可以轻松地构建凭证提供程序并将其作为值来进行传递。

```
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 && $secret) {
            return Create::promise_for(
                new Credentials($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));
    };
}
```

适用于 PHP 的 SDK 中的凭证记忆

有时，您可能需要创建能记住之前返回值的凭证提供程序。当加载凭证是一项代价高昂的操作时或在使用 Aws\Sdk 类跨多个客户端共享凭证提供程序时，这有助于改进性能。您可以通过将凭证提供程序函数包装在 memoize 函数中来向凭证提供程序中添加记忆功能。

```
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());
```

当记住的凭证过期时，记忆包装器将调用可尝试刷新凭证的包装器提供程序。

使用 适用于 PHP 的 Amazon SDK 版本 3 假设一个 IAM 角色

将 IAM 角色用于 Amazon EC2 实例可变凭证

如果您在 Amazon EC2 实例上运行应用程序，则提供用于调用的证书的首选方法 Amazon 是使用 I AM 角色获取临时安全证书。

使用 IAM 角色时，无需担心应用程序的凭证管理。它们允许实例通过从 Amazon EC2 实例的元数据服务器中检索临时凭证来“代入”角色。

这些临时凭证通常称为实例配置文件凭证，允许访问该角色的策略允许的操作和资源。Amazon EC2 将处理为 IAM 服务对实例进行安全身份验证以代入角色的所有调查工作，并定期刷新检索到的角色凭证。这样，您几乎无需承担任何工作，便可保证应用程序的安全。有关接受临时安全凭证的服务列表，请参阅 IAM 用户指南中的使用 IAM 的Amazon 服务。

> **Note**
>
> 要避免每次都击中元数据服务，可将 `Aws\CacheInterface` 的实例作为 `'credentials'` 选项传递给客户端构造函数。这样，开发工具包便可改用缓存的实例配置文件凭证。有关详细信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 的配置。

有关使用开发 Amazon EC2 应用程序的更多信息 SDKs，请参阅Amazon SDKs 和工具参考指南中的为 Amazon EC2 实例使用 IAM 角色。

创建 IAM 角色并将其分配到 Amazon EC2 实例

1. 创建 IAM 客户端。

   导入

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

```
use Aws\Iam\IamClient;
```

示例代码

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

2. 创建 IAM 角色，并为其授予您将使用的操作和资源的权限。

   示例代码

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

3. 创建 IAM 实例配置文件，并存储结果的 Amazon 资源名称 (ARN)。

   > **Note**
   >
   > 如果您使用 IAM 控制台代替 适用于 PHP 的 Amazon SDK，则控制台会自动创建实例配置文件，并为其指定与其对应的角色相同的名称。
   >
   > 示例代码

```
$IPN = 'InstanceProfileName';

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

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

4. 创建 Amazon EC2 客户端。

   导入

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

```
use Aws\Ec2\Ec2Client;
```

示例代码

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

5. 将实例配置文件添加到正在运行或已停止的 Amazon EC2 实例。使用您的 IAM 角色的实例配置文件名称。

   示例代码

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

有关更多信息，请参阅《Amazon EC2 用户指南》中的适用于 Amazon EC2 的 IAM 角色。

使用 Amazon ECS 任务的 IAM 角色

亚马逊弹性容器服务 (Amazon ECS) 中的任务可以扮演 IAM 角色来调 Amazon 用 API。这是管理供应用程序使用的凭证的策略，与 Amazon EC2 实例配置文件为 Amazon EC2 实例提供凭证的方式相似。

您可以将使用临时 Amazon 证书的 IAM 角色与 ECS 任务定义或 `RunTask` API 操作相关联，而不必为容器创建和分配长期证书，也无需使用 Amazon EC2 实例的角色。

有关使用容器任务可以代入的 IAM 角色的更多信息，请参阅 Amazon ECS 开发人员指南中的任务 IAM 角色主题。有关在任务定义中以 `taskRoleArn` 形式使用任务 IAM 角色的示例，另请参阅 Amazon ECS 开发人员指南中的示例任务定义。

在另一个角色中扮演 IAM 角色 Amazon Web Services 账户

当您在 Amazon Web Services 账户 （账户 A）中工作并想在另一个账户（账户 B）中担任角色时，必须先在账户 B 中创建 IAM 角色。该角色允许您的账户（账户 A）中的实体在账户 B 中执行特定操作。有关跨账户访问的更多信息，请参阅教程：使用 IAM 角色委派跨 Amazon 账户访问权限。

在账户 B 中创建了角色之后，请记录角色 ARN。当您代入账户 A 中的角色时，您将使用此 ARN。您使用账户 A 中与您的实体关联的 Amazon 证书代入该角色。

使用您的凭据创建 Amazon STS 客户端 Amazon Web Services 账户。在以下示例中，我们使用了一个凭证配置文件，但您可以使用任何方法。对于新创建的 Amazon STS 客户端，请调用 assume-role 并提供一个自定义的 sessionName。从结果中检索新的临时凭证。默认情况下，凭证有效期为一个小时。

示例代码

```
$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']
    ]
]);
```

有关更多信息，请参阅使用 IAM 角色或 适用于 PHP 的 Amazon SDK API 参考AssumeRole中。

使用具备 Web 身份的 IAM 角色

Web 联合身份验证允许客户在访问 Amazon 资源时使用第三方身份提供商进行身份验证。在您代入具备 Web 身份的角色之前，您必须先创建一个 IAM 角色并配置 Web 身份提供商 (IdP)。有关更多信息，请参阅 创建用于 Web 联合身份验证或 OpenID Connect 联合身份验证的角色（控制台）。

创建身份提供商并为您的 Web 身份创建角色后，使用 Amazon STS 客户端对用户进行身份验证。ProviderId 为你的身份提供 webIdentityToken 和，为用户提供权限的 IAM 角色的角色 ARN。

示例代码

```
$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']
    ]
]);
```

有关更多信息，请参阅 “通过基于 Web 的身份提供商进行AssumeRoleWithWebIdentity联合” 或 AssumeRoleWithWebIdentity“ 适用于 PHP 的 Amazon SDK API 参考”。

代入角色与配置文件

在 ~/.aws/credentials 中定义配置文件

您可以通过在中定义配置文件 适用于 PHP 的 Amazon SDK 来将配置为使用 IAM 角色~/.aws/credentials。

使用将代入的角色的 role_arn 设置来创建新的配置文件。还需要包含另一配置文件的 source_profile 设置，以及有权代入 IAM 角色的凭证。有关这些配置设置的更多详细信息，请参阅《工具参考指南》Amazon SDKs 和《工具参考指南》中的代入角色凭证。

例如，在下面的 ~/.aws/credentials 中，project1 配置文件设置 role_arn 并指定 default 配置文件作为凭证来源，以验证与其关联的实体是否可以代入该角色。

```
[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
```

如果在实例化客户端时设置 AWS_PROFILE 环境变量，或使用 profile 参数，则将代入在 project1 中指定的角色，并使用 default 配置文件作为源凭证。

以下片段展示了在 S3Client 构造函数中使用 profile 参数的情况。S3Client 将拥有与 project1 配置文件所关联角色相关联的权限。

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

在 ~/.aws/config 中定义配置文件

~/.aws/config 文件还可以包含要代入的配置文件。如果设置了环境变量 AWS_SDK_LOAD_NONDEFAULT_CONFIG，则适用于 PHP 的 SDK 会从 config 文件中加载配置文

件。设置 AWS_SDK_LOAD_NONDEFAULT_CONFIG 后，SDK 会同时从 ~/.aws/config 和 ~/.aws/credentials 中加载配置文件。来自 ~/.aws/credentials 的配置文件最后加载，它们将优先于 ~/.aws/config 中同名的配置文件。来自任一位置的配置文件都可用作 source_profile 或要代入的配置文件。

以下示例使用 config 文件中定义的 project1 配置文件和 credentials 文件中的 default 配置文件。也需设置 AWS_SDK_LOAD_NONDEFAULT_CONFIG。

```
# 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
```

当 S3Client 构造函数运行时（如以下代码段所示），将使用与 default 配置文件关联的凭证来代入 project1 配置文件中定义的角色。

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

使用适用于 PHP 的 SDK Amazon STS 中的临时证书

Amazon Security Token Service (Amazon STS) 允许您为 IAM 用户或通过联合身份验证进行身份验证的用户申请有限权限、临时证书。有关深入了解，请参阅 IAM 用户指南中的临时安全凭证。您可以使用临时安全证书访问大多数 Amazon 服务。有关接受临时安全凭证的服务列表，请参阅 IAM 用户指南中的使用 IAM 的Amazon 服务。

临时证书的一个常见用例是，通过第三方身份提供商对用户进行身份验证，授予移动或客户端应用程序访问 Amazon 资源的权限（请参阅 Web 联合身份验证）。

获得临时凭证

Amazon STS 有几个返回临时证书的GetSessionToken操作，但演示起来最简单。以下代码段通过调用 PHP SDK 的 STS 客户端的 getSessionToken 方法来检索临时凭证。

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

$stsClient = $sdk->createSts();

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

和其他 Amazon STS 操作的GetSessionToken结果始终包含一个'Credentials'值。如果打印$result（例如，通过使用 print_r($result)），则看起来如下所示。

```
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>'
    )
    ...
)
```

向提供临时证书 适用于 PHP 的 Amazon SDK

您可以通过实例化 Amazon 客户端并直接传入从 Amazon STS 中接收到的值，将临时证书用于其他客户端。

```
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']
    ]
]);
```

您还可以构建 `Aws\Credentials\Credentials` 对象并在实例化客户端时使用该对象。

```
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
]);
```

但是，提供临时凭证的最佳 方式是使用 `StsClient` 随附的 `createCredentials()` 帮助程序方法。此方法从 Amazon STS 结果中提取数据并为您创建`Credentials`对象。

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

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

有关为何可能需要在应用程序或项目中使用临时证书的更多信息，请参阅 Amazon STS 文档中的授予临时访问权限的场景。

在适用于 PHP 的 SDK 中创建匿名客户端

在某些情况下，您可能想创建不与任何凭证关联的客户端。这样您便可以向服务发出匿名请求。

例如，您可以同时配置 Amazon S3 对象和 Amazon CloudSearch 域以允许匿名访问。

要创建匿名客户端，您可以将 `'credentials'` 选项设置为 `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' => 'amzn-s3-demo-bucket',
    'Key'    => 'my-key',
]);
```

使用适用于 PHP 的 SDK 中的 Amazon 通用运行时 (Amazon CRT) 扩展

Amazon CRT 库提供了基本功能，性能良好，几个 Amazon SDKs库占用空间最小。本主题讨论适用于 PHP 的 SDK 何时使用 Amazon CRT 以及如何安装 Amazon CRT 扩展。

当你需要安装 C Amazon RT 扩展时

适用于 PHP 的 SDK 使用 Amazon CRT 库的授权和校验和功能。使用以下功能时需要 Amazon CRT 扩展程序：

- Amazon S3 多区域访问点
- Amazon EventBridge 全球终端节点
- Amazon Simple Storage Service（Amazon S3）中的 CRC-32C 校验和算法

如果您使用上面列出的功能，但您的 PHP 环境中未安装 Amazon CRT 扩展，则 PHP SDK 将报告错误消息并提醒您安装该扩展。

安装 Amazon 通用运行时 (Amazon CRT) 扩展

有关如何安装 Amazon CRT 扩展的说明可在GitHub存储库的主页上找到 aws-crt-php。

使用 适用于 PHP 的 Amazon SDK 版本 3

本章提供了有关使用 适用于 PHP 的 Amazon SDK 版本 3 进行 Amazon Web Services 服务 有效交互的全面指导。您将学习一些必备技巧，包括提出服务请求、处理错误以及利用高级 SDK 功能在Amazon上构建可靠的 PHP 应用程序。

无论您是在构建简单的应用程序还是复杂的系统，本章中描述的技术都将帮助您优化代码、改进错误处理并实现 Amazon Web Services 服务 通过 PHP SDK 使用的高效模式。

主题

使用 适用于 PHP 的 Amazon SDK 版本 3 提出 Amazon Web Services 服务 请求

SDK 请求工作流程概述

使用 适用于 PHP 的 Amazon SDK 版本 3 在所有版本中都遵循一致的模式 Amazon Web Services 服务。基本工作流程包括三个主要步骤：

1. 创建服务客户端-为 Amazon Web Services 服务 要使用的实例化一个客户端对象。
2. 执行操作 — 在客户端上调用与服务 API 中的操作相对应的方法。
3. 处理结果 — 使用成功时返回的类似数组的 Result 对象，或者处理失败时引发的任何 Exception。

以下各节将详细说明上述每个步骤，首先介绍如何创建和配置服务客户端。

创建基本服务客户端

您可以通过向客户端的构造函数传递选项的关联数组来创建客户端。

导入

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

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

示例代码

```
//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'.
```

有关可选“版本”参数的信息，请参阅配置选项主题。

请注意，我们并未向客户端显式提供凭证。这是因为 SDK 使用默认凭证提供程序链来查找凭证信息。

在 适用于 PHP 的 Amazon SDK 版本 3 的客户端构造器选项 中详细介绍了所有通用的客户端配置选项。创建的客户端不同，提供的选项数组也不同。每个客户端的 API 文档中介绍了这些自定义客户端配置选项。

上述示例显示的是基本客户端创建过程，您也可以自定义服务客户端以满足特定要求。有关通过代码配置服务客户端的更多详细信息，请参阅在 适用于 PHP 的 Amazon SDK 版本 3 的代码中配置服务客户端。如果需要使用外部配置文件或环境变量来配置服务客户端，请参阅在外部配置 适用于 PHP 的 Amazon SDK 版本 3 的服务客户端。

发出请求

通过在客户端对象上调用同名方法，可发出服务请求。例如，要执行 Amazon S3 PutObject操作，您需要调用`Aws\S3\S3Client::putObject()`方法。

导入

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

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

示例代码

```
// 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' => 'amzn-s3-demo-bucket',
    'Key' => 'my-key',
    'Body' => 'this is the body!'
]);

// Download the contents of the object.
$result = $s3Client->getObject([
    'Bucket' => 'amzn-s3-demo-bucket',
    'Key' => 'my-key'
]);

// Print the body of the result by indexing into the result object.
echo $result['Body'];
```

客户端提供的操作，以及输入、输出的结构是根据服务描述文件在运行时定义的。创建客户端时，如果您未提供服务模型的 `version` 参数（例如，“2006-03-01”或“latest”），则客户端将默认使用最新版本。开发工具包会根据提供的版本找到相应的配置文件。

所有操作方法 (如 `putObject()`) 均接受单独的参数，或代表操作参数的关联数组。此数组的结构（以及结果对象的结构）是在开发工具包的 API 文档中针对每个操作进行定义的（例如，请参阅 API 文档中的 putObject 操作）。

HTTP 处理程序选项

您还可以使用特殊的 @http 参数微调底层 HTTP 处理程序执行请求的方式。可包含在 @http 参数中的选项与您在使用“http”客户端选项对客户端进行实例化时可设置的选项相同。

```
// Send the request through a proxy
$result = $s3Client->putObject([
    'Bucket' => 'amzn-s3-demo-bucket',
    'Key'    => 'my-key',
    'Body'   => 'this is the body!',
    '@http'  => [
        'proxy' => 'http://192.168.16.1:10'
    ]
]);
```

使用 Result 对象

执行成功的操作会返回 Aws\Result 对象。开发工具包不会返回服务的原始 XML 或 JSON 数据，而是会将响应数据强制加入关联数组结构中。这样可以根据它对特定服务和底层响应结构的认知，将数据的某些方面规范化。

您可以访问 AWS\Result 对象的数据，就像访问关联 PHP 数组那样。

导入

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

示例代码

```
// 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();
```

结果对象的内容取决于执行的操作和服务的版本。每个 API 操作的结果结构均记录在每个操作的 API 文档中。

SDK 集成了用于搜索和操作 J SON 数据的 DSL，或者在我们的例子中是 PHP 数组。JMESPath结果对象包含 `search()` 方法，可用于更具声明性地从结果中提取数据。

示例代码

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

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

适用于 PHP 的 Amazon SDK 版本 3 中的命令对象

适用于 PHP 的 Amazon SDK 使用命令模式封装稍后用于传输 HTTP 请求的参数和处理程序。

隐式使用命令

如果您检查任何客户端类，就会发现对应于 API 操作的方法实际上并不存在。它们是使用 `__call()` 魔术方法实施。这些虚拟方法实际上是封装开发工具包对命令对象的使用的快捷方式。

您通常不需要直接与命令对象交互。当您调用类似于 `Aws\S3\S3Client::putObject()` 的方法时，开发工具包实际上会根据所提供的参数创建 `Aws\CommandInterface` 对象，执行命令，并返回填充的 `Aws\ResultInterface` 对象（或针对错误引发异常）。调用客户端的任意 `Async` 方法（例如 `Aws\S3\S3Client::putObjectAsync()`）时，会发生类似流程：客户端根据所提供的参数创建一个命令，序列化 HTTP 请求，发起请求，并返回 Promise。

以下示例在功能上等效。

```
$s3Client = new Aws\S3\S3Client([
    'version' => '2006-03-01',
```

```
    'region'  => 'us-standard'
]);

$params = [
    'Bucket' => 'amzn-s3-demo-bucket',
    '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);
```

命令参数

所有命令均支持几个特殊参数，这些参数不属于服务的 API，但可以控制开发工具包的行为。

@http

使用此参数可以微调基础 HTTP 处理程序执行请求的方式。可包含在 @http 参数中的选项与您在使用“http”客户端选项对客户端进行实例化时可设置的选项相同。

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

@retries

与“retries”客户端选项类似，@retries 控制一个命令在被视为已失败之前可以重试的次数。将其设置为 0 可禁用重试。

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

> **Note**
>
> 如果您已对客户端禁用重试，则无法选择性地对传递给该客户端的各个命令启用重试。

创建命令对象

您可以使用客户端的 getCommand() 方法创建命令。它不会立即执行或传输 HTTP 请求，而是仅在传递到客户端的 execute() 方法时才执行。这样一来，您有机会在执行命令之前修改命令对象。

```
$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);
```

命令 **HandlerList**

从客户端创建命令后，该命令将得到客户端 Aws\HandlerList 对象的一个克隆。该命令将得到客户端处理程序列表的一个克隆，以允许命令使用不影响客户端执行的其他命令的自定义中间件和处理程序。

这意味着您可以对每条命令使用不同的 HTTP 客户端（例如 Aws\MockHandler），并通过中间件对每条命令添加自定义行为。以下示例使用 MockHandler 创建模拟结果，而不是发送实际 HTTP 请求。

```
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'
```

除了更改命令所使用的处理程序外，您还可以将自定义中间件注入该命令。以下示例使用 tap 中间件，其作用是充当处理程序列表中的观察者。

```
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

Aws\CommandPool 使您能够使用生成 Aws\CommandInterface 对象的迭代器并发执行命令。CommandPool 确保并发执行的命令数量保持恒定，同时迭代池中的命令（当命令完成时，会执行更多命令，以确保恒定的池大小）。

下面是使用 CommandPool 发送几条命令的简单示例。

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

// Create the client
$client = new S3Client([
```

```
    'region'  => 'us-standard',
    'version' => '2006-03-01'
]);

$bucket = 'amzn-s3-demo-bucket';
$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();
```

该示例对于 CommandPool 显得非常苍白无力。我们来试试更复杂的示例。假设您要将磁盘上的文件上传到 Amazon S3 存储桶。要获取磁盘文件的列表，我们可以使用 PHP 的 DirectoryIterator。此迭代器生成 SplFileInfo 对象。CommandPool 接受生成 Aws\CommandInterface 对象的迭代器，因此我们将映射 SplFileInfo 对象以返回 Aws\CommandInterface 对象。

```
<?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 = 'amzn-s3-demo-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` 配置

`Aws\CommandPool` 构造函数接受各种配置选项。

concurrency（可调用|整数）

并发执行的命令的最大数量。提供一个函数来动态调整池大小。该函数将获得当前待处理请求数，并且预计会返回一个表示新池大小限制的整数。

before（可调用）

发送每个命令之前要调用的函数。`before` 函数接受命令和该命令的迭代器密钥。您可以在发送命令之前根据需要更改 `before` 函数中的命令。

fulfilled（可调用）

执行 Promise 时要调用的函数。此函数将获得结果对象、生成该结果的迭代器的 ID，以及可解析或拒绝的聚合 Promise（如果您需要让池短路）。

rejected（可调用）

拒绝 Promise 时要调用的函数。此函数将获得 `Aws\Exception` 对象、生成该异常的迭代器的 ID，以及可解析或拒绝的聚合 Promise（如果您需要让池短路）。

命令之间的手动垃圾回收

如果您达到大型命令池的内存限制，这可能是由于在达到您的内存限制时开发工具包生成的、但尚未由 PHP 垃圾回收器收集的循环引用导致的。在命令之间手动调用收集算法可允许在达到该限制之前收集循环。以下示例将创建一个 CommandPool，该池将在发送每个命令之前使用一个回调来调用收集算法。请注意，调用垃圾回收器会降低性能，最佳用法将取决于您的使用案例和环境。

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

使用 适用于 PHP 的 Amazon SDK 版本 3 进行异步编程

您可以使用开发工具包的异步功能并发发送命令。您可以在操作名称后添加 Async 后缀，异步发送请求。这样可以启动请求并返回 Promise。

如果成功，结果对象可满足 Promise；如果失败，异常会导致拒绝 Promise。您可以创建多个 Promise，并由它们在底层 HTTP 处理程序传输请求时并发发送 HTTP 请求。

导入

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

示例代码

```
// 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();
```

您可以使用 Promise 的 `wait` 方法，强制 Promise 同步完成。默认情况下，强制完成 Promise 也会“解封”Promise 的状态，这意味着它会返回 Promise 的结果或引发异常。如果针对 Promise 调用 `wait()`，流程将会阻塞，直到 HTTP 请求完成并填充结果，或引发异常。

如果使用具有事件循环库的开发工具包，请不要阻塞结果。请使用结果的 `then()` 方法，在操作完成时访问已解决或被拒绝的 Promise。

导入

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

示例代码

```
// 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();
    });
```

适用于 PHP 的 Amazon SDK 版本 3 中的 Promise

适用于 PHP 的 Amazon SDK使用 Promise 支持异步工作流，这种异步性允许同时发送 HTTP 请求。开发工具包使用的 Promise 规范为 Promises/A+。

什么是 Promise？

Promise 表示异步操作的最终结果。与 Promise 交互的主要方式是通过其 `then` 方法。此方法注册回调以接收 Promise 的最终值或无法执行 Promise 的原因。

适用于 PHP 的 Amazon SDK依赖 guzzlehttp/promises Composer 程序包来实施 Promise。Guzzle Promise 支持阻止和非阻止性工作流，并可与任何非阻止性事件循环一起使用。

> **Note**
>
> 使用单一线程在 适用于 PHP 的 Amazon SDK 中同时发送 HTTP 请求，其中非阻止性调用用于在响应状态更改（例如，执行或拒绝 Promise）时传输一个或多个 HTTP 请求。

开发工具包中的 Promise

Promise 的使用贯穿整个开发工具包。例如，Promise 用于 SDK 提供的大多数高级别抽象化处理：分页器、Waiter、命令池、分段上传、S3 目录/存储桶传输，等等。

当您调用任何 `Async` 后缀的方法时，该开发工具包提供的所有客户端均返回 Promise。例如，以下代码展示了如何创建 Promise 以获取 Amazon DynamoDB`DescribeTable` 操作的结果。

```
$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']);
```

请注意，您可以调用 `describeTable` 或 `describeTableAsync`。这些方法是客户端上的魔术 `__call` 方法，受到与该客户端关联的 API 模型和 `version` 号的支持。通过调用 `describeTable` 之类的没有 `Async` 后缀的方法，客户端将阻止其发送 HTTP 请求，并返回 `Aws\ResultInterface` 对象或引发 `Aws\Exception\AwsException`。通过在操作名称后面添加 `Async` 后缀（即 `describeTableAsync`），客户端将创建一个最终使用 `Aws\ResultInterface` 对象执行或因 `Aws\Exception\AwsException` 而被拒绝的 Promise。

> **Important**
>
> 返回 Promise 时，结果可能已到达（例如，使用模拟处理程序时），或者 HTTP 请求可能未被启动。

您可以通过使用 then 方法向 Promise 注册回调。此方法接受两个回调（$onFulfilled 和 $onRejected），两者均为可选项。如果执行 Promise，则调用 $onFulfilled 回调；如果 Promise 被拒绝（表示失败），则调用 $onRejected 回调。

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

同时执行命令

多个 Promise 可以组合在一起，以便同时执行。这可通过将开发工具包与非阻止性事件循环集成或者通过构建多个 Promise 并等待它们同时完成来实现。

```
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**
>
> CommandPool 提供更强大的机制用于同时执行多个 API 操作。

串联 Promise

Promise 非常棒的一个方面是它们可以组合，从而允许您创建转换管道。Promise 是通过将 `then` 回调与后续 `then` 回调串联在一起而构成的。`then` 方法的返回值是根据所提供回调的结果而执行或拒绝的 Promise。

```
$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**
>
> Promise 回调的返回值是提供给下游 Promise 的 `$value` 参数。如果您想为下游 Promise 链提供一个值，必须在回调函数中返回一个值。

拒绝转发

您可以注册在 Promise 被拒绝时要调用的回调。如果任何回调引发异常，Promise 会因该异常而被拒绝，链中接下来的 Promise 也会因该异常而被拒绝。如果您成功从 `$onRejected` 回调返回一个值，则 Promise 链中接下来的 Promise 将使用 `$onRejected` 回调的返回值执行。

正在等待 Promise

您可以通过使用 Promise 的 `wait` 方法来同步强制完成 Promise。

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

如果在调用 `wait` 函数时遇到异常，Promise 将因该异常而被拒绝，并且会引发该异常。

```
use Aws\Exception\AwsException;

$promise = $client->listTablesAsync();

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

对已执行的 Promise 调用 `wait` 不会触发 wait 函数。只是返回之前提供的值。

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

对已拒绝的 Promise 调用 `wait` 会引发异常。如果拒绝原因是 `\Exception` 的实例，则会引发该原因。否则会引发 `GuzzleHttp\Promise\RejectionException`，并且该原因可通过调用异常的 `getReason` 方法获得。

> **Note**
>
> 适用于 PHP 的 Amazon SDK中的 API 操作调用因 `Aws\Exception\AwsException` 类的子类而被拒绝。但是，由于添加了会修改拒绝原因的自定义中间件，因此发送到 `then` 方法的原因可能会不同。

取消 Promise

可以使用 Promise 的 `cancel()` 方法取消 Promise。如果 Promise 已解析，则调用 `cancel()` 将没有任何作用。取消 Promise 会取消该 Promise 以及任何等待从该 Promise 发送的 Promise。已取消的 Promise 被拒绝并显示 `GuzzleHttp\Promise\RejectionException`。

组合 Promise

您可以将 Promise 组合成聚合 Promise，以构建更复杂的工作流。`guzzlehttp/promise` 程序包包含各种可用于组合 Promise 的函数。

您可以在 namespace-GuzzleHttp.Promise 中找到适用于所有 Promise 集合函数的 API 文档。

each 和 each_limit

当您需要同时执行 `Aws\CommandInterface` 命令的任务队列并且池大小固定（这些命令可位于内存中，也可由延迟迭代器生成）时，请使用 CommandPool。`CommandPool` 可确保同时发送固定数量的命令，直到提供的迭代器用尽。

`CommandPool` 只使用由同一客户端执行的命令。您可以使用 `GuzzleHttp\Promise\each_limit` 函数同时执行不同客户端的发送命令（使用固定的池大小）。

```
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 协同例程

Guzzle Promise 库的一个更强大的功能是允许您使用协同例程 Promise，这使编写异步工作流似乎更像编写传统同步工作流。事实上，适用于 PHP 的 Amazon SDK 在大多数高级别抽象化处理中使用协同例程 Promise。

假设您要创建多个存储桶并在存储桶变为可用时将文件上传到存储桶，并且您想同时执行这些操作，以便尽快完成操作。您可以通过使用 `all()` Promise 函数将多个协同例程 Promise 组合在一起，从而轻松完成这一操作。

```
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 = ['amzn-s3-demo-bucket1', 'amzn-s3-demo-bucket2', 'amzn-s3-demo-bucket3'];
$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();
```

处理 适用于 PHP 的 Amazon SDK 版本 3 中的错误

同步处理错误

如果在执行操作时发生错误，将引发异常。因此，如果您需要在代码中处理错误，请围绕您的操作使用 try/catch 数据块。发生错误时，开发工具包会引发特定于服务的异常。

下面的示例使用了 Aws\S3\S3Client。如果发生错误，引发的异常将为 Aws\S3\Exception \S3Exception 类型。开发工具包引发的所有特定于服务的异常均是从 Aws\Exception \AwsException 类中扩展而来的。这个类中包含有关故障的有用信息，包括请求 ID、错误代码和错误类型。注意：对于某些支持它的服务，响应数据被强制转换为关联数组结构（类似于 Aws\Result 对象），可以像普通 PHP 关联数组那样进行访问。toArray() 方法返回任意此类数据（如果存在）。

导入

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

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

示例代码

```
// 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' => 'amzn-s3-demo-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());
}
```

异步处理错误

在发送异步请求的时候不会引发异常。您必须使用返回的 Promise 的 `then()` 或 `otherwise()` 方法来接收结果或错误。

导入

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

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

示例代码

```
//Asynchronous Error Handling
$promise = $s3Client->createBucketAsync(['Bucket' => 'amzn-s3-demo-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);
});
```

您可以“解封”Promise，从而引发异常。

导入

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

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

示例代码

```
$promise = $s3Client->createBucketAsync(['Bucket' => 'amzn-s3-demo-bucket']);
```

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

适用于 PHP 的 Amazon SDK 版本 3 中的处理程序和中间件

适用于 PHP 的 Amazon SDK的主要扩展机制是，通过处理程序和中间件进行扩展。每个开发工具包客户端类都拥有一个 `Aws\HandlerList` 实例，可通过客户端的 `getHandlerList()` 方法访问。您可以检索客户端的 `HandlerList`，并对其进行修改来添加或删除客户端行为。

处理程序

处理程序是一个将命令和请求实际转换为结果的函数。处理程序通常发送 HTTP 请求。处理程序可由中间件组成，以增强行为。处理程序是一个函数，它接受 `Aws\CommandInterface` 和 `Psr\Http`

\Message\RequestInterface，并返回用 Aws\ResultInterface 执行或因 Aws\Exception \AwsException 原因而被拒绝的 Promise。

以下处理程序的每次调用均返回相同的模拟结果。

```
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);
};
```

您可以在客户端的构造函数中提供 handler 选项，将此处理程序与开发工具包客户端结合使用。

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

您还可以在客户端构造完成后使用 setHandler 的Aws\ClientInterface 方法更改其处理程序。

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

> **Note**
>
> 要在多区域客户端构造完成后更改其处理程序，请使用 Aws\MultiRegionClient 的 useCustomHandler 方法。
>
> ```
> $multiRegionClient->useCustomHandler($myHandler);
> ```

模拟处理程序

我们建议在使用开发工具包编写测试时使用 MockHandler。您可以使用 Aws\MockHandler 返回模拟结果或引发模拟异常。加入队列的结果或异常，MockHandler 会按 FIFO 顺序将它们从队列中移除。

```
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();
```

中间件

中间件是一类特殊的高级函数，可对传输命令的行为进行增强，并委托给“下一个”处理程序。中间件函数接受 Aws\CommandInterface 和 Psr\Http\Message\RequestInterface，并返回用 Aws\ResultInterface 执行或因 Aws\Exception\AwsException 原因而被拒绝的 Promise。

中间件是更高阶的函数，可修改经过中间件传递的命令、请求或结果。中间件具有以下形式。

```
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;
        };
    };
};
```

中间件接收要执行的命令和可选的请求对象。中间件可增强请求和命令，或将它们保留原样。然后中间件会调用链条中的下一个处理程序，或选择将下一个处理程序短路并返回 Promise。调用下一个处理程序创建的 Promise，可使用 Promise 的 `then` 方法进行增强，在将 Promise 返回中间件组之前，修改最终结果或错误。

HandlerList

开发工具包使用 `Aws\HandlerList` 管理执行命令所用的中间件和处理程序。每个开发工具包客户端都拥有一个 `HandlerList`，这个 `HandlerList` 会克隆并添加到客户端创建的每条命令。您可以将中间件添加到客户端的 `HandlerList`，为客户端创建的每条命令附加要使用的中间件和默认处理程序。您可以修改特定命令的 `HandlerList`，添加或删除特定命令的中间件。

`HandlerList` 表示用于包装处理程序的一组中间件。`HandlerList` 可将中间件组拆分为一些具名步骤，表示传输命令的生命周期中的各个部分。这样有助于管理中间件的列表，以及包装处理程序的顺序。

1. `init` - 添加默认参数
2. `validate` - 验证必要参数
3. `build` - 序列化 HTTP 请求，准备发送
4. `sign` - 对序列化 HTTP 请求进行签名
5. <handler>（不是一个步骤，但执行实际传输过程）

init

生命周期中的这个步骤表示命令的初始化，以及尚未序列化的请求。此步骤通常用于在命令中添加默认参数。

您可以使用 `init` 和 `appendInit` 方法在 `prependInit` 步骤中添加中间件，其中 `appendInit` 将中间件添加到 `prepend` 列表的最后，`prependInit` 将中间件添加到 `prepend` 列表的开头。

```
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');
```

验证

生命周期中的这个步骤用于验证命令的输入参数。

您可以使用 `validate` 和 `appendValidate` 方法在 `prependValidate` 步骤中添加中间件，其中 `appendValidate` 将中间件添加到 `validate` 列表的最后，`prependValidate` 将中间件添加到 `validate` 列表的开头。

```
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

生命周期中的这个步骤用于针对所执行的命令序列化 HTTP 请求。下游生命周期事件会收到一条命令和 PSR-7 HTTP 请求。

您可以使用 build 和 appendBuild 方法在 prependBuild 步骤中添加中间件，其中 appendBuild 将中间件添加到 build 列表的最后，prependBuild 将中间件添加到 build 列表的开头。

```
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');
```

签名

生命周期中的这个步骤用于在通过线路发送 HTTP 请求之间进行签名。通常，在 HTTP 请求经过签名后，您即不应该进行更改，以避免签名错误。

这是处理程序传输 HTTP 请求之前 HandlerList 中的最后一步。

您可以使用 sign 和 appendSign 方法在 prependSign 步骤中添加中间件，其中 appendSign 将中间件添加到 sign 列表的最后，prependSign 将中间件添加到 sign 列表的开头。

```
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');
```

可用中间件

您可使用开发工具包提供的若干中间件增强客户端的行为，或观察命令的执行。

mapCommand

如果要在将命令序列化为 HTTP 请求之前修改命令，Aws\Middleware::mapCommand 中间件很有用。例如，可使用 mapCommand 执行验证或添加默认参数。mapCommand 函数接受的可调用函数可接受 Aws\CommandInterface 对象并返回 Aws\CommandInterface 对象。

```
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'] = 'amzn-s3-demo-bucket';
        // Be sure to return the command!
        return $command;
    }),
    'add-param'
);
```

mapRequest

如果您需要修改已序列化但尚未发送的请求，Aws\Middleware::mapRequest 中间件很有用。例如，可使用它为请求添加自定义 HTTP 标题。mapRequest 函数接受的可调用函数可接受 Psr\Http\Message\RequestInterface 参数并返回 Psr\Http\Message\RequestInterface 对象。

```
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' => 'amzn-s3-demo-bucket'
]);

// 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'
);
```

当执行命令时，会随自定义标题发送。

> **Important**
>
> 请注意，中间件是在 build 步骤的最后追加到处理程序列表中的。这是为了确保在调用中间件之前生成请求。

mapResult

如果您需要修改命令执行的结果，Aws\Middleware::mapResult 中间件很有用。mapResult 函数接受的可调用函数可接受 Aws\ResultInterface 参数并返回 Aws\ResultInterface 对象。

```
use Aws\Middleware;
use Aws\ResultInterface;

$command = $s3Client->getCommand('HeadObject', [
    'Key'    => 'test',
    'Bucket' => 'amzn-s3-demo-bucket'
]);

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

命令执行后，返回的结果中将包含 foo 属性。

历史记录

history 中间件可用于测试开发工具包是否执行了您期望的命令、发送了您期望的 HTTP 请求，并收到了您期望的结果。这个中间件与 Web 浏览器的历史记录基本类似。

```
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));
```

在清除条目之前，Aws\History 历史记录容器中默认可存储 10 条记录，您可以将要保留的条目数量传递到构造函数中，从而自定义条目数量。

```
// Create a history container that stores 20 entries
$history = new History(20);
```

您可以在执行传递历史记录中间件的请求之后，检查历史记录容器。

```
// 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

tap 中间件可用作观察工具。您可以使用此中间件，在通过中间件链条发送命令时调用函数。tap 函数接受的可调用函数可接受 Aws\CommandInterface，以及被执行的可选 Psr\Http\Message\RequestInterface。

```
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";
        }
    }
);
```

创建自定义处理程序

处理程序只是一个接受 Aws\CommandInterface 对象和 Psr\Http\Message\RequestInterface 对象的函数，它可返回由 GuzzleHttp\Promise\PromiseInterface 满足、或由 Aws\ResultInterface 拒绝的 Aws\Exception\AwsException。

虽然开发工具包有多个 @http 选项，处理程序只需要知道如何使用以下选项：

- connect_timeout
- debug
- decode_content（可选）
- delay
- progress（可选）
- proxy
- sink
- synchronous（可选）
- stream（可选）
- -timeout
- 确认
- http_stats_receiver（可选）- 如果使用 stats 配置参数进行请求，通过 HTTP 传输统计数据的关联数组进行调用的函数。

除非此选项指定为可选，否则处理程序必须处理该选项，或必须返回拒绝的 Promise。

除了处理特定的 @http 选项，处理程序还必须添加具有以下形式的 User-Agent 标题，其中“3.X”可替换为 Aws\Sdk::VERSION，“HandlerSpecificData/version ...”应替换为您的处理程序特定的 User-Agent 字符串。

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

适用于 PHP 的 Amazon SDK 版本 3 中的流

由于 集成了 PSR-7适用于 PHP 的 Amazon SDK HTTP 消息标准，因此它在内部使用 PSR-7 StreamInterface，作为对 PHP 流的抽象。命令的输入字段如果定义为 blob（例如 BodyS3::PutObject 命令的 参数），字符串、PHP 流资源或 Psr\Http\Message\StreamInterface 的实例均可满足该命令。

> **Warning**
>
> 开发工具包会负责处理所有作为命令输入参数提供的原始 PHP 流资源。将代表您使用并关闭流。
> 如果您需要在开发工具包操作和代码之间共享流，请先将它包装在 GuzzleHttp\Psr7\Stream 的实例中，再作为参数包含在命令中。开发工具包会使用流，因此您的代码

需要考虑流的内部游标的移动。Guzzle 流在被 PHP 的垃圾回收器销毁时会对底层流资源调用 `fclose`，因此您无需自己关闭流。

流装饰器

Guzzle 提供多种流装饰器，可用于控制开发工具包和 Guzzle 与作为命令输入参数提供的流资源进行交互的方式。这些装饰器可修改处理程序在给定流上读取和搜寻的方式。以下是列表的一部分，在 GuzzleHttpPsr7 存储库中可以找到更多内容。

AppendStream

GuzzleHttp\Psr7\AppendStream

从多个流中依次读取。

```
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

允许在不可搜寻的流上搜寻之前读取的字节。如果由于需要倒回流导致未能传输不可搜寻的正文（例如重定向后），它非常有用。从远程流中读取的数据将缓存在 PHP 临时流中，这样上次读取的字节将首先缓存到内存中，然后再缓存到磁盘上。

```
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

使用 PHP 的 zlib.inflate 筛选条件来增加或减少使用 gzip 压缩的内容。

这个流装饰器会跳过给定流的前 10 个字节，以删除 gzip 标头，将提供的流转换为 PHP 流资源，然后附加 zlib.inflate 筛选器。然后该流将会转换回 Guzzle 流资源，用作 Guzzle 流。

LazyOpenStream

GuzzleHttp\Psr7\LazyOpenStream

仅在流中发生 I/O 操作后，再延时读取或写入已打开的文件。

```
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

用于读取现有流对象的子集或分片。如需将较大文件拆分为小块，以便分段发送（例如 Amazon S3 Multipart Upload API），那么它很有用。

```
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

对流进行包装，不允许搜寻。

```
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

提供只读流，从 PHP 可调用函数中提取数据。

如果调用提供的可调用函数，PumpStream 会向该可调用函数传递请求读取的数据量。可调用函数可以选择忽略此值，并返回少于请求或多于请求的字节。提供的可调用函数返回的所有额外数据将缓存于内部，直到使用 PumpStream 的 read() 函数耗尽。如果没有可读取的数据，提供的可调用函数必须返回 false。

实施流装饰器

由于有了 GuzzleHttp\Psr7\StreamDecoratorTrait，创建流装饰器变得非常容易。此特性通过代理底层流，提供实施 `Psr\Http\Message\StreamInterface` 的方法。只需 `use StreamDecoratorTrait`，并实施您的自定义方法。

例如，假设我们希望在每次读取流的最后一个字节时调用特定的函数。这可通过覆盖 `read()` 方法实施。

```
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;
    }
}
```

此装饰器可添加到任何现有流中，使用方法与以下示例类似。

```
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!"
```

使用适用于 PHP 的 Amazon SDK 版本 3 中的分页结果

某些 Amazon 服务操作会分页，并以截断的结果进行响应。例如，Amazon S3 `ListObjects` 操作每次最多只能返回 1000 个对象。与此类似的操作（前缀通常为“list”或“describe”）需要利用令牌（或标记）参数生成后续请求，以检索完整的结果集。

Paginator 是适用于 PHP 的 Amazon SDK的一种功能，充当此流程的抽象层，使开发人员能够更轻松地使用分页的 API。分页工具本质上是结果的迭代器。它们是使用客户端的 `getPaginator()` 方法创建的。如果调用 `getPaginator()`，您必须提供操作名称以及操作的参数（与执行操作时的方法相同）。您可以使用 `foreach` 迭代分页工具对象，以获得单个 `Aws\Result` 对象。

```
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'amzn-s3-demo-bucket'
]);

foreach ($results as $result) {
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
}
```

分页工具对象

`getPaginator()` 方法返回的对象是 `Aws\ResultPaginator` 类的实例。此类实现 PHP 的原生 `iterator` 接口，因此它可与 `foreach` 配合使用。它还可与迭代器函数 (如 `iterator_to_array`) 结合使用，并与 SPL iterators（如 `LimitIterator` 对象）良好集成。

分页工具对象每次只保留一“页”结果，并延时执行。这就意味着，请求数量是根据生成当前的结果页面的需求决定的。例如，Amazon S3 `ListObjects` 操作每次最多只返回 1000 个对象，如果您的存储桶中有约 10000 个对象，分页工具共需执行 10 次请求。如果您需要迭代结果，则会在开始迭代时执行第一个请求，在循环第二次迭代时执行第二个请求，依此类推。

枚举结果数据

分页工具对象拥有名为 `search()` 的方法，允许您为一组结果中的数据创建迭代器。您在调用 `search()` 时，请提供 JMESPath 表达式，指定要提取的数据。调用 `search()` 会返回迭代器，生成每页结果的表达式结果。在您迭代返回的迭代器时将延时评估。

以下示例与之前的代码示例等效，但使用 `ResultPaginator::search()` 方法可以更加精确。

```
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'amzn-s3-demo-bucket'
]);

foreach ($results->search('Contents[].Key') as $key) {
    echo $key . "\n";
}
```

可使用 JMESPath 表达式执行非常复杂的操作。例如，如果您希望打印所有的对象键和常用前缀（即针对存储桶执行 `ls`），可以执行以下操作。

```
// List all prefixes ("directories") and objects ("files") in the bucket
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket'    => 'amzn-s3-demo-bucket',
    'Delimiter' => '/'
]);

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

异步分页

您可以提供 `each()` 的 `Aws\ResultPaginator` 方法的回调，异步迭代分页工具的结果。分页工具生成的每个值都会调用该回调。

```
$results = $s3Client->getPaginator('ListObjects', [
    'Bucket' => 'amzn-s3-demo-bucket'
]);

$promise = $results->each(function ($result) {
```

```
    echo 'Got ' . var_export($result, true) . "\n";
});
```

> **Note**
>
> 您可以使用 each() 方法将 API 操作的结果分页，同时异步发送其他请求。

底层基于协同程序的 Promise 会生成回调的非 null 返回值。这就意味着，您可以从回调返回 Promise，而在继续迭代其余项目之前，必须解决该回调，也就是要将其他 Promise 合并到迭代中。回调返回的最后一个非 null 值，是可满足对任何下游 Promise 所做的 Promise 的结果。如果返回的最后一个值是 Promise，解决该 Promise 所得的结果将满足或拒绝下游 Promise。

```
// 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' => 'amzn-s3-demo-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();
```

适用于 PHP 的 Amazon SDK 版本 3 中的 Waiter

Waiter 通过对资源进行轮询，以一种抽象方式等待资源进入特定状态，从而更加轻松地使用最终一致性系统。您可以查看单一服务客户端版本的 API 文档，找到客户端支持的 Waiter 列表。要导航到那

里，请转到 API 文档中的客户端页面，导航到特定版本号（用日期表示），然后向下滚动到“Waiter”部分。此链接将带您进入 S3 的“Waiter”部分。

在以下示例中，使用 Amazon S3 客户端来创建存储桶。然后使用 Waiter 等待存储桶建好。

```
// Create a bucket
$s3Client->createBucket(['Bucket' => 'amzn-s3-demo-bucket']);

// Wait until the created bucket is available
$s3Client->waitUntil('BucketExists', ['Bucket' => 'amzn-s3-demo-bucket']);
```

如果 Waiter 轮询存储桶次数过多，会引发 `\RuntimeException` 异常。

Waiter 配置

Waiter 是由配置选项的关联数组推动的。特定 Waiter 所用的所有选项均有默认值，但可以覆盖这些默认值，以支持不同的等待策略。

将 `@waiter` 选项的关联数组传递到客户端的 `$args` 和 `waitUntil()` 方法的 `getWaiter()` 参数，可以修改 Waiter 配置选项。

```
// Providing custom waiter configuration options to a waiter
$s3Client->waitUntil('BucketExists', [
    'Bucket'  => 'amzn-s3-demo-bucket',
    '@waiter' => [
        'delay'       => 3,
        'maxAttempts' => 10
    ]
]);
```

delay (int)

两次轮询尝试之间的延迟秒数。每个 Waiter 都有默认的 `delay` 配置值，但您可能需要针对具体的使用情形修改此设置。

maxAttempts (int)

Waiter 失败之前轮询尝试的次数上限。此选项可确保您不会无限期地等待某个资源。每个 Waiter 都有默认的 `maxAttempts` 配置值，但您可能需要针对具体的使用情形修改此设置。

initDelay (int)

首次轮询尝试之前等待的秒数。如果您知道某一资源需要一段时间才能进入所需状态，此选项将会很有用。

before（可调用）

一个 PHP 可调用函数，在每次尝试之前调用。该可调用函数根据将要执行的 Aws\CommandInterface 命令以及目前已执行的尝试次数调用。before 可调用函数的用途可以是在执行命令之前修改命令，或提供进度信息。

```
use Aws\CommandInterface;

$s3Client->waitUntil('BucketExists', [
    'Bucket'  => 'amzn-s3-demo-bucket',
    '@waiter' => [
        'before' => function (CommandInterface $command, $attempts) {
            printf(
                "About to send %s. Attempt %d\n",
                $command->getName(),
                $attempts
            );
        }
    ]
]);
```

异步等待

除了同步等待，您还可以调用 Waiter 异步等待，同时发送其他请求或同时等待多个资源。

您可以使用客户端的 getWaiter($name, array $args = []) 方法检索客户端的 Waiter，从而访问 Waiter 的 Promise。使用 Waiter 的 promise() 方法启动 Waiter。Waiter 中最近执行的 Aws\CommandInterface 可满足 Waiter Promise，错误引起的 RuntimeException 可拒绝 Waiter Promise。

```
use Aws\CommandInterface;

$waiterName = 'BucketExists';
$waiterOptions = ['Bucket' => 'amzn-s3-demo-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();
```

公开基于 Promise 的 Waiter API 适用于一些功能强大、开销相对较低的使用情形。例如，如果您希望等待多个资源，并在第一个 Waiter 成功完成后执行某些操作，这时该怎么办呢？

```
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 适用于 PHP 的 Amazon SDK 版本 3 中的表达式

JMESPath使您能够以声明方式指定如何从 JSON 文档中提取元素。依赖于 jmespath.php 来支持某些高级抽象，例如版本 3 中的 Paginators 和 适用于 PHP 的 Amazon SDK 版本 3 中的 Waiters，但也公开 JMESPath 了 适用于 PHP 的 Amazon SDK对和的搜索。 适用于 PHP 的 Amazon SDK `Aws\ResultInterface` `Aws\ResultPaginator`

你可以尝试在线JMESPath 示例， JMESPath 在浏览器中试一试。您可以在JMESPath 规范中了解有关该语言的更多信息，包括可用的表达式和函数。

支Amazon CLI撑 JMESPath。针对 CLI 输出编写的表达式与针对 适用于 PHP 的 Amazon SDK编写的表达式完全兼容。

从结果中提取数据

该`Aws\ResultInterface`接口具有一种基于 JMESPath 表达式从结果模型中提取数据的`search($expression)`方法。使用 JMESPath 表达式查询结果对象中的数据有助于删除样板条件代码，并更简洁地表达正在提取的数据。

为演示其工作方式，我们来看下面的默认 JSON 输出，它描述了连接到不同 Amazon EC2 实例的两个 Amazon Elastic Block Store (Amazon EBS) 卷。

```
$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"
        }
    }
}
```


首先，我们可以利用以下命令，只从“卷”列表中检索第一个卷。

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

现在，我们使用 wildcard-index 表达式 [*] 迭代整个列表，提取三个元素并重命名：将 VolumeId 重命名为 ID，将 AvailabilityZone 重命名为 AZ，并将 Size 保留为 Size。我们可以将 multi-hash 表达式置于 wildcard-index 表达式之后，提取并重命名这些元素。

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

这样我们就有了如下的 PHP 数据：

```
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)
  }
}
```

在 multi-hash 表示法中，您还可以使用串联的键（例如 key1.key2[0].key3）提取深度嵌套在结构中的元素。以下示例利用 Attachments[0].InstanceId 键演示此功能，别名指定为简单的 InstanceId。（在大多数情况下， JMESPath 表达式将忽略空格。）

```
$expr = 'Volumes[*].{ID: VolumeId,
                     InstanceId: Attachments[0].InstanceId,
                     AZ: AvailabilityZone,
                     Size: Size}';

$data = $result->search($expr);
var_dump($data);
```

上一表达式会输出以下数据：

```
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)
  }
}
```

您还可以利用 `multi-list` 表达式筛选多个元素：`[key1, key2]`。此表达式会将每个对象筛选出的所有属性放入单个排序列表中，不考虑类型。

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

运行上一搜索会生成以下数据：

```
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)
  }
}
```

使用 `filter` 表达式按特定字段的值筛选结果。以下示例查询仅输出 `us-west-2a` 可用区中的卷。

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

JMESPath 还支持函数表达式。假设您要运行与上述相同的查询，但要检索卷位于以 “us-” 开头的 Amazon 区域中的所有卷。以下表达式使用 `starts_with` 函数，传递 `us-` 文本字符串。然后将此函数的结果与 JSON 文本值 `true` 进行比较，仅传递通过筛选投影返回 `true` 的筛选断言结果。

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

从分页工具提取数据

正如您可以通过 适用于 PHP 的 Amazon SDK 版本 3 中的 Paginator 指南了解到的，`Aws\ResultPaginator` 对象可用于从可分页的 API 操作生成结果。 适用于 PHP 的 Amazon SDK 使您可以从`Aws\ResultPaginator`对象中提取和迭代过滤后的数据，本质上是在迭代器上实现平面映射，其中 JMESPath 表达式的结果是映射函数。

假设您希望创建 `iterator`，以仅从存储桶生成大于 1 MB 的对象。要做到这一点，应首先创建 `ListObjects` 分页工具，然后为该分页工具应用 `search()` 函数，针对分页数据创建扁平化映射的迭代器。

```
$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);
}
```

从适用于 PHP 的 Amazon SDK 版本 3 调用 Amazon Web Services 服务

以下各部分包含示例、教程、任务和指南，向您展示如何使用 适用于 PHP 的 Amazon SDK 来利用 Amazon 服务。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 的功能和选项
- 带有适用于 适用于 PHP 的 Amazon SDK 的指导的代码示例

使用 适用于 PHP 的 Amazon SDK 版本 3 的功能和选项

适用于 PHP 的 Amazon SDK 版本 3 支持其他功能和可使用的选项 Amazon Web Services 服务 APIs。本主题中的各部分按服务介绍了这些选项。

主题

- 在版本 3 中使用 DynamoDB 会话处理程序 适用于 PHP 的 Amazon SDK
- 适用于 PHP 的 Amazon SDK 版本 3 中的 Amazon S3 功能和选项

在版本 3 中使用 DynamoDB 会话处理程序 适用于 PHP 的 Amazon SDK

DynamoDB 会话处理程序是一个适用于 PHP 的自定义会话处理程序，让开发人员能够将 Amazon DynamoDB 用作会话存储。使用 DynamoDB 进行会话存储可将会话移离本地文件系统并将其移入共享位置，从而可减少在分布式 Web 应用程序中进行会话处理时出现的问题。DynamoDB 速度快、可扩展且易于部署，还能够自动处理数据复制。

DynamoDB 会话处理程序使用 `session_set_save_handler()` 函数将 DynamoDB 操作挂接到 PHP 的本机会话函数中，以允许简易替代。其中包括对会话锁定和垃圾回收等功能的支持，这些功能是 PHP 的默认会话处理程序的一部分。

有关 DynamoDB 服务的更多信息，请参阅 Amazon DynamoDB 主页。

基本用法

步骤 1：注册处理程序

首先，实例化并注册会话处理程序。

```
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();
```

步骤 2：创建用于存储会话的表

在实际使用会话处理程序之前，您需要创建一个用于存储会话的表。通过使用适用于 Amazon DynamoDB 的Amazon 控制台或使用 适用于 PHP 的 Amazon SDK，您可以提前完成此操作。

在创建此表时，请使用“id”作为主键的名称。此外，建议使用“expires”属性设置“生存时间”属性以便从会话自动垃圾回收功能中受益。

步骤 3：像平常一样使用 PHP 会话

一旦注册会话处理程序并且已存在表，您便可以像通常使用 PHP 的默认会话处理程序那样使用 $_SESSION 超级全局变量向会话读写内容。DynamoDB 会话处理程序会封装并提取与 DynamoDB 的交互，让您可以轻松地使用 PHP 的本机会话函数和接口。

```
// 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();
```

配置

您可以使用以下选项配置会话处理程序的行为。所有选项都是可选的，但务必要了解默认值为何值。

table_name

存储会话的 DynamoDB 表的名称。默认值为 'sessions'。

hash_key

DynamoDB 会话表中哈希键的名称。默认值为 'id'。

data_attribute

DynamoDB 会话表中的属性名称，该会话表中存储了会话数据。默认值为 'data'。

data_attribute_type

DynamoDB 会话表中属性的类型，该会话表中存储了会话数据。此项默认为 'string'，不过可以选择设置为 'binary'。

session_lifetime

非活动会话在被垃圾回收之前的生命周期。如果未提供，将使用的实际生命周期值为 ini_get('session.gc_maxlifetime')。

session_lifetime_attribute

DynamoDB 会话表中属性的名称，该会话表中存储了会话过期时间。默认值为 'expires'。

consistent_read

会话处理程序是否应对 GetItem 操作使用一致性读取。默认值为 true。

locking

是否使用会话锁定。默认值为 false。

batch_config

用于在垃圾回收期间进行批量删除的配置。这些选项直接传递到 DynamoDB 对象 WriteRequestBatch中。通过 SessionHandler::garbageCollect() 手动触发垃圾回收。

max_lock_wait_time

会话处理程序在放弃之前应等待获取锁定的最长时间（以秒为单位）。默认值为 10 且只用于会话锁定。

min_lock_retry_microtime

会话处理程序在两次获取锁定的尝试之间应等待的最短时间（以微秒为单位）。默认值为 10000 且只用于会话锁定。

max_lock_retry_microtime

会话处理程序在两次获取锁定的尝试之间应等待的最长时间（以微秒为单位）。默认值为 50000 且只用于会话锁定。

要配置会话处理程序，请在实例化处理程序时指定配置选项。以下代码是已指定所有配置选项的示例。

```
$sessionHandler = SessionHandler::fromClient($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,
]);
```

定价

除了数据存储和数据传输费用外，与使用 DynamoDB 相关的成本是根据表的预置的吞吐量容量来计算的（请参阅 Amazon DynamoDB 定价详细信息）。吞吐量用写入容量和读取容量单位来测量。Amazon DynamoDB 主页显示以下内容：

一个读取容量单位表示对大小为 4 KB 的项目每秒执行一次强一致性读取（或每秒执行两次最终一致性读取）。一个写入容量单位表示对大小为 1 KB 的项目每秒执行一次写入。

最后，吞吐量和会话表所需的成本将与您的预期流量和会话大小关联。下表说明了针对每个会话函数，对 DynamoDB 表执行的读取和写入操作量。

通过 `session_start()` 读取	• 1 次读取操作（如果 `consistent_read` 为 `false`，则仅为 0.5 次读取操作）。 • （条件性）如果已过期，则为用于删除会话的 1 次写入操作。

通过 session_start() 读取（使用会话锁定）	• 至少 1 次写入 操作。 • （条件性）获取会话锁定时每次尝试的附加写入操作。基于配置的锁定等待时间和重试选项。 • （条件性）如果已过期，则为用于删除会话的 1 次写入操作。
通过 session_write_close() 写入	• 1 次写入操作。
通过 session_destroy() 删除	• 1 次写入操作。
垃圾回收	• 表中每 4 KB 数据 0.5 次读取操作，用于扫描过期的会话。 • 每个过期项目 1 次写入操作，用于删除该项目。

会话锁定

DynamoDB 会话处理程序支持保守的会话锁定，以模拟 PHP 的默认会话处理程序的行为。默认情况下，DynamoDB 会话处理程序会关闭此功能，因为它会成为性能瓶颈并增加成本，尤其是在应用程序使用 Ajax 请求或 iframe 来访问会话时。在启用该功能之前请认真考虑您的应用程序是否需要会话锁定。

要启用会话锁定，请在实例化 'locking' 时将 true 选项设置为 SessionHandler。

```
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name' => 'sessions',
    'locking'    => true,
]);
```

垃圾回收

在您的 DynamoDB 表中使用属性“expires”设置 TTL 属性。这将自动对您的会话进行垃圾回收，从而无需自行对其进行垃圾回收。

或者，DynamoDB 会话处理程序通过使用一系列 Scan 和 BatchWriteItem 操作来支持会话垃圾回收。考虑到 Scan 操作的工作原理，为了找到所有过期会话并将其删除，垃圾回收过程可能需要大量预配置的吞吐量。

因此，我们不支持自动垃圾回收。更好的做法是将垃圾回收安排在非高峰时段进行，此时，突发吞吐量不会中断应用程序的剩余部分。例如，您可以有一个夜间 cron 作业触发器脚本，用于运行垃圾回收。此脚本需要执行类似如下的内容。

```
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name'   => 'sessions',
    'batch_config' => [
        'batch_size' => 25,
        'before' => function ($command) {
            echo "About to delete a batch of expired sessions.\n";
        }
    ]
]);

$sessionHandler->garbageCollect();
```

您还可以在 `'before'` 内使用 `'batch_config'` 选项，以便在垃圾回收流程所执行的 `BatchWriteItem` 操作之间引入延迟。这将增加完成垃圾回收所需的时间，但可以帮助您分散 DynamoDB 会话处理程序发出的请求，从而帮助您在垃圾回收期间接近预置的吞吐能力或保持在该容量内。

```
$sessionHandler = SessionHandler::fromClient($dynamoDb, [
    'table_name'   => 'sessions',
    'batch_config' => [
        'before' => function ($command) {
            $command['@http']['delay'] = 5000;
        }
    ]
]);

$sessionHandler->garbageCollect();
```

最佳实践

1. 在地理位置最接近应用程序 Amazon 服务器或与应用程序服务器位于同一区域的区域中创建会话表。这可以确保您的应用程序与 DynamoDB 数据库之间的延迟最低。
2. 认真选择您的会话表的预配的吞吐能力。考虑您的应用程序的预期流量和会话的预期大小。或者，对您的桌子使用 “按需” Read/Write 容量模式。
3. 通过 Amazon 管理控制台或 Amazon 监控您消耗的吞吐量 CloudWatch，并根据需要调整吞吐量设置以满足应用程序的需求。

4. 使会话尽量小（最好小于 1 KB）。小型会话执行起来性能更好，需要的预置的吞吐能力更少。
5. 除非您的应用程序需要，否则请勿使用会话锁定。
6. 不使用 PHP 的内置会话垃圾回收触发器，而通过 cron 作业或其他计划机制将您的垃圾回收安排在非高峰时段运行。使用 `'batch_config'` 选项对您有利。

所需的 IAM 权限

要使用 Dynam SessionHhandler oDB，您配置的凭证必须有权使用您在上一步中创建的 DynamoDB 表。下面的 IAM policy 包含您需要的最低权限。要使用此策略，请将资源值替换为您之前创建的表的 Amazon 资源名称 (ARN)。有关创建和附加 IAM 策略的详细信息，请参阅 IAM 用户指南中的管理 IAM 策略。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SessionHandler",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/table-
name"
        }
    ]
}
```

适用于 PHP 的 Amazon SDK 版本 3 中的 Amazon S3 功能和选项

本主题讨论 适用于 PHP 的 Amazon SDK 版本 3 提供的与 Amazon S3 配合使用的其他功能和选项。

主题

适用于 PHP 的 Amazon SDK 版本 3 中的 Amazon S3 多区域客户端

适用于 PHP 的 Amazon SDK 版本 3 提供了一个通用的多区域客户端，可与任何服务一起使用。这使用户能够通过向任何命令提供`@region`输入参数来指定向哪个 Amazon 区域发送命令。此外，SDK 为 Amazon S3 提供的多区域客户端可针对特定的 S3 错误智能响应，并据此重新路由命令。它支持用户使用同一客户端与多个区域通信。对于使用适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 Stream Wrapper（存储桶位于多个区域）的用户来说，这是一项特别有用的功能。

基本用法

无论使用标准 S3 客户端还是多区域客户端，Amazon S3 客户端的基本用法模式都是相同的。命令级别的唯一用法区别是，可以使用`@region`输入参数指定 Amazon 区域。

```
// 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**
>
> 如果使用多区域 Amazon S3 客户端，您不会遇到永久性的重定向异常。如果命令发送到错误的区域，标准 Amazon S3 客户端会抛出 `Aws\S3\Exception\PermanentRedirectException` 的实例。而多区域客户端会将该命令分派到正确的区域。

存储桶区域缓存

Amazon S3 多区域客户端维护给定存储桶所在 Amazon 区域的内部缓存。默认情况下，每个客户端都有它自己的内存中的缓存。要在客户端或进程之间共享缓存，请提供 `Aws\CacheInterface` 的一个实例，作为多区域客户端的 `bucket_region_cache` 选项。

```
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),
    ],
]);
```

适用于 PHP 的 Amazon SDK 版本 3 中的亚马逊 S3 流包装器

Amazon S3 Stream Wrapper 支持您使用内置 PHP 函数从 Amazon S3 存储和检索数据，这些函数包括 `file_get_contents`、`fopen`、`copy`、`rename`、`unlink`、`mkdir` 和 `rmdir`。

您需要注册 Amazon S3 Stream Wrapper 才可使用。

```
$client = new Aws\S3\S3Client([/** options **/]);

// Register the stream wrapper from an S3Client object
$client->registerStreamWrapper();
```

这样您就可以使用 `s3://` 协议来访问 Amazon S3 中存储的存储桶和对象。Amazon S3 Stream Wrapper 接受包含存储桶名称的字符串，后跟正斜杠和可选的对象键或前缀：`s3://<bucket>[/<key-or-prefix>]`。

> **Note**
>
> Stream Wrapper 可处理您至少拥有读取权限的对象和存储桶。这就意味着，您的用户应有权针对任何需要交互的存储桶执行 `ListBucket`，并针对任何需要交互的对象执行 `GetObject`。对于不具有此级别权限的使用情形，建议您直接使用 Amazon S3 客户端操作。

下载数据

您可以使用 `file_get_contents` 获取对象内容。但使用此功能要小心，它会将对象的全部内容载入内存。

```
// Download the body of the "key" object in the "bucket" bucket
$data = file_get_contents('s3://bucket/key');
```

如果处理较大文件或需要从 Amazon S3 流式传输数据，请使用 `fopen()`。

```
// 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**
>
> 只有调用 `fflush` 才会返回文件写入错误。如果调用了未刷新的 `fclose`，不会返回这些错误。如果 `fclose` 关闭了流，它的返回值将为 `true`；无论其内部 `fflush` 是否会响应任何错误。如果调用 `file_put_contents` 也不会返回这些错误，这是由于 PHP 的实施方式导致的。

打开可搜寻的流

以“r”模式打开的流仅允许读取流中的数据，默认情况下不可搜寻。这样数据才能够真正以流式处理的方式从 Amazon S3 下载，之前读取的字节不需要缓冲到内存中。如果需要对流进行搜寻，可以将 `seekable` 传递到函数的流上下文选项。

```
$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);
}
```

“打开可搜寻的流”支持您搜寻之前读取的字节。您不可跳转到尚未从远程服务器读取的字节。要允许重新调用之前读取的数据，需使用流装饰器将数据缓冲到 PHP 临时流中。如果缓存数据量超过 2 MB，临时流中的数据会从内存传输到磁盘中。在使用 `seekable` 流上下文设置从 Amazon S3 下载大文件时，请留意这一点。

上传数据

您可以使用 `file_put_contents()` 来将数据上传到 Amazon S3。

```
file_put_contents('s3://bucket/key', 'Hello!');
```

您可以结合使用 `fopen()` 和“w”、“x”或“a”流访问模式来流式传输数据，从而上传较大的文件。Amazon S3 Stream Wrapper 不支持同时读取和写入流（例如“r+”、“w+”等）。这是因为 HTTP 协议不允许同时读取和写入。

```
$stream = fopen('s3://bucket/key', 'w');
fwrite($stream, 'Hello!');
fclose($stream);
```

> **Note**
>
> Amazon S3 需要在发送请求负载之前指定内容长度标题。因此，在 `PutObject` 操作中上传的数据会使用 PHP 临时流进行内部缓冲，直到流刷新或关闭。

> **Note**
>
> 只有调用 `fflush` 才会返回文件写入错误。如果调用了未刷新的 `fclose`，不会返回这些错误。如果 `fclose` 关闭了流，它的返回值将为 `true`；无论其内部 `fflush` 是否会响应任何错误。如果调用 `file_put_contents` 也不会返回这些错误，这是由于 PHP 的实施方式导致的。

fopen 模式

PHP 的 fopen() 函数需要您指定 `$mode` 选项。mode 选项指定数据是否可在流中进行读写，以及打开流时文件是否必须存在。

对于以 Amazon S3 对象为目标的流，Amazon S3 Stream Wrapper 支持以下模式。

r	只读流，对象必须已存在。
w	只可写入的流。如果对象已存在，则将覆盖该对象。
a	只可写入的流。如果对象已存在，则会将它下载到临时流中，并且对流的任何写入操作将追加到之前上传的数据后面。
x	只可写入的流。如果对象已存在，将引发错误。

其他对象函数

Stream Wrapper 允许多种不同的内置 PHP 函数与 Amazon S3 这样的自定义系统配合使用。Amazon S3 Stream Wrapper 允许您针对存储在 Amazon S3 中的对象执行以下函数。

unlink()	从存储桶中删除一个对象。

```
// Delete an object from a bucket
unlink('s3://bucket/key');
```

您可以传递 DeleteObject 操作的任何可用选项，修改对象的删除方式（例如指定具体的对象版本）。

```
// Delete a specific version of an
 object from a bucket
unlink('s3://bucket/key', stream_co
ntext_create([
    's3' => ['VersionId' => '123']
]);
```

filesize()	获取对象的大小。
	
	`// Get the Content-Length of an object`
	`$size = filesize('s3://bucket/`
	`key', );`

is_file()	检查 URL 是否为文件。
	
	`if (is_file('s3://bucket/key')) {`
	`    echo 'It is a file!';`
	`}`

file_exists()	检查某个对象是否存在。
	
	`if (file_exists('s3://bucket/key'))`
	` {`
	`    echo 'It exists!';`
	`}`

filetype()	检查 URL 是否对应于文件或存储桶 (dir)。
file()	将对象内容加载到一些行中。您可以传递 GetObject 操作的任何可用选项，修改文件的下载方式。

filemtime()	获取对象的最新修改日期。
rename()	复制对象并删除原始版本，从而对对象重命名。您可以将 CopyObject 和 DeleteObject 操作的可用选项传递到流上下文参数，修改复制和删除对象的方式。

> **Note**
>
> 虽然 copy 一般与 Amazon S3 Stream Wrapper 配合使用，由于 PHP 中 copy 函数的内部原因，某些错误可能不会正确报告。我们建议您改用 awss3 ObjectCopier 的实例。

使用存储桶和文件夹

使用 **mkdir()** 来使用存储桶

您可以创建和浏览 Amazon S3 存储桶，这与 PHP 允许您在文件系统上创建和遍历目录的方式类似。

以下是创建存储桶的示例。

```
mkdir('s3://amzn-s3-demo-bucket');
```

> **Note**
>
> 2023 年 4 月，Amazon S3 自动启用了 S3 屏蔽公共访问权限，并对所有新创建的存储桶禁用了访问控制列表。此更改还会影响StreamWrapper的mkdir函数如何使用权限和 ACLs. 更多信息可参见这篇新增内容一 Amazon 文。

您可以使用CreateBucket操作可用的参数将流上下文选项传递给该mkdir()方法，以修改存储桶的创建方式。

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

```
        ]));
```

您可以使用 `rmdir()` 函数删除存储桶。

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

> **Note**
>
> 只有空存储桶才可删除。

使用 `mkdir()` 来使用文件夹

创建存储桶后，您可以使用 `mkdir()` 来创建对象，这些对象可用作文件夹，就像在文件系统中一样。

以下代码段将名为“my-folder”的文件夹对象添加到名为“amzn-s3-demo-bucket”的现有存储桶中。使用正斜杠 (/) 字符将文件夹对象名称与存储桶名称和任何其他文件夹名称分开。

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

关于 2023 年 4 月之后权限变更的上一个注释也会在您创建文件夹对象时发挥作用。这篇博客文章包含有关如何在需要时调整权限的信息。

使用 `rmdir()` 函数删除空文件夹对象，如下面的代码段所示。

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

列出存储桶的内容。

可将 opendir()、readdir()、rewinddir() 和 closedir() PHP 函数与 Amazon S3 Stream Wrapper 配合使用，遍历存储桶的内容。您可以将ListObjects操作可用的参数作为自定义流上下文选项传递给`opendir()`函数，以修改对象的列出方式。

```
$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);
}
```

您可以使用 PHP 递归列出存储桶中的每个对象和前缀。RecursiveDirectoryIterator

```
$dir = 's3://bucket';
$iterator = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($dir));

foreach ($iterator as $file) {
    echo $file->getType() . ': ' . $file . "\n";
}
```

还可以使用 `Aws\recursive_dir_iterator($path, $context = null)` 函数以递归方式列出存储桶内容，这样可产生较少的 HTTP 请求。

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

$iter = Aws\recursive_dir_iterator('s3://bucket/key');
foreach ($iter as $filename) {
    echo $filename . "\n";
}
```

流上下文选项

您可以通过传递自定义流上下文选项，自定义 Stream Wrapper 使用的客户端，或用于缓存之前加载的存储桶和键信息的缓存。

Stream Wrapper 针对每个操作支持以下流上下文选项。

`client`

`Aws\AwsClientInterface` 对象，用于执行命令。

`cache`

`Aws\CacheInterface` 的实例可用于缓存之前获得的文件统计数据。默认情况下，Stream Wrapper 使用内存中 LRU 缓存。

传输文件和目录

适用于 PHP 的 Amazon SDK 版本 3 提供了两种在 Amazon S3 之间传输文件和目录的方法。这两种解决方案都能处理大文件分段上传和下载的复杂性，但它们的设计理念、功能集和使用模式各不相同。

转账选项概述

选择最适合您的应用程序需求的传输方法：

S3 传输管理器（推荐）

一个现代化的高级库，为文件传输提供全面的解决方案。它提供了广泛的配置选项、内置的进度跟踪、自定义下载处理程序和强大的错误处理功能。S3 传输管理器使用基于 Promise 的 API，通过高级筛选功能支持单个文件操作和目录传输。

转移

一种专门针对批量目录操作的目录传输实现。它为上传和下载整个目录提供了一个更简单的 API，其中包含基本的配置选项。与 S3 传输管理器相比，这种方法的功能较少。

主要区别

下表突出显示了两种传输方法之间的主要区别：

功能	S3 Transfer Manager	Transfer
单个文件操作	是（上传/下载单个文件）	否（仅限目录操作）
目录操作	是（使用高级筛选）	是（基本目录传输）
进度跟踪	内置自定义监听器	有限（仅限调试输出）
自定义下载处理程序	是	否
校验和验证	自动，带配置	手动（add_content_md5 选项）
错误处理	全面的故障策略	基于承诺的基本处理
配置选项	广泛（8 个以上的选项）	基本（6 个选项）
API 设计	请求/响应对象	简单的构造器参数

选择正确的方法

需要时使用 S3 传输管理器：

- 个人文件上传或下载操作
- 高级进度跟踪和监控
- 用于专门处理的自定义下载处理程序
- 全面的错误处理和重试政策
- 对多部分操作的精细控制
- 具有复杂筛选逻辑的目录操作

需要时使用 “转移”：

- 简单的 “目录到/从 S3” 传输
- 最少的配置和设置
- 与使用 Transfer 的现有代码的兼容性
- 基本的分段上传功能

> **Note**
>
> 对于新应用程序，我们建议使用 S3 传输管理器，因为它为文件传输提供了更全面、更灵活的解决方案。

S3 Transfer Manager

S3 传输管理器提供了一个界面，用于向 Amazon S3 上传和下载文件。您可以将其用于单文件操作或目录操作。

传输管理器会自动处理分段上传和下载。它可以管理大文件并跟踪进度。该库实施了 S3 数据传输的最佳实践。您可以使用它在 PHP 应用程序中构建文件传输功能。

主要 功能

S3 传输管理器提供以下主要功能：

- 简单 API：上传和下载文件和目录

- 自定义下载处理程序：实现自己的下载逻辑
- 自动分段上传和下载：自动处理大文件
- 并行处理：最大限度提高吞吐量
- 进度跟踪：监控传输状态
- 可自定义的行为：配置大量选项
- 错误处理：配置重试和失败策略
- 目录操作：批量传输多个文件
- 校验和验证：确保数据完整性

安装

S3 传输管理器包含在 适用于 PHP 的 Amazon SDK 版本 3 中。您无需单独安装。

要使用 Composer 进行安装，请运行以下命令：

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

此命令安装完整的 适用于 PHP 的 Amazon SDK 版本 3，包括 S3 传输管理器。

基本用法

以下示例说明如何使用 S3 传输管理器：

```
<?php

use Aws\S3\S3Client;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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


// Create an S3 client.
$s3Client = new S3Client([
    'version' => 'latest',
    'region'  => 'us-west-2',
]);

// Create a transfer manager with default configuration.
```

```
$transferManager = new S3TransferManager($s3Client);

// Alternative: Create transfer manager with null client. S3 Transfer Manager uses a
 default S3 client.
$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

// Example: Upload a file.
$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/local/file.txt',
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'path/to/s3/file.txt',
        ]
    )
);

// Wait for the upload to complete.
$result = $uploadPromise->wait();

echo "Upload complete!\n";
```

Important

使用 S3 传输管理器创建默认 Amazon S3 客户端时，客户可以使用传输管理器config选项中的default_region参数为客户端传递默认区域，否则 Amazon S3 客户端会尝试使用默认行为来解析配置，如果未解析该区域，则会引发异常。

创建转接管理器

您可以通过两种方式创建转接管理器：

使用现有的 S3 客户端

将现有S3Client实例传递给S3TransferManager*<add link>*构造函数。

```
<?php

use Aws\S3\S3Client;
```

```
use Aws\S3\S3Transfer\S3TransferManager;

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

// Create an S3 client.
$s3Client = new S3Client([
    'version' => 'latest',
    'region'  => 'us-west-2',
]);

$transferManager = new S3TransferManager($s3Client);
```

使用默认创建 S3 客户端

以客户端null身份传递并指定配置选项。

```
<?php

use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);
```

配置

S3 传输管理器接受配置选项以自定义其行为。在创建转移管理器的实例时，请提供这些选项。配置参数可以是数组，也可以是的实例S3TransferManagerConfig*<add link>*。

以下示例配置 S3 传输管理器实例：

```
<?php

use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(
    null,
    [
```

```
        // 10MB parts for multipart operations.
        'target_part_size_bytes' => 10 * 1024 * 1024,

        // Use multipart upload for files larger than 20MB.
        'multipart_upload_threshold_bytes' => 20 * 1024 * 1024,

        // Enable checksum calculation for data integrity.
        'request_checksum_calculation' => 'when_supported',

        // Enable checksum validation when getting objects.
        'response_checksum_validation' => 'when_supported',

        // Use part-based multipart downloads.
        'multipart_download_type' => 'part',

        // Allow up to 10 concurrent operations.
        'concurrency' => 10,

        // Enable progress tracking.
        'track_progress' => true,

        // Set default region for default S3 client construction.
        'default_region' => 'us-west-2',
    ]
);
```

> **Note**
>
> 当您将配置作为数组提供给时`S3TransferManager`，SDK 会在内部调用`S3TransferManagerConfig::fromArray`以将其转换为正确的类型。

配置选项

所有配置选项都是可选的，如果未指定，则使用默认值。

Option	Type	默认值	说明
`target_part_size_bytes`	int	8MB	分段上传/下载的最小分段大小。

Option	Type	默认值	说明
multipart_upload_threshold_bytes	int	16 MB	使用分段上传的文件大小阈值。
request_checksum_calculation	字符串	'when_supported'	启用校验和计算。有效值为 “支持时”、“需要时”。
response_checksum_validation	字符串	'when_supported'	在获取对象时启用校验和验证。有效值为 “支持时”、“需要时”。
multipart_download_type	字符串	'部分'	下载大文件策略。有效值为 “部分”（分段下载）、“范围”（范围请求）。
concurrency	int	5	最大并发操作数。
track_progress	布尔	FALSE	是否跟踪传输进度。
default_region	字符串	'us-east-1'	Amazon Web Services 区域 如果未提供 S3 客户端，则使用。

文件操作

S3 传输管理器提供了上传和下载单个文件的方法。

上传本地文件

要将文件上传到 Amazon S3，请使用upload*<add link>*方法。

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;
```

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

$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

// Source can be from a local path to a file.
$source = '/path/to/local/file.txt';
// Or the source can be an instance of StreamInterface.
$source = GuzzleHttp\Psr7\Utils::streamFor('Hello World!');

$uploadPromise = $transferManager->upload(
    new UploadRequest(
        $source,
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'path/to/s3/file.txt',
            // Additional `putObject` parameters as needed.
        ],
        [
            // Optional configuration overrides.
            'multipart_upload_threshold_bytes' => 100 * 1024 * 1024, // 100MB
            'target_part_size_bytes'           => 10 * 1024 * 1024, // 10MB
            'track_progress'                   => true,
            'request_checksum_calculation'     => 'when_required',
        ]
    )
);

// The upload is asynchronous, you can wait for it to complete.
$result = $uploadPromise->wait();

// Or you can use the promise for more complex workflows.
$result = $uploadPromise->then(
    function ($result) {
        echo "Upload succeeded!";
    },
    function ($error) {
        echo "Upload failed: " . $error->getMessage();
    }
)->wait();
```

`upload` 方法参数

该upload方法接受的实例UploadRequest*<add link>*作为参数。

`UploadRequest` 参数

参数	Type	必需	说明
$source	字符串| StreamInterface	是	包含要上传的数据的本地文件路径或流的路径。
$uploadRequestArgs	array	是	上传请求参数（必须包含存储桶和密钥）。
$config	array	否	特定于此上传的配置替换。有关配置选项的更多信息，请参阅以下部分。
$listeners	array	否	用于监视此上传的TransferListener 对象数组。
$progressTracker	TransferListener	否	此次上传的进度跟踪器。

`$config`的选项 `UploadRequest`

SDK 解析 S3 传输管理器配置中的默认$config值。

Option	Type	必需	说明
multipart_upload_threshold_bytes	int	否	覆盖触发分段上传时的默认阈值。
target_part_size_bytes	int	否	覆盖默认的目标段大小（以字节为单位）。

Option	Type	必需	说明
`track_progress`	布尔	否	覆盖默认选项以启用进度跟踪。如果此选项是，`true`而您未提供`progressTracker` 参数，则 SDK 将使用默认实现。
`concurrency`	int	否	覆盖并发的默认值。
`request_checksum_calculation`	字符串	否	覆盖是否必须执行请求校验和计算的默认值。

当该`upload`方法成功运行时，它将返回`UploadResult`*<add link>*。

下载 S3 对象

要从 Amazon S3 下载对象，请使用`download`*<add link>*方法。

```
<?php

use Aws\S3\S3Transfer\Models\DownloadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

// Download using an S3 URI.
$downloadPromise = $transferManager->download(
    new DownloadRequest(
        's3://amzn-s3-demo-bucket/path/to/s3/file.txt',
        [
            // Additional `getObject` parameters as needed.
        ],
        [
            // Optional configuration overrides.
            'response_checksum_validation' => 'when_required',
            'track_progress'               => true,
```

```
            'target_part_size_bytes'       => 10 * 1024 * 1024, // 10MB
        ]
    )
);

// Wait for the download to complete.
$result = $downloadPromise->wait();

// The SDK uses a stream-based download handler by default.
$stream = $result->getDownloadDataResult();

// You can either get the content.
$content = $stream->getContents();

// Or write the stream to a file.
file_put_contents('/path/to/local/file.txt', $stream);

// Don't forget to close the stream.
$stream->close();
```

download 方法参数

该download方法接受的实例DownloadRequest*<add link>*作为参数。

DownloadRequest 参数

参数	Type	必需	说明
$source	字符串|数组|null	否	要从 S3 下载的对象。您可以将此参数作为 S3 URI 字符串 (" s3://amzn-s3-demo-bucket/key ")、包含存储桶和密钥密钥的数组或null。当此值为时null，在$downloadRequestArgs 参数中传递存储桶和密钥值。
$downloadRequestArgs	array	否	其他下载对象请求参数。如果$source是null，请在此处提供存储桶和密钥值。

参数	Type	必需	说明
`$config`	array	否	此下载专用的配置替换。有关配置选项的更多信息，请参阅以下部分。
`$downloadHandler`	`AbstractDownloadHandler <add link>`|null	否	接收每个对象块并管理其下载的处理程序。 该download方法的默认处理程序使用基于流的处理程序。此处理程序将每个对象块推送到一个流中。 该downloadFile 方法的默认处理程序（参见the section called “将 S3 对象下载到本地文件”）使用基于文件的处理程序。该处理程序将每个区块写入一个文件。 您可以自己实现`DownloadHandler` ，以自定义 SDK 存储下载数据的位置和方式。例如，您可以将数据存储在数据库、云存储或自定义处理管道中，而不是文件或流中。
`$progressTracker`	`TransferListener`	否	此下载的进度跟踪器。
`$listeners`	`array`	否	用于监视此下载的`AbstractTransferListener` 对象数组。

`$config`的选项 `DownloadRequest`

SDK 解析 S3 传输管理器配置中的默认`$config`值。

Option	Type	必需	描述
multipart_download_type	字符串	否	覆盖默认的分段下载类型。有效值为“部分”、“远程”
response_checksum_validation	字符串	否	覆盖传输管理器配置中的解析值，以确定是否应进行校验和验证。只有在中不存在此选项ChecksumMode 时，SDK 才会$getObjectRequestArgs 考虑使用此选项DownloadRequest 。
track_progress	布尔	否	覆盖启用进度跟踪的默认选项。如果此选项是，true而您没有提供progressTracker 参数，则 SDK DownloadRequest, 将使用默认实现。 当$progressTracker 参数为时，使用此选项启用默认进度跟踪器null。
target_part_size_bytes	int	否	在范围分段下载中使用的分段大小（以字节为单位）。如果您未提供此参数，则下载将使用传输管理器上target_part_size_bytes 配置的。

当该download方法成功运行时，它将返回DownloadResult*<add link>*。

将 S3 对象下载到本地文件

要将 S3 对象直接下载到本地文件，请使用以下downloadFile方法：

```
<?php

use Aws\S3\S3Transfer\Models\DownloadFileRequest;
```

```
use Aws\S3\S3Transfer\Models\DownloadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

$downloadFilePromise = $transferManager->downloadFile(
    new DownloadFileRequest(
        '/path/to/local/file.txt',  // Destination file path.
        false,  // Fail when destination exists.
        new DownloadRequest(
            's3://amzn-s3-demo-bucket/path/to/s3/file.txt',
            [],  // `getObject` request args
            [
                'track_progress'         => true,
                'target_part_size_bytes' => 10 * 1024 * 1024, // 10MB parts
            ]
        )
    )
);

// Wait for download to complete.
$result = $downloadFilePromise->wait();

// `getDownloadDataResult()` returns the string for the file path of the downloaded
 content because
//  the default `DownloadHandler` used by `downloadFile()` is a file-based handler.
echo "File downloaded successfully at {$result->getDownloadDataResult()}!\n";
```

downloadFile 方法参数

该downloadFile方法接受的实例DownloadFileRequest作为参数。

DownloadFileRequest 参数

Option	Type	必需	描述
$destination	字符串	是	保存文件的本地路径。

Option	Type	必需	描述
`$failsWhenDestinationExists`	布尔	是	如果目标文件已经存在，则是否失败。如果此选项为`false`且目标文件存在，则 SDK 会删除或覆盖现有文件。
`$downloadRequest`	`DownloadRequest`	是	配置的`DownloadRequest` 对象。有关更多信息，请参阅DownloadRequest 对象。

当该`downloadFile`方法成功运行时，它将返回`DownloadResult`*<add link>*。

目录操作

S3 传输管理器可以处理整个目录传输。

上传目录

S3 传输管理器可以将整个目录上传到 S3 存储桶。 *<Are the key values the filenames when uploaded?>*

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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


$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/local/directory',
        'amzn-s3-demo-bucket',
        [
            // Additional `putObject` parameters that apply to all files.
            'ACL'          => 'public-read',
```

```
            'CacheControl' => 'max-age=3600',
        ],
        [
            // Configuration options.
            'recursive'                     => true,
            'follow_symbolic_links'         => false,
            's3_prefix'                     => 'uploads/2023/',
            's3_delimiter'                  => '/',
            'track_progress'                => true,
            'filter'                        => function ($file) {
                // Upload only .jpg files.
                return pathinfo($file, PATHINFO_EXTENSION) === 'jpg';
            },
            'upload_object_request_modifier' => function ($args) {
                // Customize request arguments for each file.
                $args['ContentType'] = 'image/jpeg';
            },
        ]
    )
);

// Wait for the upload process to complete.
$result = $uploadDirPromise->wait();

$uploaded = $result->getObjectsUploaded();
$failed = $result->getObjectsFailed();
echo "Uploaded {$uploaded} files. Failed: {$failed}\n";
```

uploadDirectory 方法参数

该uploadDirectory方法接受的实例UploadDirectoryRequest*<add link>*作为参数。

UploadDirectoryRequest 参数

参数	Type	必需	描述
$sourceDirectory	字符串	是	要上传的本地目录的路径。
$targetBucket	字符串	是	目标 S3 存储桶名称。

参数	Type	必需	描述
`$uploadRequestArgs`	array	否	所有文件的其他上传对象请求参数。
`$config`	array	否	目录上传的配置选项。有关配置选项的更多信息，请参阅以下部分。
`$listeners`	array	否	`TransferListener` 对象数组。SDK 会对`TransferListener` 每个文件进行克隆。
`$max_depth`	布尔	-1 “运行时默认值”	要表示递归文件的最大深度，请三步走。
`$progressTracker`	`TransferListener`	否	所有上传的进度跟踪器。

`$config`的选项 `UploadDirectoryRequest`

Option	Type	默认值	说明
`recursive`	布尔	`false`	是否递归上传子目录。
`follow_symbolic_links`	布尔	`false`	是否关注符号链接。
`s3_prefix`	字符串	''	要添加到所有对象密钥的前缀。
`s3_delimiter`	字符串	'/'	在 S3 对象键中使用的分隔符。
`track_progress`	布尔	`false`	是否跟踪进度。
`max_concurrency`	int	100	并发上传的最大数量。

Option	Type	默认值	说明
`filter`	可调用	null	筛选要上传哪些文件的函数。
`upload_object_request_modifier`	可调用	null	用于自定义每个上传请求的函数。
`failure_policy`	可调用	null	处理上传失败的函数。

可调用的选项类型 **`$config`**

选项名称	参数名	参数类型	参数信息
`filter`	`$file`	`SplFileInfo` |字符串	如果转换为字符串，则为文件路径。否则，它就是一个实例`SplFileInfo` 。
`upload_object_request_modifier`	`$requestArgs`	array	每个单独上传请求的请求参数。有关数组选项的更多信息，请参阅 PutoBject 方法的参数语法部分。
`failure_policy`	`$uploadRequestArgs`	array	每个单独上传请求的请求参数。有关数组选项的更多信息，请参阅数组选项的 PutoBject 方法的参数语法部分。
	`$uploadDirectoryArgs`	array	包含源目录和上传目录操作的目标存储桶的数组。

选项名称	参数名	参数类型	参数信息
	$reason	Throwable |字符串	包含有关导致失败的原因的信息的异常持有者。
	$uploadDirectoryResult	UploadDirectoryResult	包含成功上传的对象数量和失败的对象数量的对象。

当该uploadDirectory方法成功运行时，它将返回UploadDirectoryResult*<add link>*。

下载目录

您可以从 S3 存储桶下载目录。*<Are the filenames the key values when downloaded? If there is no extension in the key, is there also no extension on the file?>*

```
<?php

use Aws\S3\S3Transfer\Models\DownloadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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


$transferManager = new S3TransferManager(null, [
    'default_region' => 'us-west-2'
]);

$downloadDirPromise = $transferManager->downloadDirectory(
    new DownloadDirectoryRequest(
        'amzn-s3-demo-bucket',
        '/path/to/local/directory',
        [
            // Additional `getObject` parameters that apply to all files.
        ],
        [
            // Configuration options.
            's3_prefix'                         => 'uploads/2023/',
            's3_delimiter'                      => '/',
```

```
            'track_progress'                    => true,
            'filter'                            => function ($key) {
                // Download only .jpg files.
                return pathinfo($key, PATHINFO_EXTENSION) === 'jpg';
            },
            'download_object_request_modifier' => function ($args) {
                // Customize request arguments for each file.
                $args['ResponseContentType'] = 'image/jpeg';
            },
            'list_objects_v2_args'              => [
                'MaxKeys' => 1000,
                'Prefix'  => 'uploads/2023/',
            ],
            'fails_when_destination_exists'    => true,
        ]
    )
);

// Wait for the download process to complete.
$result = $downloadDirPromise->wait();

$downloaded = $result->getObjectsDownloaded();
$failed = $result->getObjectsFailed();
echo "Downloaded {$downloaded} files. Failed: {$failed}\n";
```

downloadDirectory 方法参数

该downloadDirectory方法接受的实例DownloadDirectoryRequest*<add link>*作为参数。

DownloadDirectoryRequest 参数

参数	Type	必需	描述
$sourceBucket	字符串	是	从中下载对象的源 S3 存储桶名称。
$destinationDirectory	字符串	是	要将文件下载到的本地目录。

参数	Type	必需	描述
`$downloadRequestArgs`	array	否	所有文件的其他下载对象请求参数。
`$config`	array	否	目录下载的配置选项。有关配置选项的更多信息，请参阅以下部分。
`$listeners`	array	否	`TransferListener` 对象数组。SDK 会对`TransferListener` 每个文件进行克隆。
`$progressTracker`	`TransferListener`	否	所有下载的进度跟踪器。

`$config`的选项 `DownloadDirectoryRequest`

Option	Type	默认值	说明
`s3_prefix`	字符串	"	筛选前缀（如果不在`list_objects_v2_args` ）。
`track_progress`	布尔	`false`	是否跟踪进度。
`filter`	可调用	`null`	筛选要下载的 S3 对象的函数。
`download_object_request_modifier`	可调用	`null`	自定义每个下载请求的功能。
`list_objects_v2_args`	array	[]	`ListObjectsV2` 操作的参数。此操作获取要下载的对象的元数据。

Option	Type	默认值	说明
`fails_when_destination_exists`	布尔	false	当对象目标文件路径已经存在时是否失败。
`failure_policy`	可调用	null	处理下载失败的函数。
`max_concurrency`	int	100	并发下载的最大数量。
`target_part_size_bytes`	int	变化	当分段下载类型设定范围时，分段下载的分段大小。

可调用的选项类型 **`$config`**

选项名称	参数名	参数类型	参数信息
`filter`	`$objectKey`	字符串	存储桶中的对象密钥名称。
`download_object_request_modifier`	`$requestArgs`	array	每个单独下载请求的请求参数。有关数组选项的更多信息，请参阅 getObject 方法的参数语法部分。
`failure_policy`	`$downloadObjectRequestArgs`	array	每个单独下载请求的请求参数。有关数组选项的更多信息，请参阅 getObject 方法的参数语法部分。

选项名称	参数名	参数类型	参数信息
	`$downloadDirectoryRequest`	array	包含源存储桶和下载目录操作的目标目录的数组。
	`$reason`	`Throwable`	包含有关导致失败的原因的信息的异常持有者。
	`$downloadDirectoryResult`	DownloadDirectoryResult *<add link>*	包含成功下载的对象数量和失败的对象数量的对象。

目录操作的工作原理

本节介绍 S3 传输管理器在目录操作期间如何处理文件路径和对象密钥。

上传路径处理

上传目录时，SDK 会根据文件路径构造 S3 对象密钥：

1. 计算每个文件相对于源目录的路径
2. 如果指定，则在`s3_prefix`值前面加上（自动添加尾部斜杠）
3. 将 OS 目录分隔符替换为`s3_delimiter`（默认`/`）

该`recursive`选项控制是否包含子目录。`false`（默认）时，仅上传顶级文件。当`true`上传子目录中的所有文件时，在对象密钥中保留目录结构。

> **Note**
>
> S3 中每个上传文件的密钥将是提供的 s3 前缀，默认情况下为空，再加上文件从指定源目录开始的相对路径。此外，我们将解析的密钥名称中的目录分隔符替换为提供的 s3 分隔符，默认情况下它将是/。
> 示例 1
>
> ```
> $sourceDirectory = "/opt/my-upload-directory";
> $s3Prefix = "important/";
> ```

```
$s3Delimiter = "/"; // Default value
```

目录结构

```
/opt/my-upload-directory/
-- my-file1.txt
-- my-file-2.txt
-- sub-dir/
---- my-another-file1.txt
---- my-another-file2.txt
```

每个文件的密钥将是：

```
$s3Prefix + $fileRelativePath;
```

```
- important/my-file1.txt
- important/my-file-2.txt
- important/sub-dir/my-another-file1.txt
- important/sub-dir/my-another-file2.txt
```

示例 2

```
$sourceDirectory = "/opt/my-docs";
$s3Prefix = ''; // No provided and it will defaulted to empty string
$s3Delimiter = "/"; // Default value
```

目录结构

```
/opt/my-docs/
-- my-file1.txt
-- my-file-2.txt
-- sub-dir/
---- my-another-file1.txt
---- my-another-file2.txt
```

每个文件的密钥将是：

```
$s3Prefix + $fileRelativePath;
```

```
- my-file1.txt
- my-file-2.txt
- sub-dir/my-another-file1.txt
- sub-dir/my-another-file2.txt
```

Example上传目录路径处理

```
<?php
// Source directory: /home/user/photos/
//   fla/beach.jpg
//   fla/sunset.jpg
//   fla/pool/party.jpg
// With s3_prefix: '2023' and recursive: true
// Results:
//   2023/fla/beach.jpg
//   2023/fla/sunset.jpg
//   2023/fla/pool/party.jpg
```

下载路径处理

下载目录时，SDK 会根据 S3 对象密钥构造本地文件路径：

1. 如果指定，则`s3_prefix`从对象密钥中移除
2. 将 S3 分隔符 (/) 替换为操作系统目录分隔符
3. 将结果追加到目标目录

下载总是递归的。SDK 会检索指定前缀下的所有对象，并根据需要创建子目录。为了安全起见，它会验证路径不会解析到目标目录之外的路径。

Example下载目录路径处理

```
<?php
// S3 objects:
//   2023/fla/beach.jpg
```

```
//   2023/fla/sunset.jpg
//   2023/fla/pool/party.jpg
// With s3_prefix: '2023' and destination: /home/user/downloads
// Results:
//   /home/user/downloads/fla/beach.jpg
//   /home/user/downloads/fla/sunset.jpg
//   /home/user/downloads/fla/pool/party.jpg
```

Note

下载对象的文件路径将按如下方式解析：

- s3 前缀（如果提供）将从对象密钥中删除。
- 如果 s3Delimiter 与操作系统目录分隔符不同，则 s3 分隔符将替换为操作系统目录分隔符。

示例 1

```
$s3Prefix = "my-prefix";
$s3Objects = [
    "my-prefix/file-1.txt",
    "my-prefix/file-2.txt",
    "my-prefix/subdir/file-1.txt"
];
$targetDirectory = "/opt/bucket/downloads/";
```

下载完这些对象后，将按如下方式放置：

```
/opt/bucket/downloads/
-- file-1.txt
-- file-2.txt
-- subdir/
---- file-1.txt
```

如我们所见，该前缀已从对象的最终文件路径中删除。
示例 2-没有 s3 前缀

```
$s3Prefix = ""; // No provided
$s3Objects = [
    "README.md",
    "my-docs/file-1.txt",
```

```
    "my-docs/file-2.txt",
    "my-docs/statements/file-1.txt"
];
$targetDirectory = "/opt/bucket/downloads/";
```

下载完这些对象后，将按如下方式放置：

```
/opt/bucket/downloads/
-- README.md
-- my-docs/
---- file-1.txt
---- file-2.txt
---- statements/
------ file-1.txt
```

转移听众

您可以实现自己的侦听器，用于对不同的传输事件做出反应。这些事件是：

- **transferInitiated**: 当传输启动时
- **bytesTransferred**: 当传输将字节从源传输到目标时
- **transferComplete**: 当传输完成且没有错误时
- **transferFail**: 当传输在完成之前失败时

每个S3TransferManager操作（例如uploadFile和downloadDirectory）都有一个listeners参数。此参数是一个数组，它期望每个项目都是一个实例AbstractTransferListener。

AbstractTransferListener是一个抽象基类，你可以扩展它来监视传输操作。它为所有传输事件提供了空方法实现。您只需要重写要处理的事件的方法即可。

每个AbstractTransferListener方法都接收一个$context参数。此参数包含有关操作的键值信息。上下文包括：

- request_args（数组）-请求参数
- progress_snapshot（的实例 TransferProgressSnapshot*<add link>*）-当前请求状态
- reason（的实例Throwable）-仅为该transferFail方法提供

该progress_snapshot属性还包含故障事件的reason属性。您可以通过$context['reason']或通过访问例外$context['progress_snapshot']#getReason()。

以下代码显示了AbstractTransferListener类的结构：

```
<?php

namespace Aws\S3\S3Transfer\Progress;

abstract class AbstractTransferListener
{
    /**
     * @param array $context
     * - request_args: (array) The request arguments that will be provided
     *   as part of the request initialization.
     * - progress_snapshot: (TransferProgressSnapshot) The transfer snapshot holder.
     *
     * @return void
     */
    public function transferInitiated(array $context): void {}

    /**
     * @param array $context
     * - request_args: (array) The request arguments that will be provided
     *   as part of the operation that originated the bytes transferred event.
     * - progress_snapshot: (TransferProgressSnapshot) The transfer snapshot holder.
     *
     * @return void
     */
    public function bytesTransferred(array $context): bool {}

    /**
     * @param array $context
     * - request_args: (array) The request arguments that will be provided
     *   as part of the operation that originated the bytes transferred event.
     * - progress_snapshot: (TransferProgressSnapshot) The transfer snapshot holder.
     *
     * @return void
     */
    public function transferComplete(array $context): void {}

    /**
     * @param array $context
     * - request_args: (array) The request arguments that will be provided
```

```
     *    as part of the operation that originated the bytes transferred event.
     * - progress_snapshot: (TransferProgressSnapshot) The transfer snapshot holder.
     * - reason: (Throwable) The exception originated by the transfer failure.
     *
     * @return void
     */
    public function transferFail(array $context): void {}
}
```

以下示例显示了一个仅重写transferInitiated和transferFail方法的AbstractTransferListener实现：

```
<?php

namespace MyApp\Listeners;

use Aws\S3\S3Transfer\Progress\AbstractTransferListener;

class MyCustomListener extends AbstractTransferListener
{
    public function transferInitiated(array $context): void
    {
        $progressSnapshot = $context['progress_snapshot'];

        echo "Transfer initiated for object with identifier: "
            . $progressSnapshot->getIdentifier();
    }

    public function transferFail(array $context): void
    {
        $progressSnapshot = $context['progress_snapshot'];
        $reason = $context['reason'];

        echo "Transfer for object with identifier: "
            . $progressSnapshot->getIdentifier() . " has failed.";

        echo "Completion Status: " . $progressSnapshot->getTransferredBytes()
            . "/"
            . $progressSnapshot->getTotalBytes() . " B";

        echo "Reason of failure is: " . $reason;
    }
}
```

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;
use MyApp\Listeners\MyCustomListener;

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


// The following example uses `MyCustomListener` in an `upload` operation.
$transferManager = new S3TransferManager(
    null,
    ['default_region' => 'us-east-2']
);
$uploadPromise = $transferManager->upload(
    new UploadRequest(
        source: "/tmp/myfile.txt",
        uploadRequestArgs: [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key' => 'MyKey'
        ],
        listeners: [
            new MyCustomListener()
        ]
    )
);

$result = $uploadPromise->wait();
```

进度跟踪

S3 传输管理器提供内置的进度跟踪功能。

跟踪所有操作

您可以通过在S3TransferManager实例上设置为来启用对所有操作（文件和目录）track_progresstrue的跟踪。

```
<?php

use Aws\S3\S3Transfer\S3TransferManager;

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

```
$transferManager = new S3TransferManager(
    null,
    [
        'track_progress' => true,
    ]
);
```

通过此设置，SDK 使用内置文件SingleProgressTracker*<add link>*进行文件操作，并使用内置MultiProgressTracker<add link>进行目录操作。

跟踪单个文件操作

如果您未在转账管理器上启用进度跟踪，则可以通过两种方式启用对单个方法的跟踪。

在$config数组true中设置为track_progress，用于特定的文件操作。

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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


$transferManager = new S3TransferManager(
    null, []
);

$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [   // $config array
            'track_progress' => true,
        ],
        []
    )
);
```

创建新SingleProgressTracker实例或提供的自定义实现AbstractTransferListener。将您的跟踪器作为$progressTracker参数传递给文件操作。

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Progress\SingleProgressTracker;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(
    null, []
);

$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/Myfile.txt',  // Or an instance of StreamInterface.
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key' => 'file.txt',
        ],
        [],
        [],
        new SingleProgressTracker()  // Or custom implementation of the
 TransferLister interface.
    )
);
```

以下示例显示了名SingleProgressTracker为的文件的上传进度的内置控制台输出示例MyFile.txt：

```
MyFile.txt:
[##########################                                ] 50% 5242880/10485760 B
```

跟踪单个目录操作

如果您没有在转移管理器上启用进度跟踪，则可以通过两种方式为目录操作启用该功能。

在方法的$config数组参数true中将 “track_progress” 设置为。

```
<?php
```

```
use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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


$transferManager = new S3TransferManager(
    null, []
);

$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [   // $config array
            'track_progress' => true,
        ],
        []
    )
);
```

创建一个新MultiProgressTracker实例并将其作为$progressTracker参数传递给目录操作。

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\Progress\MultiProgressTracker;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(
    null, []
);

$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [],
```

```
        [],
        new MultiProgressTracker()
    )
);
```

以下示例显示了目录上传进MultiProgressTracker度的内置控制台输出示例：

```
MyFile.4.txt:
[##################################################] 100% 4194304/4194304 B
MyFile.5.txt:
[##################################################] 100% 24117248/24117248 B
MyFile.2.txt:
[                                                  ] 0% 0/10485760 B
MyFile.3.txt:
[##################################################] 100% 18874368/18874368 B
MyFile.1.txt:
[##################################################] 100% 1048576/1048576 B
--------------------------------------------------
[##########################################         ] 80% Completed: 3/5, Failed: 0/5
```

最后一行列出了每个文件的进度以及摘要进度。

自定义进度跟踪

您可以实现自己的文件操作进度跟踪。创建一个扩展AbstractTransferListener接口的类，然后根据请求将其传递给方法。

以下实现以文本格式显示单个文件操作的进度信息：

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Progress\AbstractTransferListener;
use Aws\S3\S3Transfer\S3TransferManager;

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

class MyProgressTracker extends AbstractTransferListener
{
    public function transferInitiated(array $context): void
    {
        $key = $context['request_args']['Key'] ?? 'unknown';
        echo "Starting transfer of {$key}...\n";
    }
```

```
    public function bytesTransferred(array $context): bool
    {
        $snapshot = $context['progress_snapshot'];
        $percent = round($snapshot->ratioTransferred() * 100, 2);
        $transferred = round($snapshot->getTransferredBytes() / 1024 / 1024, 2);
        $total = round($snapshot->getTotalBytes() / 1024 / 1024, 2);

        echo "Progress: {$percent}% ({$transferred}MB of {$total}MB)\n";
    }

    public function transferComplete(array $context): void
    {
        $key = $context['request_args']['Key'] ?? 'unknown';
        echo "Transfer complete for {$key}!\n";
    }

    public function transferFail(array $context): void
    {
        $key = $context['request_args']['Key'] ?? 'unknown';
        $reason = $context['reason']->getMessage();
        echo "Transfer failed for {$key}: {$reason}\n";
    }
}

$transferManager = new S3TransferManager(
    null, []
);

// Use your custom tracker.
$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/file.txt', // Or an instance of StreamInterface.
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'file.txt',
        ],
        [],
        [],
        new MyProgressTracker()
    )
);
```

以下示例显示了正在进行的文件上传到控制台的示例输出：

```
Starting transfer of file.txt...
Progress: 42% (16.8MB of 40MB)
```

自定义内置的进度跟踪器

您可以通过以下几种方式自定义进度跟踪：

- 通过扩展AbstractTransferListener和实现来创建自定义实现ProgressTrackerInterface*<add link>*。
- 通过配置传递给内置跟踪器ConsoleProgressBar*<add link>*的，自定义现有实现。
- 实现自定义ProgressBar*<add link>*并将其传递给内置跟踪器。

以下示例说明如何自定义进度跟踪。请参阅类图，了解 SDK 中进度跟踪组件之间的关系。

自定义 **ConsoleProgressBar**

自定义默认条形格式

以下示例自定义了默认的进度条格式，即ColoredTransferProgressBarFormat*<add link>*。此格式根据传输状态以不同的颜色显示进度。

示例发生了变化：

- 从默认 “#” 到 “=” 的字符
- 条形宽度从默认值 50 到 100

```
<?php

use Aws\S3\S3Client;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Progress\ConsoleProgressBar;
use Aws\S3\S3Transfer\Progress\SingleProgressTracker;
use Aws\S3\S3Transfer\S3TransferManager;

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

$progressBar = new ConsoleProgressBar(
    progressBarChar: '=',
    progressBarWidth: 100,
```

```
// Default `ColoredTransferProgressBarFormat` is used.
);
$progressTracker = new SingleProgressTracker(
    progressBar: $progressBar,
);
$s3Client = new S3Client([
    'region' => 'us-east-2',
]);
$s3TransferManager = new S3TransferManager($s3Client);
$source = "/path/file.txt";

$result = $s3TransferManager->upload(
    new UploadRequest(
        source: $source,
        uploadRequestArgs: [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'key',
        ],
        progressTracker: $progressTracker,
    )
)->wait();
print_r($result);
```

下图显示了宽度为 100、字符为 “=” 的自定义内置ConsoleProgressBar组件的控制台输出：

```
key:
[                                                  ] 0% 0/1048576 B

key:
[==================================================] 100% 1048576/1048576 B
```

使用普通进度条格式进行自定义

以下示例使用PlainProgressBarFormat*<add link>*带有自定义字符和宽度的内置进度条来自定义默认进度条。PlainProgressBarFormat显示屏仅为黑色。颜色不会根据传输状态而改变。

示例发生了变化：

- 从默认 “#” 到 “*” 的字符
- 条形宽度从默认值 50 到 25
- 默认progressBarFormat实例从ColoredTransferProgressBarFormat到PlainProgressBarFormat

```
<?php

use Aws\S3\S3Client;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Progress\ConsoleProgressBar;
use Aws\S3\S3Transfer\Progress\PlainProgressBarFormat;
use Aws\S3\S3Transfer\Progress\SingleProgressTracker;
use Aws\S3\S3Transfer\S3TransferManager;

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

$progressBar = new ConsoleProgressBar(
    progressBarChar: '*',
    progressBarWidth: 25,
    progressBarFormat: new PlainProgressBarFormat(),
);
$progressTracker = new SingleProgressTracker(
    progressBar: $progressBar,
);
$s3Client = new S3Client([
    'region' => 'us-east-2',
]);
$s3TransferManager = new S3TransferManager($s3Client);
$source = "/path/file.txt";

$result = $s3TransferManager->upload(
    new UploadRequest(
        source: $source,
        uploadRequestArgs: [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'key',
        ],
        progressTracker: $progressTracker,
    )
)->wait();
print_r($result);
```

下图显示了ConsoleProgressBar使用宽度为 25、字符为 “*” 的纯进度条格式的自定义内置组件的控制台输出：

```
key:
[                              ] 0%
key:
[***********                  ] 44%
key:
[*************************] 100%
```

通过创建新的条形格式进行自定义

以下示例使用新的进度条格式（扩展 ProgressBarFormat*<add link>*）而不是自定义内置格式来自定义默认进度条。

```
<?php

use Aws\S3\S3Client;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Progress\ConsoleProgressBar;
use Aws\S3\S3Transfer\Progress\ProgressBarFormat;
use Aws\S3\S3Transfer\Progress\SingleProgressTracker;
use Aws\S3\S3Transfer\S3TransferManager;

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

// Implement a custom bar format.
class PercentBarFormat extends ProgressBarFormat
{
    public const FORMAT_TEMPLATE = "|object_name| - |percent|% [|transferred| |unit|
ytes]";
    public const FORMAT_PARAMETERS = [
        'object_name',       // `uploadRequestArgs` Key parameter.
        'percent',           //  Percent transferred.
        // 'progress_bar',      Optional, if used, default is a `ConsoleProgressBar`.
        // 'to_be_transferred', Optional.
        'transferred',       //  Default is bytes transferred.
        'unit',              //  Default is 'B'.
    ];

    public function getFormatTemplate(): string
    {
```

```
        return self::FORMAT_TEMPLATE;
    }

    public function getFormatParameters(): array
    {
        return self::FORMAT_PARAMETERS;
    }

    protected function getFormatDefaultParameterValues(): array
    {
        return [];
    }
}

// Create an instance of the custom implemention
$progressBarFormat = new PercentBarFormat();
$progressBar = new ConsoleProgressBar(
    progressBarFormat: $progressBarFormat,
);
$progressTracker = new SingleProgressTracker(
    progressBar: $progressBar,
);
$s3Client = new S3Client([
    'region' => 'us-east-2',
]);
$s3TransferManager = new S3TransferManager($s3Client);
$source = "/path/file";

$result = $s3TransferManager->upload(
    new UploadRequest(
        source: $source,
        uploadRequestArgs: [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'key',
        ],
        progressTracker: $progressTracker,
    )
)->wait();
print_r($result);
```

下图显示了ConsoleProgressBar使用自定义进度条格式的自定义内置组件的控制台输出：

```
key - 0% [0 Bytes]
key - 11% [1153434 Bytes]
key - 55% [5767168 Bytes]
key - 100% [10485760 Bytes]
```

自定义条形格式接受多个参数：

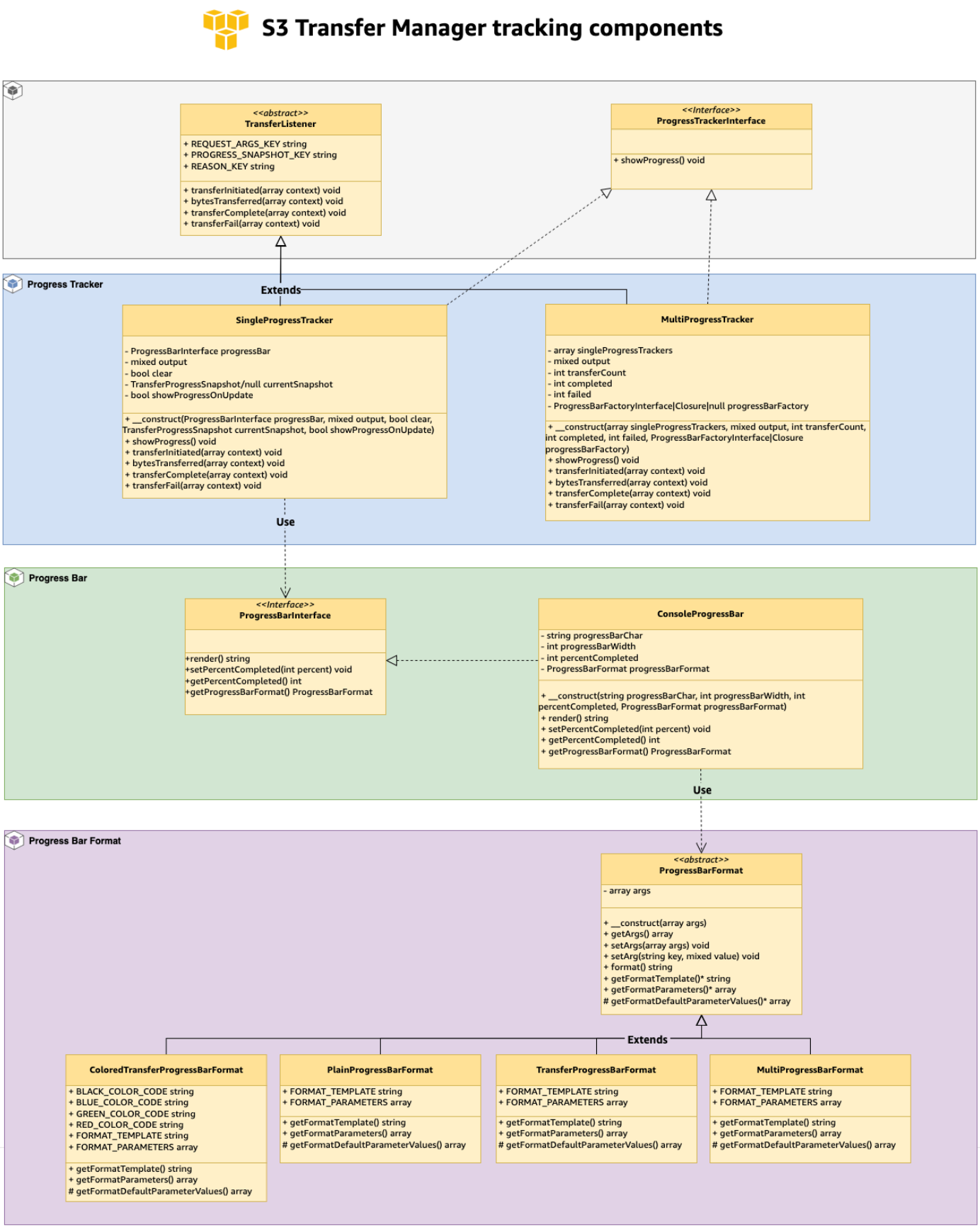
- object_name
- 百分比
- 进度条
- 待转移
- 已转移
- unit

您可以组合分隔的（竖线字符）参数名称来自定义输出。该示例使用以下模板。
该`PercentBarFormat`类的`getFormatTemplate`方法返回以下模板：

```
"|object_name| - |percent|% [|transferred| |unit|ytes]"
```

类图：S3 传输管理器跟踪组件

以下类图显示了跟踪组件如何在 S3 传输管理器中协同工作。了解这些关系有助于您实现自定义监听器和进度跟踪。


S3 Transfer Manager tracking components
<<abstract>>
TransferListener
+ REQUEST_ARGS_KEY string
+ PROGRESS_SNAPSHOT_KEY string
+ REASON_KEY string
+ transferInitiated(array context) void
+ bytesTransferred(array context) void
+ transferComplete(array context) void
+ transferFail(array context) void
<<Interface>>
ProgressTrackerInterface
+ showProgress() void
Progress Tracker
Extends
SingleProgressTracker
- ProgressBarInterface progressBar
- mixed output
- bool clear
- TransferProgressSnapshot/null currentSnapshot
- bool showProgressOnUpdate
+ __construct(ProgressBarInterface progressBar, mixed output, bool clear, TransferProgressSnapshot currentSnapshot, bool showProgressOnUpdate)
+ showProgress() void
+ transferInitiated(array context) void
+ bytesTransferred(array context) void
+ transferComplete(array context) void
+ transferFail(array context) void
MultiProgressTracker
- array singleProgressTrackers
- mixed output
- int transferCount
- int completed
- int failed
- ProgressBarFactoryInterface|Closure|null progressBarFactory
+ __construct(array singleProgressTrackers, mixed output, int transferCount, int completed, int failed, ProgressBarFactoryInterface|Closure progressBarFactory)
+ showProgress() void
+ transferInitiated(array context) void
+ bytesTransferred(array context) void
+ transferComplete(array context) void
+ transferFail(array context) void
Use
Progress Bar
<<Interface>>
ProgressBarInterface
+render() string
+setPercentCompleted(int percent) void
+getPercentCompleted() int
+getProgressBarFormat() ProgressBarFormat
ConsoleProgressBar
- string progressBarChar
- int progressBarWidth
- int percentCompleted
- ProgressBarFormat progressBarFormat
+ __construct(string progressBarChar, int progressBarWidth, int percentCompleted, ProgressBarFormat progressBarFormat)
+ render() string
+ setPercentCompleted(int percent) void
+ getPercentCompleted() int
+ getProgressBarFormat() ProgressBarFormat
Use
Progress Bar Format
<<abstract>>
ProgressBarFormat
- array args
+ __construct(array args)
+ getArgs() array
+ setArgs(array args) void
+ setArg(string key, mixed value) void
+ format() string
+ getFormatTemplate()* string
+ getFormatParameters()* array
# getFormatDefaultParameterValues()* array
Extends
ColoredTransferProgressBarFormat
+ BLACK_COLOR_CODE string
+ BLUE_COLOR_CODE string
+ GREEN_COLOR_CODE string
+ RED_COLOR_CODE string
+ FORMAT_TEMPLATE string
+ FORMAT_PARAMETERS array
+ getFormatTemplate() string
+ getFormatParameters() array
# getFormatDefaultParameterValues() array
PlainProgressBarFormat
+ FORMAT_TEMPLATE string
+ FORMAT_PARAMETERS array
+ getFormatTemplate() string
+ getFormatParameters() array
# getFormatDefaultParameterValues() array
TransferProgressBarFormat
+ FORMAT_TEMPLATE string
+ FORMAT_PARAMETERS array
+ getFormatTemplate() string
+ getFormatParameters() array
# getFormatDefaultParameterValues() array
MultiProgressBarFormat
+ FORMAT_TEMPLATE string
+ FORMAT_PARAMETERS array
+ getFormatTemplate() string
+ getFormatParameters() array
# getFormatDefaultParameterValues() array

错误处理

S3 传输管理器使用承诺进行异步操作。你可以使用 promise 的`otherwise`或`then`方法来处理错误。你也可以将你的实现封装在一个`try-catch`区块中。

承诺错误处理

使用 `otherwise`：

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);
$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/file.txt',
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'file.txt',
        ]
    )
);

$uploadPromise->otherwise(function (Throwable $reason) {
    echo "Upload failed: " . $reason->getMessage() . "\n";
});
```

使用 `then`：

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\Models\UploadResult;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);
$uploadPromise = $transferManager->upload(
```

```
    new UploadRequest(
        '/path/to/file.txt',
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'file.txt',
        ]
    )
);


$uploadPromise->then(
    function (UploadResult $result) {
        echo "Upload succeeded!\n";
    },
    function (Throwable $error) {
        echo "Upload failed: " . $error->getMessage() . "\n";
    }
)->wait();
```

使用 try-catch 方块

```
<?php

use Aws\S3\S3Transfer\Exception\S3TransferException;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/file.txt',
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'file.txt',
        ]
    )
);

try {
    $uploadPromise->wait();
```

```
} catch (S3TransferException $exception) {
    echo "Upload failed: " . $exception->getMessage() . "\n";
}
```

<Add more details about S3TransferManager exception and also a link to the API reference for it.>

目录操作失败策略

对于目录操作，您还可以指定失败策略回调。此回调处理单个文件的失败，并决定是否应继续上传目录。

Example使用 “failure_policy” 函数进行操作 uploadDirectory

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [
            'failure_policy' => function (
                $requestArgs,
                $uploadDirectoryRequestArgs,
                $reason,
                $uploadDirectoryResponse
            ) {
                echo "Failed to upload {$requestArgs['Key']}: " .
                    "{$reason->getMessage()}\n";
                echo "So far, uploaded: " .
                    "{$uploadDirectoryResponse->getObjectsUploaded()}, " .
                    "failed: {$uploadDirectoryResponse->getObjectsFailed()}\n";

                // Return true to continue with other files,
                // or throw an exception to abort.
```

```
                return true;
            },
        ]
    )
);
$uploadDirPromise->wait();
```

Example使用 “failure_policy” 函数进行操作 downloadDirectory

```
<?php

use Aws\S3\S3Transfer\Models\DownloadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

// Log errors but continue with other files.
$downloadDirPromise = $transferManager->downloadDirectory(
    new DownloadDirectoryRequest(
        'amzn-s3-demo-bucket',
        '/path/to/directory',
        [],
        [
            'failure_policy' => function (
                $requestArgs,
                $downloadDirectoryRequestArgs,
                $reason,
                $downloadDirectoryResponse
            ) {
                error_log("Failed to download {$requestArgs['Key']}: " .
                    "{$reason->getMessage()}");

                // If we've had too many failures, abort the entire operation.
                if ($downloadDirectoryResponse->getObjectsFailed() > 10) {
                    throw new \Exception(
                        "Too many download failures, aborting operation"
                    );
                }

                // Return void to continue with other files.
                return;
```

```
            },
        ]
    )
);
$downloadDirPromise->wait();
```

高级用法

本节介绍 S3 传输管理器的高级使用模式和技术。

分段上传

S3 传输管理器会自动对大文件使用分段上传。您可以使用配置选项自定义此行为。

```
<?php

use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

$uploadPromise = $transferManager->upload(
    new UploadRequest(
        '/path/to/large/file.mp4',
        [
            'Bucket' => 'amzn-s3-demo-bucket',
            'Key'    => 'videos/large-file.mp4',
        ],
        [
            // Use multipart upload for files larger than 100MB.
            'multipart_upload_threshold_bytes' => 100 * 1024 * 1024,

            // Use 25MB parts for multipart uploads.
            'target_part_size_bytes'           => 25 * 1024 * 1024,
        ]
    )
);
$uploadPromise->wait();
```

分段下载

您可以自定义分段下载行为。

```
<?php

use Aws\S3\S3Transfer\AbstractMultipartDownloader;
use Aws\S3\S3Transfer\Models\DownloadRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

$downloadPromise = $transferManager->download(
    new DownloadRequest(
        's3://amzn-s3-demo-bucket/large-file.mp4',
        [],
        [
            // Use 25MB parts for multipart downloads.
            'target_part_size_bytes'  => 25 * 1024 * 1024,

            // Use ranged-based download instead of part-based.
            'multipart_download_type' =>
 AbstractMultipartDownloader::RANGED_GET_MULTIPART_DOWNLOADER,
        ]
    )
);
$downloadPromise->wait();
```

有关该AbstractMultipartDownloader类其他成员的更多信息，请参阅 API 文档*<add link>*。

自定义过滤器

您可以使用筛选功能有选择地上传或下载文件以进行目录操作。

上传目录过滤器

有关参数信息的更多信息，请参阅该uploadDirectory方法的 filter 可调用选项。

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

```
$transferManager = new S3TransferManager(null, []);

// Upload files modified in the last 24 hours.
$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [
            'filter' => function ($file) {
                $modTime = filemtime($file);
                return (time() - $modTime) < 86400; // 24 hours
            },
        ]
    )
);
$uploadDirPromise->wait();
```

下载目录过滤器

有关参数信息的更多信息，请参阅该downloadDirectory方法的 filter 可调用选项。

```
<?php

use Aws\S3\S3Transfer\Models\DownloadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

// Download files with a specific prefix.
$downloadDirPromise = $transferManager->downloadDirectory(
    new DownloadDirectoryRequest(
        'amzn-s3-demo-bucket',
        '/path/to/directory',
        [],
        [
            'filter' => function ($key) {
                return strpos($key, 'reports/2023/') === 0;
            },
        ]
    )
);
```

```
$downloadDirPromise->wait();
```

预请求回调

您可以使用预请求回调修改每个文件的请求参数。有关参数的更多信息，请参阅以下内容：

- 该方法的 put_object_request_cal laback 可调用选项 `uploadDirectory`
- 该@@ 方法的 download_object_request_mod ifier `downloadDirectory`

Example该方法的预请求回调 uploadDirectory

```
<?php

use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\S3TransferManager;

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

$transferManager = new S3TransferManager(null, []);

// Set custom metadata for each uploaded file.
$uploadDirPromise = $transferManager->uploadDirectory(
    new UploadDirectoryRequest(
        '/path/to/directory',
        'amzn-s3-demo-bucket',
        [],
        [
            'upload_object_request_modifier' => function ($args) {
                $extension = pathinfo($args['Key'], PATHINFO_EXTENSION);

                switch ($extension) {
                    case 'jpg':
                    case 'jpeg':
                        $args['ContentType'] = 'image/jpeg';
                        break;
                    case 'png':
                        $args['ContentType'] = 'image/png';
                        break;
                    case 'pdf':
                        $args['ContentType'] = 'application/pdf';
                        break;
                }
```

```
                $args['Metadata'] = [
                    'uploaded_at' => date('Y-m-d H:i:s'),
                ];
            },
        ]
    )
);
$uploadDirPromise->wait();
```

Ready-to-use 例子

本节提供了一个完整的帮助器类，其中包含用于常见 S3 传输管理器操作的静态方法。辅助类简化了使用。它在内部处理初始化和配置。

S3 TransferHelper 级

```
<?php

use Aws\S3\S3Transfer\Models\DownloadDirectoryRequest;
use Aws\S3\S3Transfer\Models\DownloadFileRequest;
use Aws\S3\S3Transfer\Models\DownloadRequest;
use Aws\S3\S3Transfer\Models\UploadDirectoryRequest;
use Aws\S3\S3Transfer\Models\UploadRequest;
use Aws\S3\S3Transfer\S3TransferManager;
use Psr\Http\Message\StreamInterface;

class S3TransferHelper
{
    /**
     * Upload a file from a local path to S3.
     */
    public static function uploadFile(
        string $filePath,
        string $bucket,
        string $key
    ): array
    {
        $transferManager = new S3TransferManager();

        $uploadRequest = new UploadRequest(
            $filePath,
            [
                'Bucket' => $bucket,
```

```
                'Key'    => $key,
            ],
            []
        );

        return $transferManager->upload($uploadRequest)->wait()->getResult();
    }

    /**
     * Upload from a stream to S3
     */
    public static function uploadFromStream(
        StreamInterface $stream,
        string          $bucket,
        string          $key,
        bool            $trackProgress = true,
    ): array
    {
        $transferManager = new S3TransferManager();

        $uploadRequest = new UploadRequest(
            $stream,
            [
                'Bucket' => $bucket,
                'Key'    => $key,
            ],
            ['track_progress' => $trackProgress]
        );

        return $transferManager->upload($uploadRequest)->wait()->getResult();
    }

    /**
     * Download a file from S3 to a local path
     */
    public static function downloadFile(
        string $bucket,
        string $key,
        string $destinationPath,
        bool   $failsWhenDestinationExists = false,
        bool   $trackProgress = true,
    ): void
    {
        $transferManager = new S3TransferManager();
```

```
        $downloadRequest = new DownloadRequest(
            [
                'Bucket' => $bucket,
                'Key'    => $key
            ],
            [],
            ['track_progress' => $trackProgress]
        );

        $downloadFileRequest = new DownloadFileRequest(
            $destinationPath,
            $failsWhenDestinationExists,
            $downloadRequest
        );

        $transferManager->downloadFile($downloadFileRequest)->wait();
    }

    /**
     * Upload an entire directory to S3
     */
    public static function uploadDirectory(
        string $sourceDirectory,
        string $bucket,
        string $s3Prefix = '',
        bool   $trackProgress = true,
    ): array
    {
        $transferManager = new S3TransferManager();
        $uploadDirectoryRequest = new UploadDirectoryRequest(
            $sourceDirectory,
            $bucket,
            [],
            [
                's3_prefix'      => $s3Prefix,
                'track_progress' => $trackProgress,
            ]
        );

        $result = $transferManager->uploadDirectory($uploadDirectoryRequest)->wait();

        return [
            'uploaded' => $result->getObjectsUploaded(),
            'failed'   => $result->getObjectsFailed(),
```

```
        ];
    }

    /**
     * Download directory from S3
     */
    public static function downloadDirectory(
        string $bucket,
        string $destinationDirectory,
        string $s3Prefix = '',
        bool   $trackProgress = true,
    ): array
    {
        $transferManager = new S3TransferManager();
        $downloadDirectoryRequest = new DownloadDirectoryRequest(
            $bucket,
            $destinationDirectory,
            [],
            [
                's3_prefix'      => $s3Prefix,
                'track_progress' => $trackProgress,
            ]
        );

        $result = $transferManager->downloadDirectory($downloadDirectoryRequest)-
>wait();

        return [
            'downloaded' => $result->getObjectsDownloaded(),
            'failed'     => $result->getObjectsFailed(),
        ];
    }
}
```

辅助类用法示例

以下示例说明如何使用该S3TransferHelper类。

```
<?php

require __DIR__ . '/../vendor/autoload.php';
require __DIR__ . '/S3TransferHelper.php';

// Upload a local file
```

```
$result = S3TransferHelper::uploadFile(
    '/path/to/local/document.pdf',
    'amzn-s3-demo-bucket',
    'documents/document.pdf'
);

echo "File uploaded successfully. ETag: " . $result['ETag'] . "\n";

// Download a file to local path
S3TransferHelper::downloadFile(
    'amzn-s3-demo-bucket',
    'documents/document.pdf',
    '/path/to/local/downloaded-document.pdf',
    false, // Don't fail if destination exists
    true   // Track progress
);

echo "File downloaded successfully!\n";

// Upload entire directory
$result = S3TransferHelper::uploadDirectory(
    '/path/to/local/photos',
    'amzn-s3-demo-bucket',
    'photos/vacation-2023/', // S3 prefix
    true // Track progress
);

echo "Directory upload completed.\n";
echo "Uploaded: {$result['uploaded']} files\n";
echo "Failed: {$result['failed']} files\n";

// Download directory from S3
$result = S3TransferHelper::downloadDirectory(
    'amzn-s3-demo-bucket',
    '/path/to/local/downloads',
    'photos/vacation-2023/', // S3 prefix to download
    true // Track progress
);

echo "Directory download completed.\n";
echo "Downloaded: {$result['downloaded']} files\n";
echo "Failed: {$result['failed']} files\n";
```

使用 适用于 PHP 的 Amazon SDK 版本 3 将目录传入和传出 Amazon S3

您可以使用 适用于 PHP 的 Amazon SDK 版本 3 中的Transfer类将整个目录上传到 Amazon S3 存储桶，并将整个存储桶下载到本地目录。

将本地目录上传到 Amazon S3

Aws\S3\Transfer对象执行传输。以下示例展示了如何将本地文件目录递归上传到 Amazon S3 存储桶。

```
// 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.
$directoryTransfer = new \Aws\S3\Transfer($client, $source, $dest);

// Perform the transfer synchronously.
$directoryTransfer->transfer();
```

在此示例中，我们创建了一个 Amazon S3 客户端，创建了一个 Transfer 对象，并执行了同步传输。

前面的示例演示了执行传输所需的最小代码量。Transfer对象也可以异步执行传输，并具有各种配置选项，可用于自定义传输。

您可以通过在 s3:// URI 中提供键前缀，将本地文件上传到 Amazon S3 存储桶的“子文件夹”。以下示例将磁盘上的本地文件上传到 bucket 存储桶，并将文件存储在 foo 键前缀下。

```
$source = '/path/to/source/files';
$dest = 's3://bucket/foo';
$directoryTransfer = new \Aws\S3\Transfer($client, $source, $dest);
$directoryTransfer->transfer();
```

下载 Amazon S3 存储桶

您可以通过将 $source 参数指定为 Amazon S3 URI（例如 s3://bucket）并将 $dest 参数指定为本地目录的路径，从而将 Amazon S3 存储桶递归下载到磁盘上的本地目录。

```
// Where the files will be sourced from.
$source = 's3://bucket';

// Where the files will be transferred to.
$dest = '/path/to/destination/dir';

$directoryTransfer = new \Aws\S3\Transfer($client, $source, $dest);
$directoryTransfer->transfer();
```

> **Note**
>
> 下载存储桶中的对象时，SDK 会自动创建所有必要的目录。

您可以通过在 Amazon S3 URI 中于存储桶后面添加键前缀，仅下载存储在“虚拟文件夹”下的对象。以下示例仅下载存储在给定存储桶的“/foo”键前缀下的文件。

```
$source = 's3://bucket/foo';
$dest = '/path/to/destination/dir';
$directoryTransfer = new \Aws\S3\Transfer($client, $source, $dest);
$directoryTransfer->transfer();
```

配置

Transfer 对象构造函数接受以下参数。

$client

Aws\ClientInterface 对象用于执行传输。

$source（字符串 | **Iterator**）

正在传输的源数据。这可以指向本地磁盘上的路径（例如 /path/to/files）或 Amazon S3 存储桶（例如 s3://bucket）。s3:// URI 还可能包含可用于仅传输通用前缀下的对象的键前缀。

如果 $source 参数为 Amazon S3 URI，则 $dest 参数必须是一个本地目录（反之亦然）。

除了提供字符串值之外，您还可以提供生成绝对文件名的 \Iterator 对象。如果提供\Iterator对象，则必须在$options关联数组中提供一个base_dir选项。

$dest

文件传输目的地。如果 $source 参数是磁盘上的本地路径，则 $dest 必须为 Amazon S3 存储桶 URI（例如 s3://bucket）。如果 $source 参数是 Amazon S3 存储桶 URI，则 $dest 参数必须为磁盘上的本地路径。

$options

传输选项的关联数组。以下传输选项是有效的：

add_content_md5 (bool)

设置true为可计算上传的 MD5 校验和。

base_dir（字符串）

源的基本目录（如果 $source 为迭代器）。如果 $source 选项不是数组，则此选项将被忽略。

before（可调用）

每次传输前要调用的回调。此回调应具有类似 function (Aws \Command $command) {...} 的函数签名。提供的命令将是 GetObject、PutObject、CreateMultipartUpload、UploadPart 或 CompleteMultipartUpload 命令。

mup_threshold (int)

字节大小，应使用分段上传而不是 PutObject。默认值为 16777216（16 MB）。

concurrency（整数，默认值=5）

并发上传的文件数。理想的并发值会有所不同，具体取决于正在上传的文件数以及每个文件的平均大小。通常，更高的并发度对较小文件有利，对较大文件则不然。

debug (bool)

设置为 true 可打印输出调试信息用于传输。设置为 fopen() 资源可写入特定流，而不是写入 STDOUT。

异步传输

Transfer 对象是 GuzzleHttp\Promise\PromisorInterface 的实例。这意味着传输可以异步发生，并通过调用对象的 promise 方法启动。

```
$source = '/path/to/source/files';
$dest = 's3://bucket';
$directoryTransfer = new \Aws\S3\Transfer($client, $source, $dest);

// Initiate the transfer and get a promise.
$promise = $directoryTransfer->promise();

// Do something when the transfer is complete using the then() method.
$promise->then(function () {
    echo 'Done!';
});
```

如果有任何文件传输失败，则承诺将被拒绝。您可以使用 Promise 的 otherwise 方法以异步方式处理失败的传输。出现错误时，otherwise 函数接受要调用的回调。回调接受拒绝$reason的，通常是的实例Aws\Exception\AwsException（尽管可以将任何类型的值传递给回调）。

```
$promise->otherwise(function ($reason) {
    echo 'Transfer failed: ';
    var_dump($reason);
});
```

由于 Transfer 对象返回一个 Promise，这些传输可以与其他异步 Promise 同时进行。

自定义目录传输

通过向构造函数添加回调，可以自定义其Transfer执行的选项。

```
$uploader = new Transfer($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';
        }
    },
]);
```

适用于 PHP 的 Amazon SDK 版本 3 中的 Amazon S3 客户端加密

使用客户端加密，数据可在您的环境中直接加密和解密。这就意味着，数据在传输到 Amazon S3 之前已加密，您无需使用外部服务来处理加密。对于新的实现，我们建议使用`S3EncryptionClientV3`和`S3EncryptionMultipartUploaderV3`而不是和`S3EncryptionClientV2`以及`S3EncryptionMultipartUploaderV2`已弃用的`S3EncryptionClient`和。`S3EncryptionMultipartUploader`建议仍在使用已弃用版本的旧实现尝试迁移。`S3EncryptionClientV3` 继续支持对使用旧版 `S3EncryptionClient` 加密的数据进行解密。

适用于 PHP 的 Amazon SDK 实现了信封加密，并使用 OpenSSL 进行加密和解密。该实现可与其他与其功能支持相匹配 SDKs 的实现互操作。它还与开发工具包基于 Promise 的异步工作流程相兼容。

迁移指南

对于那些试图从已弃用的客户端迁移到新客户端的用户，这里有从 v1 迁移到 v2 的迁移指南，此处有从 v2 迁移到 v3 的迁移指南。

设置

要开始使用客户端加密，您需要：

- Amazon KMS 加密密钥
- 一个 S3 存储桶

在运行任何示例代码之前，请配置您的 Amazon 证书。参见适用于 PHP 的 Amazon SDK 版本 3 的凭证。

加密

上传加密对象除了标准参数之外还`S3EncryptionClientV3`需要四个额外的`PutObject`参数：

- `'@KmsEncryptionContext'` 是一个密钥值对，可用于为加密对象添加额外的安全层。加密客户端必须传入相同密钥，这将在 get 调用时自动进行。如果不需要其他上下文，请传入空数组。
- `@CipherOptions` 是加密的其他配置，包括要使用的密码和密钥大小。
- `@MaterialsProvider` 是一个提供程序，用于处理生成密码密钥和初始化向量，以及加密密码密钥。
- `@CommitmentPolicy`是一种策略选项，它规定了如何使用密钥承诺或不使用密钥承诺来读取对象，以及如何使用密钥承诺或不使用密钥承诺来编写对象。

```
use Aws\S3\S3Client;
use Aws\S3\Crypto\S3EncryptionClientV3;
use Aws\Kms\KmsClient;
use Aws\Crypto\KmsMaterialsProviderV3;

 // Let's construct our S3EncryptionClient using an S3Client
 $encryptionClient = new S3EncryptionClientV3(
     new S3Client([
         'profile' => 'default',
         'region' => 'us-east-1',
         'version' => 'latest',
     ])
 );

 $kmsKeyId = 'kms-key-id';
 $materialsProvider = new KmsMaterialsProviderV3(
     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,
     '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
     '@KmsEncryptionContext' => ['context-key' => 'context-value'],
     'Bucket' => $bucket,
     'Key' => $key,
     'Body' => fopen('file-to-encrypt.txt', 'r'),
 ]);
```

> **Note**
>
> 除了基于 Amazon S3 和 Amazon KMS基于 Amazon S3 的服务错误外，如果配置不`'@CipherOptions'`正确，您可能会收到抛出的`InvalidArgumentException`对象。

解密

下载和解密对象除了标准`GetObject`参数外，还有五个额外的参数，其中两个是必需的。客户端将为您检测基本的密码选项。

- **`'@SecurityProfile'`**: 如果设置为 “V3”，则仅在兼容 v3 的情况下加密的对象

  的对象。将此参数设置为 “V3_AND_LEGACY” 还允许解密以 V1 兼容格式加密的对象。要支持迁移，请将 @ 设置为 “V3_AND SecurityProfile _LEGACY”。仅在开发新应用程序时使用 “V3”。
- **`'@MaterialsProvider'`** 是一个提供程序，用于处理生成密码密钥和初始化向量，

  以及加密密码密钥。
- **`'@KmsAllowDecryptWithAnyCmk'`**：（可选）将此参数设置为 True 即可启用解密，

  无需向的构造函数提供 KMS 密钥 ID MaterialsProvider。默认值为 False。
- **`'@CipherOptions'`**（可选）是加密的其他配置，包括

  要使用的密码和密钥大小。
- **`@CommitmentPolicy`**policy 选项，它规定了如何使用任一方法读取对象

  关键承诺或不带关键承诺，以及如何用关键承诺或不带关键承诺来撰写对象。

```
$result = $encryptionClient->getObject([
    '@KmsAllowDecryptWithAnyCmk' => true,
    '@SecurityProfile' => 'V2_AND_LEGACY',
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_ALLOW_DECRYPT',
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

> **Note**
>
> 除了基于 Amazon S3 和 Amazon KMS基于 Amazon S3 的服务错误外，如果配置不`'@CipherOptions'`正确，您可能会收到抛出的`InvalidArgumentException`对象。

密码配置

`'Cipher'`（字符串）

加密客户端在加密时使用的密码方法。此时只支持“gcm”。

> **Important**
>
> PHP 通过版本 7.1 的更新包含了额外的参数，在使用 OpenSSL 进行 GCM 加密时，这些参数对于加密和解密是必要的。对于 PHP 7.0 及更早版本，加密客户端 `S3EncryptionClientV2` 和 `S3EncryptionMultipartUploaderV2` 提供并使用支持 GCM 的填充代码。但是，使用填充代码来处理大型输入的性能会比使用 PHP 7.1+ 的本机实现时慢得多，因此可能需要升级较旧的 PHP 版本环境才能有效使用它们。

`'KeySize'` (int)

生成的用于加密的内容加密密钥的长度。默认为 256 位。有效的配置选项为 256 位。

`'Aad'`（字符串）

可添加到加密负载中的可选“附加身份验证数据”。在解密时将验证此信息。`Aad` 仅在使用“gcm”密码时才可用。

> **Important**
>
> 并非所有人都支持其他身份验证数据 Amazon SDKs ，因此其他人 SDKs 可能无法解密使用此参数加密的文件。

元数据策略

您还可以选择提供实施 Aws\Crypto\MetadataStrategyInterface 的类的实例。这个简单的接口可保存和加载 Aws\Crypto\MetadataEnvelope，其中包含您的信封加密材料。开发工具包提供两个类来实施此功能：Aws\S3\Crypto\HeadersMetadataStrategy 和 Aws\S3\Crypto\InstructionFileMetadataStrategy。默认情况下使用 HeadersMetadataStrategy。

```
$strategy = new InstructionFileMetadataStrategy(
    $s3Client
);

$encryptionClient->putObject([
    '@MaterialsProvider' => $materialsProvider,
    '@MetadataStrategy' => $strategy,
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@KmsEncryptionContext' => [],
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

$result = $encryptionClient->getObject([
    '@KmsAllowDecryptWithAnyCmk' => false,
    '@MaterialsProvider' => $materialsProvider,
    '@SecurityProfile' => 'V3',
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@MetadataStrategy' => $strategy,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

调用 HeadersMetadataStrategy::class 也可提供 和 InstructionFileMetadataStrategy 的类名常量。

```
$result = $encryptionClient->putObject([
    '@MaterialsProvider' => $materialsProvider,
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@MetadataStrategy' => HeadersMetadataStrategy::class,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
```

```
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);
```

> **Note**
>
> 如果构造文件上传后发生错误，不会自动删除该文件。

分段上传

也可以利用客户端加密执行分段上传。Aws\S3\Crypto\S3EncryptionMultipartUploaderV3 会在上传之前准备用于加密的源流。使用 Aws\S3\MultipartUploader 和 Aws\S3\Crypto\S3EncryptionClientV3 创建的过程也与此类似。S3EncryptionMultipartUploaderV3 能以与 '@MetadataStrategy' 相同的方式处理 S3EncryptionClientV3 选项，以及所有可用的 '@CipherOptions' 配置。

```
$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV3(
    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 S3EncryptionMultipartUploaderV3(
    new S3Client([
        'region' => 'us-east-1',
        'version' => 'latest',
        'profile' => 'default',
    ]),
    fopen('large-file-to-encrypt.txt', 'r'),
```

```
    [
        '@MaterialsProvider' => $materialsProvider,
        '@CipherOptions' => $cipherOptions,
        '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
        'bucket' => $bucket,
        'key' => $key,
    ]
);
$multipartUploader->upload();
```

> **Note**
>
> 除了基于 Amazon S3 和 Amazon KMS基于 Amazon S3 的服务错误外，如果配置不`'@CipherOptions'`正确，您可能会收到抛出的`InvalidArgumentException`对象。

使用校验和实现数据完整性保护

Amazon Simple Storage Service (Amazon S3) 允许您在上传对象时指定校验和。当您指定校验和时，校验和与对象一起存储，并且可以在下载对象时验证该校验和。

传输文件时，校验和可提供额外的数据层完整性。使用校验和，您可以通过确认收到文件与原始文件是否匹配来验证数据一致性。有关 Amazon S3 校验和的更多信息，请参阅 Amazon Simple Storage Service 用户指南，包括支持的算法。

您可以灵活地选择最适合自己需求的算法，并让 SDK 计算校验和。或者，您也可以使用任一受支持的算法，提供预先计算好的校验和值。

> **Note**
>
> 从版本的 3.337.0 开始 适用于 PHP 的 Amazon SDK，SDK 通过自动计算上传的`CRC32`校验和来提供默认的完整性保护。如果您未提供预先计算的校验和值，或者没有指定 SDK 计算校验和时应使用的算法，SDK 就会计算此校验和。
> 该 SDK 还提供了可在外部设置的数据完整性保护全局设置，您可以在 Amazon SDK 和工具参考指南中查阅相关说明。

> **Important**
>
> 要使用该CRC32C算法，您的 PHP 环境需要安装 Amazon 公共运行时 (Amazon CRT) 扩展。

我们在两个请求阶段讨论校验和：上传对象和下载对象。

上传对象

您可以使用 S3Client 的 putObject 方法将对象上传到 Amazon S3。使用参数数组中的 ChecksumAlgorithm 对来启用校验和计算并指定算法。

```
$client = new \Aws\S3\S3Client(['region' => 'us-east-2']); // See the note below.
$result = $client->putObject([
        'Bucket' => 'amzn-s3-demo-bucket',
        'Key' => 'key',
        'ChecksumAlgorithm' => 'CRC32',
        'Body' => 'Object contents to test the checksum.'
]);
```

如果您未在请求中提供校验和算法，则校验和行为将根据您使用的 SDK 版本而异，具体如下表所示。

未提供校验和算法时的校验和行为

PHP SDK 版本	校验和行为
3.337.0 之前的版本	SDK 不会自动计算 CRC-based 校验和并在请求中提供校验和。
3.337.0 或更高版本	SDK 使用 CRC32 算法计算校验和，并在请求中提供该值。Amazon S3 通过计算自己的 CRC32 校验和并将其与 SDK 提供的校验和进行对比，从而验证传输的完整性。如果校验和匹配，则校验和将与对象一起保存。

使用预先计算的校验和值

与请求一起提供的预先计算校验和值会禁用 SDK 的自动计算，而是使用提供的值。

以下示例显示了具有预先计算的 SHA256 校验和的请求。

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

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

// Calculate the SHA256 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' => 'amzn-s3-demo-bucket',
    'Key' => 'key',
    'Body' => $body,
    'ChecksumSHA256' => $sha256
]);
```

如果 Amazon S3 确定指定算法的校验和值不正确，服务就会返回错误响应。

分段上传

您也可以将校验和用于分段上传。

如以下示例所示，在 MultipartUploader 构造函数的 params 数组中将校验和算法指定为键值对。

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

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

$mpUploader = new MultipartUploader($s3Client, $stream, [
    'bucket' => 'amzn-s3-demo-bucket',
    'key' => 'key',
    'params' => ['ChecksumAlgorithm' => 'CRC32']
]);
```

下载对象

使用 getObject 方法下载对象时，在以下情况下 SDK 会自动验证校验和：如果 ChecksumMode 键的值为 enabled。

以下代码段中的请求引导 SDK 通过计算校验和并比较值来验证响应中的校验和。

```
$result = $client->getObject([
        'Bucket' => 'amzn-s3-demo-bucket',
        'Key' => 'test-checksum-key',
        'ChecksumMode' => 'enabled',
]);
```

> **Note**
>
> 如果上传对象时没有使用校验和，则不会进行验证。

带有适用于 适用于 PHP 的 Amazon SDK 的指导的代码示例

本节包含了演示使用 适用于 PHP 的 Amazon SDK 的常见 Amazon 场景的代码示例。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon CloudFront 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来针对自定义 Amazon CloudSearch 域请求签名
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon CloudWatch 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon EC2 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来签署 Amazon OpenSearch Service 搜索请求
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Identity and Access Management 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Key Management Service 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Kinesis 示例
- AWS Elemental MediaConvert 使用 适用于 PHP 的 Amazon SDK 版本 3 的示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 示例
- 使用 Secrets Manager API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理密钥
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SES 示例

- 使用版本 3 的亚马逊 SNS 示 适用于 PHP 的 Amazon SDK 例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 示例
- 使用 适用于 PHP 的 Amazon SDK 版本 3 向 Amazon EventBridge 全球终端节点发送事件

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon CloudFront 示例

Amazon CloudFront 是一项 Amazon Web 服务，可加快从您自己的 Web 服务器或 Amazon 服务器（例如 Amazon S3）提供静态和动态 Web 内容的速度。CloudFront 通过全球数据中心（称作边缘站点）网络来传输内容。当用户请求您通过 CloudFront 分发的内容时，将被引导至提供最低延迟的边缘站点。如果内容尚未在该处缓存，CloudFront 将从原始服务器检索副本，处理副本，然后将其缓存以供将来的请求使用。

有关 CloudFront 的更多信息，请参阅 Amazon CloudFront 开发人员指南。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

使用 CloudFront API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon CloudFront 分配

Amazon CloudFront 将内容缓存在全球边缘站点中，以加快您存储在自己的服务器或 Amazon 服务（如 Amazon S3 和 Amazon EC2）上的静态和动态文件的分配速度。当用户从您的网站请求内容时，CloudFront 将从最近的边缘站点提供内容（如果文件已在该处缓存）。否则，CloudFront 将检索文件的副本，处理副本，然后将其缓存以供下一个请求使用。在边缘站点缓存内容将减少该区域中类似用户请求的延迟。

对于您创建的每个 CloudFront 分配，指定内容所在的位置以及如何在用户发出请求时分配内容。本主题重点介绍静态和动态文件（如 HTML、CSS、JSON 和图像文件）的分配。有关结合使用 CloudFront 与视频点播的信息，请参阅使用 CloudFront 的视频点播和实时流视频。

以下示例演示如何：

- 使用 CreateDistribution 创建分配。
- 使用 GetDistribution 获取分配。
- 使用 ListDistributions 列出分配。
- 使用 UpdateDistributions 更新分配。
- 使用 DisableDistribution 禁用分配。
- 使用 DeleteDistributions 删除分配。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon CloudFront 的更多信息，请参阅 Amazon CloudFront 开发人员指南。

创建 CloudFront 分配。

从 Amazon S3 桶创建分配。在以下示例中，已注释掉可选参数，但显示默认值。要向您的分配添加自定义项，请取消注释 `$distribution` 内的值和参数。

要创建 CloudFront 分配，请使用 CreateDistribution 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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 = 'amzn-s3-demo-bucket.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();
```

检索 CloudFront 分配

要检索指定 CloudFront 分配的状态和详细信息，请使用 GetDistribution 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

列出 CloudFront 分配

使用 ListDistributions 操作，从您的当前账户中获取指定 Amazon 区域中现有 CloudFront 分配的列表。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

更新 CloudFront 分配

更新 CloudFront 分配的操作与创建分配的操作类似。但是，当您更新分配时，更多字段都是必填字段且必须包含所有值。要对现有分配进行更改，我们建议您首先检索现有分配，然后更新您要在 `$distribution` 数组中更改的值。

要更新指定的 CloudFront 分配，请使用 UpdateDistribution 操作。

导入。

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

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

示例代码

```
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();
```

禁用 CloudFront 分配

要停用或删除分配，请将其状态从“已部署”更改为“已禁用”。

要禁用指定的 CloudFront 分配，请使用 DisableDistribution 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

删除 CloudFront 分配

一旦分配处于已禁用状态，您即可删除分配。

要删除指定的 CloudFront 分配，请使用 DeleteDistribution 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

使用 CloudFront API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon CloudFront 失效

Amazon CloudFront 将静态和动态文件的副本缓存在全球边缘站点中。要在所有边缘站点上删除或更新文件，请为每个文件或一组文件创建失效。

每个日历月中，您的前 1,000 个失效是免费的。要了解有关从 CloudFront 边缘站点删除内容的更多信息，请参阅使文件失效。

以下示例演示如何：

- 使用 CreateInvalidation 创建分配失效。
- 使用 GetInvalidation 获取分配失效。
- 使用 ListInvalidations 列出分配失效。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon CloudFront 的更多信息，请参阅 Amazon CloudFront 开发人员指南。

创建分配失效

通过指定需要删除的文件的路径位置来创建 CloudFront 分配失效。此示例将使分配中的所有文件均失效，但您可以在 `Items` 下标识特定文件。

要创建 CloudFront 分配失效，请使用 CreateInvalidation 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

获取分配失效

要检索有关 CloudFront 分配失效的状态和详细信息，请使用 GetInvalidation 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

列出分配失效

要列出所有当前 CloudFront 分配失效，请使用 ListInvalidations 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
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();
```

CloudFront URLs 使用 适用于 PHP 的 Amazon SDK 版本 3 签署亚马逊

URLs 通过签名，您可以向用户提供访问您的私有内容的权限。签名 URL 中包含的其他信息（例如，到期时间）可让您更好地控制对内容的访问。该额外信息出现在策略声明中，且是基于标准策略或自定义策略。有关如何设置私有分配以及为什么需要签署的信息 URLs，请参阅《亚马逊 CloudFront 开发者指南》 CloudFront中的通过亚马逊提供私有内容。

- 使用 getSigne Durl 创建已签名的亚马逊 CloudFront URL。
- 使用创建已签名的亚马逊 CloudFront Cookie getSignedCookie。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用亚马逊的更多信息 CloudFront，请参阅亚马逊 CloudFront 开发者指南。

签署私 CloudFront URLs 有发行版

您可以使用 SDK 中的 CloudFront 客户端对 URL 进行签名。首先，您必须创建一个 `CloudFrontClient` 对象。您可以使用固定政策或自定义政策为视频资源签名 CloudFront URL。

导入

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

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

示例代码

```
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();
```

创建时使用自定义策略 CloudFront URLs

要使用自定义策略，请提供 `policy` 键，而不是 `expires`。

导入

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

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

示例代码

```
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();
```

使用 CloudFront 签名网址

根据您要签名的 URL 使用的是“HTTP”还是“RTMP”方案，签名后的 URL 的形式将有所不同。如果是“HTTP”方案，将返回完整的绝对 URL。如果是“RTMP”方案，则为了方便起见，只会返回相对 URL。这是因为某些播放器要求分别提供主机和路径参数。

以下示例说明如何使用签名 URL 来构造显示视频的网页JWPlayer。相同类型的技术也适用于其他玩家FlowPlayer，例如，但需要不同的客户端代码。

```
<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>
```

为私人分发签名 CloudFront Cookie

作为已签名的替代方案 URLs，您还可以通过签名 Cookie 向客户授予访问私有分发的权限。借助经过签名的 Cookie，您可以提供对多个受限文件的访问权限，例如 HLS 格式视频的所有文件，或在网站订阅者区域中的所有文件。要详细了解为何要使用签名 Cookie 而不是签名 URLs（反之亦然），请参阅《亚马逊 CloudFront 开发者指南》中的在签名 Cookie URLs 和签名 Cookie 之间进行选择。

创建签名 cookie 的方法与创建签名 URL 的方法类似。唯一的区别是调用的方法（`getSignedCookie`，而不是 `getSignedUrl`）。

导入

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

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

示例代码

```
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();
```

创建 CloudFront Cookie 时使用自定义策略

与 `getSignedUrl` 相同，您可以提供 `'policy'` 参数（而不是 `expires` 参数），以及 `url` 参数，使用自定义策略针对 Cookie 签名。自定义策略可以在资源键中包含通配符。这样您就可以为多个文件创建一个签名 Cookie。

`getSignedCookie` 返回的键值对数组均需设置为 Cookie，这样才能为私有分配授予访问权限。

导入

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

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

示例代码

```
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();
```

向 Guzzle 客户端发送 CloudFront cookie

您也可以将这些 Cookie 传递至 `GuzzleHttp\Cookie\CookieJar`，与 Guzzle 客户端结合使用。

```
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');
```

有关更多信息，请参阅《亚马逊 CloudFront 开发者指南》中的使用签名 Cookie。

使用 适用于 PHP 的 Amazon SDK 版本 3 来针对自定义 Amazon CloudSearch 域请求签名

可以在 适用于 PHP 的 Amazon SDK 支持的范围之外对 Amazon CloudSearch 域请求进行自定义。当您需要向受 IAM 身份验证保护的域发出自定义请求时，可以使用 SDK 的凭证提供程序和签署人对任何 PSR-7 请求签名。

例如，如果您要按照 Cloud Search 的入门指南操作，并想在第 3 步中使用受 IAM 保护的域，则需要按以下方式签署并执行请求。

以下示例演示如何：

- 使用 SignatureV4 通过 Amazon 签名协议签署请求。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

签署 Amazon CloudSearch 域请求

导入。

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

use Aws\Credentials\CredentialProvider;
use Aws\Signature\SignatureV4;
use GuzzleHttp\Client;
use GuzzleHttp\Psr7\Request;
```

示例代码

```
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();
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon CloudWatch 示例

Amazon CloudWatch (CloudWatch) 是一项 Web 服务，可实时监控您的 Amazon Web Services 资源以及您在 Amazon 上运行的应用程序。您可以使用 CloudWatch 收集和跟踪指标，这些指标是您可衡量的相关资源和应用程序的变量。CloudWatch 警报可根据您定义的规则发送通知或者对您所监控的资源自动进行更改。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 来使用 Amazon CloudWatch 警报
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来从 Amazon CloudWatch 获取指标
- 使用 适用于 PHP 的 Amazon SDK 版本 3来在 Amazon CloudWatch 中发布自定义指标
- 使用 适用于 PHP 的 Amazon SDK 版本 3 向 Amazon CloudWatch 活动发送事件
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来将警报操作用于 Amazon CloudWatch 警报

使用 适用于 PHP 的 Amazon SDK 版本 3 来使用 Amazon CloudWatch 警报

Amazon CloudWatch 警报在指定时间段内监控某个指标。它在多个时间段内根据相对于给定阈值的指标值，执行一项或多项操作。

以下示例演示如何：

- 使用 DescribeAlarms 描述警报。
- 使用 PutMetricAlarm 创建警报。
- 使用 DeleteAlarms 删除警报。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

描述警报

导入。

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

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

示例代码

```
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();
```

创建警报

导入。

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

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

示例代码

```
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();
```

删除警报

导入。

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

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

示例代码

```
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();
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来从 Amazon CloudWatch 获取指标

指标是关于您的系统性能的数据。您可以启用对某些资源（例如 Amazon EC2 实例）或您自己的应用程序指标的详细监控。

以下示例演示如何：

- 使用 ListMetrics 列出指标。
- 使用 DescribeAlarmsForMetric 检索指标的警报。
- 使用 GetMetricStatistics 获取指定指标的统计数据。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

列出指标

导入。

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

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

示例代码

```
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();
```

检索指标的警报

导入。

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

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

示例代码

```
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' => 'amzn-s3-demo-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();
```

获取指标统计数据

导入。

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

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

示例代码

```
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' => 'amzn-s3-demo-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();
```

使用 适用于 PHP 的 Amazon SDK 版本 3来在 Amazon CloudWatch 中发布自定义指标

指标是关于您的系统性能的数据。警报会在您规定的时间范围内监控某一项指标。它在多个时间段内根据相对于给定阈值的指标值，执行一项或多项操作。

以下示例演示如何：

- 使用 PutMetricData 发布指标数据。
- 使用 PutMetricAlarm 创建警报。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

发布指标数据

导入。

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

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

示例代码

```
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();
```

创建警报

导入。

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

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

示例代码

```
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();
```

使用 适用于 PHP 的 Amazon SDK 版本 3 向 Amazon CloudWatch 活动发送事件

CloudWatch 事件向任意目标提供近乎实时的系统事件流，这些事件描述了 Amazon Web Services (Amazon) 资源的变化。通过简单规则，您可以匹配事件并将事件路由到一个或多个目标函数或流。

以下示例演示如何：

- 使用创建规则PutRule。
- 使用向规则添加目标PutTargets。
- 使用向事件发送自定义 CloudWatch 事件PutEvents。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

创建规则

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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());
}
```

向规则中添加目标

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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());
}
```

发送自定义事件

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来将警报操作用于 Amazon CloudWatch 警报

利用警报操作创建自动停止、终止、重启或恢复 Amazon EC2 实例的警报。当不再需要某个实例运行时，您可使用停止或终止操作。使用重启和恢复操作可以自动重启这些实例。

以下示例演示如何：

- 使用 EnableAlarmActions 为指定的警报启用操作。
- 使用 DisableAlarmActions 为指定的警报禁用操作。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

启用警报操作

导入。

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

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

示例代码

```
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();
```

禁用警报操作

导入。

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

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

示例代码

```
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();
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon EC2 示例

Amazon Elastic Compute Cloud (Amazon EC2) 是一项 Web 服务，可在云中提供虚拟服务器托管。该服务旨在通过提供大小可调整的计算容量来降低开发人员进行网络级云计算的难度。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon EC2 实例
- 在 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon EC2 上使用弹性 IP 地址
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来使用用于 Amazon EC2 的区域和可用区
- 结合使用 Amazon EC2 密钥对与 适用于 PHP 的 Amazon SDK 版本 3
- 在 Amazon EC2 中结合使用安全组与 适用于 PHP 的 Amazon SDK 版本 3

使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon EC2 实例

以下示例演示如何：

- 使用 DescribeInstances 描述 Amazon EC2 实例。
- 使用 MonitorInstances 为正在运行的实例启用详细监控。
- 使用 UnmonitorInstances 为正在运行的实例禁用监控。
- 使用 StartInstances 启动之前停止的 Amazon EBS-backed AMI。
- 使用 StopInstances 停止由 Amazon EBS 支持的实例。
- 使用 RebootInstances 请求重新启动一个或多个实例。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

描述实例

导入。

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

use Aws\Ec2\Ec2Client;
```

示例代码

```
$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";
    }
}
```

启用和禁用监控

导入。

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

示例代码

```
$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);
```

启动和停止 实例

导入。

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

示例代码

```
$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);
```

重启实例

导入。

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

示例代码

```
$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);
```

在 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon EC2 上使用弹性 IP 地址

弹性 IP 地址是专为动态云计算设计的静态 IP 地址。弹性 IP 地址与您的关联 Amazon Web Services 账户。它是公有 IP 地址，可从 Internet 访问。如果您的实例没有公有 IP 地址，则可以将弹性 IP 地址与您的实例关联以启用与 Internet 的通信。

以下示例演示如何：

- 使用描述您的一个或多个实例DescribeInstances。
- 使用获取弹性 IP 地址AllocateAddress。
- 使用将弹性 IP 地址与实例关联起来AssociateAddress。
- 使用释放弹性 IP 地址ReleaseAddress。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

描述实例

导入

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

use Aws\Ec2\Ec2Client;
```

示例代码

```
$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";
    }
}
```

分配和关联地址

导入

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

示例代码

```
$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);
```

发布地址

导入

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

示例代码

```
$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);
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来使用用于 Amazon EC2 的区域和可用区

Amazon EC2 托管在全球多个位置。这些位置由 Amazon 区域和可用区构成。每个区域都是一个独立的地理区域，其中包含多个相互隔离的位置，这些位置称为可用区。Amazon EC2 提供了将实例和数据放在多个位置的功能。

以下示例演示如何：

- 使用 DescribeAvailabilityZones 描述可供您使用的可用区。
- 使用 DescribeRegions 描述当前可供您使用的 Amazon 区域。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

描述可用区

导入。

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

示例代码

```
$ec2Client = new Aws\Ec2\Ec2Client([
```

```
    'region' => 'us-west-2',
    'version' => '2016-11-15',
    'profile' => 'default'
]);

$result = $ec2Client->describeAvailabilityZones();

var_dump($result);
```

描述区域

导入。

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

示例代码

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

$result = $ec2Client->describeRegions();

var_dump($result);
```

结合使用 Amazon EC2 密钥对与 适用于 PHP 的 Amazon SDK 版本 3

Amazon EC2 使用公有密钥密码系统来加密和解密登录信息。公有密钥加密法使用公有密钥来加密数据。随后，收件人使用私有密钥解密数据。公有和私有密钥被称为密钥对。

以下示例演示如何：

- 使用 CreateKeyPair 创建 2048 位 RSA 密钥对。
- 使用 DeleteKeyPair 删除指定密钥对。
- 使用 DescribeKeyPairs 描述一个或多个密钥对。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建密钥对

导入。

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

示例代码

```
$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);
```

删除密钥对

导入。

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

示例代码

```
$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);
```

描述密钥对

导入。

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

示例代码

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

$result = $ec2Client->describeKeyPairs();

var_dump($result);
```

在 Amazon EC2 中结合使用安全组与 适用于 PHP 的 Amazon SDK 版本 3

Amazon EC2 安全组起着虚拟防火墙的作用，可控制一个或多个实例的流量。为每个安全组添加规则来规定流入或流出其关联实例的流量。您可以随时修改安全组的规则。新规则将自动应用于与安全组相关联的所有实例。

以下示例演示如何：

- 使用 DescribeSecurityGroups 描述一个或多个安全组。
- 使用 AuthorizeSecurityGroupIngress 向安全组添加入口规则。
- 使用 CreateSecurityGroup 创建安全组。
- 使用 DeleteSecurityGroup 删除安全组。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

描述安全组

导入。

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

示例代码

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

$result = $ec2Client->describeSecurityGroups();

var_dump($result);
```

添加入口规则

导入。

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

示例代码

```
$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);
```

创建安全组

导入。

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

示例代码

```
$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';
```

删除安全组

导入。

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

示例代码

```
$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);
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来签署 Amazon OpenSearch Service 搜索请求

Amazon OpenSearch Service 是一项托管服务，让用户能够轻松地部署、操作和扩展 Amazon OpenSearch Service（一个流行的开源搜索和分析引擎）。OpenSearch Service 支持直接访问

Amazon OpenSearch Service API。这意味着，开发人员可以使用他们熟悉的工具以及强大的安全选项。许多 Amazon OpenSearch Service 客户端都支持请求签名，但如果您使用的客户端不支持，则可以使用 适用于 PHP 的 Amazon SDK 的内置凭证提供程序和签署人对任意 PSR-7 请求签名。

以下示例演示如何：

- 使用 SignatureV4 通过 Amazon 签名协议签署请求。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

签署 OpenSearch Service 请求

OpenSearch Service 使用签名版本 4。这意味着，您需要根据服务的签名名称（在此示例中为 `es`）和 OpenSearch Service 域的 Amazon 区域来签署请求。在 Amazon Web Services 一般参考 中的 Amazon 区域和终端节点页面上可以找到 OpenSearch Service 支持区域的完整列表。但在此示例中，我们将针对 `us-west-2` 区域中的 OpenSearch Service 域来签署请求。

您需要提供凭证，可以通过 SDK 的默认凭证提供程序链或通过 适用于 PHP 的 Amazon SDK 版本 3 的凭证中介绍的任何形式的凭证来提供。您还将需要一个 PSR-7 请求（在下面的代码中假定名为 `$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);
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Identity and Access Management 示例

Amazon Identity and Access Management (IAM) 是一项 Web 服务，支持 Amazon Web Services 客户在 Amazon 中管理用户和用户权限。该服务面向在云中有多个使用 Amazon 产品的用户或系统的组织。借助 IAM，您可以集中管理用户、安全凭证（如访问密钥），以及控制用户可访问何种 Amazon 资源的权限。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 IAM 访问密钥
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 IAM 用户
- 结合使用 IAM 账户别名与 适用于 PHP 的 Amazon SDK 版本 3
- 结合使用 IAM 策略与 适用于 PHP 的 Amazon SDK 版本 3
- 结合使用 IAM 服务器证书与 适用于 PHP 的 Amazon SDK 版本 3

使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 IAM 访问密钥

用户需要自己的访问密钥以编程方式调用 Amazon。要满足这一需要，您可以创建、修改、查看或轮换 IAM 用户的访问密钥 (访问密钥 ID 和秘密访问密钥)。默认情况下，当您创建访问密钥时，其状态为“活动”。这表示用户可以将访问密钥用于 API 调用。

以下示例演示如何：

- 使用 CreateAccessKey 创建秘密访问密钥和相应的访问密钥 ID。
- 使用 ListAccessKeys 返回有关与 IAM 用户关联的访问密钥 ID 的信息。
- 使用 GetAccessKeyLastUsed 检索有关上次使用访问密钥的时间的信息。
- 使用 UpdateAccessKey 将访问密钥的状态从“Active”更改为“Inactive”或相反。
- 使用 DeleteAccessKey 删除与 IAM 用户关联的访问密钥对。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建访问密钥

导入。

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

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

示例代码

```
$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());
}
```

列出访问密钥

导入。

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

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

示例代码

```
$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());
}
```

获取有关上次使用访问密钥的信息

导入。

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

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

示例代码

```
$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());
}
```

更新访问密钥

导入。

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

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

示例代码

```
$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());
}
```

删除访问密钥

导入。

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

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

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 IAM 用户

IAM 用户是在 Amazon 中创建的一个实体，代表使用其与 Amazon 进行交互的人员或服务。Amazon 中的用户包括名称和凭证。

以下示例演示如何：

- 使用 CreateUser 创建新 IAM 用户。
- 使用 ListUsers 列出 IAM 用户。
- 使用 UpdateUser 更新 IAM 用户。
- 使用 GetUser 检索有关 IAM 用户的信息。
- 使用 DeleteUser 删除 IAM 用户。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建 IAM 用户

导入。

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

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

示例代码

```
$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());
}
```

列出 IAM 用户

导入。

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

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

示例代码

```
$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());
}
```

更新 IAM 用户

导入。

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

use Aws\Exception\AwsException;
```

```
use Aws\Iam\IamClient;
```

示例代码

```
$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());
}
```

获取有关 IAM 用户的信息

导入。

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

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

示例代码

```
$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());
}
```

删除 IAM 用户

导入。

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

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

示例代码

```
$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());
}
```

结合使用 IAM 账户别名与 适用于 PHP 的 Amazon SDK 版本 3

如果您希望登录页面的 URL 包含贵公司名称（或其他友好标识符）而不是 Amazon Web Services 账户 ID，则可以为 Amazon Web Services 账户 ID 创建别名。如果创建 Amazon Web Services 账户 别名，您的登录页面 URL 将更改以包含该别名。

以下示例演示如何：

- 使用 CreateAccountAlias 创建别名。
- 使用 ListAccountAliases 列出与 Amazon Web Services 账户 关联的别名。
- 使用 DeleteAccountAlias 删除别名。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建别名

导入。

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

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

示例代码

```
$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());
}
```

列出账户别名

导入。

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

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

示例代码

```
$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());
}
```

删除别名

导入。

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

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

示例代码

```
$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());
}
```

结合使用 IAM 策略与 适用于 PHP 的 Amazon SDK 版本 3

您通过创建策略向用户授予权限。策略是一个文档，其中列出了用户可以执行的操作以及这些操作会影响的资源。默认情况下会拒绝未显式允许的任何操作或资源。可将策略附加到用户、用户组、用户代入的角色以及资源。

以下示例演示如何：

- 使用 CreatePolicy 创建托管策略。
- 使用 AttachRolePolicy 将策略附加到角色。
- 使用 AttachUserPolicy 将策略附加到用户。
- 使用 AttachGroupPolicy 将策略附加到组。
- 使用 DetachRolePolicy 删除角色策略。
- 使用 DetachUserPolicy 删除用户策略。
- 使用 DetachGroupPolicy 删除组策略。
- 使用 DeletePolicy 删除托管策略。

- 使用 DeleteRolePolicy 删除角色策略。
- 使用 DeleteUserPolicy 删除用户策略。
- 使用 DeleteGroupPolicy 删除组策略。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建策略

导入。

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

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

示例代码

```
$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());
}
```

将策略附加到角色

导入。

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

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

示例代码

```
$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());
}
```

将策略附加到用户

导入。

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

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

示例代码

```
$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());
}
```

将策略附加到组

导入。

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

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

示例代码

```
$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());
}
```

分离用户策略

导入。

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

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

示例代码

```
$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());
}
```

分离组策略

导入。

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

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

示例代码

```
$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());
}
```

删除策略

导入。

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

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

示例代码

```
$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());
}
```

删除角色策略

导入。

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

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

示例代码

```
$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());
}
```

删除用户策略

导入。

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

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

示例代码

```
$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());
}
```

删除组策略

导入。

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

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

示例代码

```
$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());
}
```

结合使用 IAM 服务器证书与 适用于 PHP 的 Amazon SDK 版本 3

要在 Amazon 上启用网站或应用程序的 HTTPS 连接，需要 SSL/TLS 服务器证书。要在 Amazon 上将从外部提供程序获得的证书与网站或应用程序结合使用，必须将证书上传到 IAM 或者导入 Amazon Certificate Manager 。

以下示例演示如何：

- 使用 ListServerCertificates 列出 IAM 中存储的证书。
- 使用 GetServerCertificate 检索有关证书的信息。
- 使用 UpdateServerCertificate 更新证书。
- 使用 DeleteServerCertificate 删除证书。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

列出服务器证书

导入。

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

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

示例代码

```
$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());
}
```

检索服务器证书

导入。

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

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

示例代码

```
$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());
```

```
}
```

更新服务器证书

导入。

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

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

示例代码

```
$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());
}
```

删除服务器证书

导入。

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

use Aws\Exception\AwsException;
```

```
use Aws\Iam\IamClient;
```

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Key Management Service 示例

Amazon Key Management Service (Amazon KMS) 是一项托管服务，可让您轻松创建和控制加密您的数据所用的加密密钥。有关 Amazon KMS 的更多信息，请参阅 Amazon KMS 文档。无论是编写安全 PHP 应用程序还是向其他 Amazon 服务发送数据，Amazon KMS 都可以帮助您控制哪些人可以使用密钥并访问您的加密数据。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

主题

- 使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理密钥
- 使用版本 3 加密和解密 Amazon KMS 数据密钥 适用于 PHP 的 Amazon SDK
- 使用 适用于 PHP 的 Amazon SDK 版本 3 使用 Amazon KMS 密钥策略
- 使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理授权
- 使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理别名

使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理密钥

Amazon Key Management Service (Amazon KMS) 中的主要资源是Amazon KMS keys。您可以使用 KMS 密钥来加密数据。

以下示例演示如何：

- 使用创建客户 KMS 密钥CreateKey。
- 使用生成数据密钥GenerateDataKey。
- 使用查看 KMS 密钥DescribeKey。
- 使用ListKeys获取 KMS 密钥的密钥 IDs 和密钥 ARNS。
- 使用启用 KMS 密钥EnableKey。
- 使用禁用 KMS 密钥DisableKey。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用 Amazon Key Management Service (Amazon KMS) 的更多信息，请参阅《Amazon KMS 开发者指南》。

创建 KMS 密钥

要创建 KMS 密钥，请使用CreateKey操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

生成数据密钥

要生成数据加密密钥，请使用GenerateDataKey操作。该操作返回其创建的数据密钥的明文和加密副本。指定 Amazon KMS key 生成数据密钥的。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

查看 KMS 密钥

要获取有关 KMS 密钥的详细信息，包括 KMS 密钥的 Amazon 资源名称 (ARN) 和密钥状态，请使用操作。DescribeKey

DescribeKey 不会获取别名。要获取别名，请使用ListAliases操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

获取 KMS 密钥 ARNs 的密钥 ID 和密钥

要获取 KMS 密钥的 ID 和 ARN，请使用操作。ListAliases

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

启用 KMS 密钥

要启用已禁用的 KMS 密钥，请使用EnableKey操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

禁用 KMS 密钥

要禁用 KMS 密钥，请使用DisableKey操作。禁用 KMS 密钥可防止其被使用。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用版本 3 加密和解密 Amazon KMS 数据密钥 适用于 PHP 的 Amazon SDK

数据密钥 是可用于加密数据的加密密钥，包括大量数据和其他数据加密密钥。

您可以使用 Amazon Key Management Service's (Amazon KMS) Amazon KMS key生成、加密和解密数据密钥。

以下示例演示如何：

- 使用 Encrypt 加密数据密钥。
- 使用 Decrypt 解密数据密钥。
- 使用ReEncrypt新的 KMS 密钥重新加密数据密钥。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用 Amazon Key Management Service (Amazon KMS) 的更多信息，请参阅《Amazon KMS 开发者指南》。

Encrypt

Encrypt 操作专用于加密数据密钥，但并不常用。GenerateDataKey和GenerateDataKeyWithoutPlaintext操作返回加密的数据密钥。将加密的数据移到新的 Amazon 区域并希望在新区域中使用 KMS 密钥来加密数据密钥时，可以使用 Encypt 方法。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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

要解密数据密钥，请使用 Decrypt 操作。

您指定的必须`ciphertextBlob`是GenerateDataKey、GenerateDataKeyWithoutPlaintext或 E ncrypt 响应中`CiphertextBlob`字段的值。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

重新加密

要解密加密的数据密钥，然后立即使用其他 KMS 密钥重新加密数据密钥，请使用操作。ReEncrypt这些操作完全在服务器端执行 Amazon KMS，因此它们永远不会将你的纯文本暴露在外面。 Amazon KMS

您指定的必须ciphertextBlob是GenerateDataKey、GenerateDataKeyWithoutPlaintext或 E ncrypt 响应中CiphertextBlob字段的值。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 使用 Amazon KMS 密钥策略

在创建 Amazon KMS key 时，您可以确定能够使用和管理该 KMS 密钥的人员。这些权限包含在名为密钥策略的文档中。您可以随时使用密钥策略添加、移除或修改客户托管 KMS 密钥的权限，但不能编辑托 Amazon 管 KMS 密钥的密钥策略。有关更多信息，请参阅 Amazon KMS的身份验证和访问控制。

以下示例演示如何：

- 使用列出密钥策略的名称ListKeyPolicies。
- 使用获取密钥策略GetKeyPolicy。
- 使用设置密钥策略PutKeyPolicy。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用 Amazon Key Management Service (Amazon KMS) 的更多信息，请参阅《Amazon KMS 开发者指南》。

列出所有密钥策略

要获取 KMS 密钥的密钥策略名称，请使用 `ListKeyPolicies` 操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

检索密钥策略

要获取 KMS 密钥的密钥策略，请使用 GetKeyPolicy 操作。

GetKeyPolicy 需要策略名称。除非您在创建 KMS 密钥时已创建密钥策略，否则，唯一有效的策略名称为默认值。要了解更多信息，请参阅 Amazon Key Management Service 开发人员指南中的默认密钥策略。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

设置密钥策略

要为 KMS 密钥建立或更改密钥策略，请使用 PutKeyPolicy 操作。

PutKeyPolicy 需要策略名称。除非您在创建 KMS 密钥时已创建密钥策略，否则，唯一有效的策略名称为默认值。要了解更多信息，请参阅 Amazon Key Management Service 开发人员指南中的默认密钥策略。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理授权

授权 是提供权限的另一种机制。它是密钥策略的替代形式。您可以使用授权来授予长期访问权限，允许 Amazon 委托人使用您的 Amazon Key Management Service (Amazon KMS) 客户Amazon KMS keys管理的权限。有关更多信息，请参阅 Amazon Key Management Service 开发人员指南中的Amazon KMS中的授权。

以下示例演示如何：

- 使用为 KMS 密钥创建授权CreateGrant。
- 使用查看 KMS 密钥的授权ListGrants。
- 使用取消对 KMS 密钥的授权RetireGrant。

- 使用RevokeGrant撤销 KMS 密钥的授权。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用 Amazon Key Management Service (Amazon KMS) 的更多信息，请参阅《Amazon KMS 开发者指南》。

创建授权

要为创建授权 Amazon KMS key，请使用CreateGrant操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

查看授权

要获取有关授予的详细信息 Amazon KMS key，请使用ListGrants操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

停用授权

要取消的授权 Amazon KMS key，请使用RetireGrant操作。在使用完授权后，请停用授权以将其清除。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

撤销授权

要撤消对的授权 Amazon KMS key，请使用RevokeGrant操作。您可以撤销授予，以显式拒绝依赖它的操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon KMS API 和 适用于 PHP 的 Amazon SDK 版本 3 处理别名

Amazon Key Management Service (Amazon KMS) 为Amazon KMS key被叫的别名提供可选的显示名称。

以下示例演示如何：

- 使用创建别名CreateAlias。
- 使用查看别名ListAliases。
- 使用更新别名UpdateAlias。
- 使用删除别名DeleteAlias。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

有关使用 Amazon Key Management Service (Amazon KMS) 的更多信息，请参阅《Amazon KMS 开发者指南》。

创建 别名

要为 KMS 密钥创建别名，请使用CreateAlias操作。别名在账户和 Amazon 地区中必须是唯一的。如果为已有别名的 KMS 密钥创建别名，`CreateAlias` 会为同一 KMS 密钥创建另一个别名。它不会替换现有别名。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

查看别名

要列出调用方 Amazon Web Services 账户 和中的所有别名 Amazon Web Services 区域，请使用ListAliases操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

更新别名

要将现有别名与其他 KMS 密钥关联，请使用UpdateAlias操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除别名

要删除别名，请使用DeleteAlias操作。删除别名不会影响底层 KMS 密钥。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon Kinesis 示例

Amazon Kinesis 是一项 Amazon 服务，可实时收集、处理和分析数据。使用 Amazon Kinesis Data Streams 来配置数据流，或者使用 Amazon Data Firehose 来将数据发送到 Amazon S3、OpenSearch Service、Amazon Redshift 或 Splunk。

有关 Kinesis 的更多信息，请参阅 Amazon Kinesis 文档。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

主题

- 使用 Kinesis Data Streams API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建数据流
- 使用 Kinesis Data Streams API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理数据分片
- 使用 Firehose API 和适用于 PHP 的 Amazon SDK 版本 3 来创建传输流

使用 Kinesis Data Streams API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建数据流

Amazon Kinesis Data Streams 允许发送实时数据。使用 Kinesis Data Streams 来创建数据创建器，该创建器在每次添加数据时将数据发送到配置的目的地。

有关更多信息，请参阅 Amazon Kinesis 开发人员指南中的创建和管理流。

以下示例演示如何：

- 使用 CreateAlias 创建数据流。
- 使用 DescribeStream 获取有关单个数据流的详细信息。
- 使用 ListStreams 列出现有数据流。
- 使用 PutRecord 将数据发送到现有数据流。
- 使用 DeleteStream 删除数据流。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon Kinesis 开发人员指南的更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南。

使用 Kinesis 数据流来创建数据流

建立 Kinesis 数据流，您可以使用以下代码示例来在其中发送要由 Kinesis 处理的信息。要了解更多信息，请参阅 Amazon Kinesis 开发人员指南中的创建和更新数据流。

要创建 Kinesis 数据流，请使用 CreateStream 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

检索数据流

获取有关使用以下代码示例的现有数据流的详细信息。默认情况下，这将返回有关连接到指定 Kinesis 数据流的前 10 个分片的信息。在将数据写入 Kinesis 数据流之前，请记住检查响应中的 `StreamStatus`。

要检索有关指定 Kinesis 数据流的详细信息，请使用 DescribeStream 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出已连接到 Kinesis 的现有数据流

列出所选 Amazon 区域中您 Amazon Web Services 账户 的前 10 个数据流。使用返回的 `HasMoreStreams 来确定是否有更多与您的账户关联的流。

要列出 Kinesis 数据流，请使用 ListStreams 操作。

导入。

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

```
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

将数据发送到现有数据流

在创建数据流后，请使用以下示例来发送数据。在向其发送数据之前，请使用 `DescribeStream` 检查数据 `StreamStatus` 是否处于活动状态。

要将单条数据记录写入 Kinesis 数据流，请使用 PutRecord 操作。要将最多 500 条记录写入 Kinesis 数据流，请使用 PutRecords 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除数据流

此示例演示如何删除数据流。删除数据流也会删除您发送到数据流的所有数据。活动 Kinesis 数据流切换到 DELETING 状态，直到完成流删除操作。在处于 DELETING 状态时，流将继续处理数据。

要删除 Kinesis 数据流，请使用 DeleteStream 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Kinesis Data Streams API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理数据分片

Amazon Kinesis Data Streams 允许向端点发送实时数据。数据流的速率取决于流中的分片数。

您可以每秒向单个分片写入 1000 条记录。每个分片的上传限制为每秒 1 MiB。用量按每个分片计算和收费，因此，请使用这些示例来管理流的数据容量和成本。

以下示例演示如何：

- 使用 ListShards 列出流中的分片。
- 使用 UpdateShardCount 增加或减少流中的分片数。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon Kinesis Data Streams 的更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南。

列出数据流分片

列出特定流中最多 100 个分片的详细信息。

要列出 Kinesis 数据流中的分片，请使用 ListShards 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

添加更多数据流分片

如果您需要更多数据流分片，则可增加当前分片数。我们建议您在增加时使分片计数加倍。这将生成当前可用的每个分片的副本以增加容量。在一个 24 小时的期间内，您只能将分片数加倍两次。

请记住，Kinesis Data Streams 用量的费用是按分片计算的，因此当需求减少时，我们建议将分片数量减半。删除分片时，只能将分片数量减少到当前分片计数的一半。

要更新 Kinesis 数据流的分片数，请使用 UpdateShardCount 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Firehose API 和适用于 PHP 的 Amazon SDK 版本 3 来创建传输流

Amazon Data Firehose 支持将实时数据发送到其他 Amazon 服务，包括 Amazon Kinesis Data Streams、Amazon S3、Amazon OpenSearch Service（OpenSearch Service）和 Amazon Redshift，或者发送到 Splunk。使用传输流创建一个数据创建器，以在您每次添加数据时将数据传送到配置的目的地。

以下示例演示如何：

- 使用 CreateDeliveryStream 创建传输流。
- 使用 DescribeDeliveryStream 获取有关单个传输流的详细信息。

- 使用 ListDeliveryStreams 列出传输流。
- 使用 PutRecord 将数据发送到传输流。
- 使用 DeleteDeliveryStream 删除传输流。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon Data Firehose 的更多信息，请参阅 Amazon Kinesis Data Firehose 开发人员指南。

使用 Kinesis 数据流来创建传输流

要建立将数据放入现有 Kinesis 数据流的传输流，请使用 CreateDeliveryStream 操作。

这使开发人员能够将现有 Kinesis 服务迁移到 Firehose。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon S3 存储桶来创建传输流

要建立将数据放入现有 Amazon S3 存储桶的传输流，请使用 CreateDeliveryStream 操作。

提供目标参数，如目标参数中所述。然后，确保向 Firehose 授予访问 Amazon S3 存储桶的权限，如向 Kinesis Data Firehose 授予访问 Amazon S3 目标的权限中所述。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 OpenSearch Service 来创建传输流

要建立将数据放入 OpenSearch Service 集群的 Firehose 传输流，请使用 CreateDeliveryStream 操作。

提供目标参数，如目标参数中所述。确保向 Firehose 授予访问 OpenSearch Service 集群的权限，如向 Kinesis Data Firehose 授予访问 Amazon ES 目标的权限中所述。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

检索传输流

要获取有关现有 Firehose 传输流的详细信息，请使用 DescribeDeliveryStream 操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出已连接到 Kinesis Data Streams 的现有传输流

要列出所有将数据发送到 Kinesis Data Streams 的现有 Firehose 传输流，请使用 ListDeliveryStreams 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出将数据发送到其他 Amazon 服务的现有传输流

要列出所有将数据发送到 Amazon S3、OpenSearch Service、Amazon Redshift，或者发送到 Splunk 的现有 Firehose 传输流，请使用 ListDeliveryStreams 操作。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

将数据发送到现有 Firehose 传输流

要通过 Firehose 传输流将数据发送到指定目的地，请在创建 Firehose 传输流之后使用 PutRecord 操作。

在将数据发送到 Firehose 传输流之前，请使用 DescribeDeliveryStream 来了解该传输流是否处于活动状态。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除 Firehose 传输流

要删除 Firehose 传输流，请使用 DeleteDeliveryStreams 操作。这也将删除您已发送到该传输流的所有数据。

导入

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

use Aws\Exception\AwsException;
```

示例代码

```
$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 使用 适用于 PHP 的 Amazon SDK 版本 3 的示例

AWS Elemental MediaConvert 是一种基于文件的视频转码服务，具有广播级功能。您可以使用它来创建用于通过互联网进行广播和 video-on-demand（VOD）交付的资产。有关更多信息，请参阅 AWS Elemental MediaConvert 《用户指南》。

的 PHP API 通过*AWS.MediaConvert*客户端类公开。 AWS Elemental MediaConvert 有关更多信息，请参阅《 API 参考》中的 `Class: AWS.MediaConvert`。

在中创建和管理转码作业 AWS Elemental MediaConvert

在此示例中，您使用 适用于 PHP 的 Amazon SDK 版本 3 调用 AWS Elemental MediaConvert 和创建转码作业。在开始之前，您需要将输入视频上传到为输入存储预配置的 Amazon S3 存储桶。有关支持的输入视频编解码器和容器的列表，请参阅 AWS Elemental MediaConvert 用户指南中的支持的输入编解码器和容器。

以下示例演示如何：

- 在中创建转码任务。 AWS Elemental MediaConvertCreateJob。
- 取消 AWS Elemental MediaConvert 队列中的转码作业。 CancelJob
- 检索已完成的转码任务的 JSON。 GetJob
- 检索最多 20 个最近创建的任务的 JSON 数组。 ListJobs

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

要访问 MediaConvert 客户端，请创建一个 IAM 角色来 AWS Elemental MediaConvert 访问您的输入文件和存储输出文件的 Amazon S3 存储桶。有关更多信息，请参阅 AWS Elemental MediaConvert 用户指南中的设置 IAM 权限。

创建客户端

适用于 PHP 的 Amazon SDK 通过创建 MediaConvert 客户端进行配置，其中包含您的代码所在的区域。在本示例中，区域设置为 us-west-2。

导入

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

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

示例代码

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

定义简单的转码任务

创建定义转码任务参数的 JSON。

这些参数有详细说明。您可以使用 AWS Elemental MediaConvert 控制台生成 JSON 作业参数，方法是在控制台中选择您的作业设置，然后选择作业部分底部的显示作业 JSON。本示例说明了简单作业的 JSON。

示例代码

```
$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"
    ]
];
```

创建作业

在创建任务参数 JSON 后，通过调用 AWS.MediaConvert service object 并传递参数，来调用 createJob 方法。所创建任务的 ID 在响应数据中返回。

示例代码

```
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";
}
```

检索任务

使用在调用 createjob 时返回的 JobID，您可以获取最近任务的 JSON 格式的详细描述。

示例代码

```
$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";
}
```

取消任务

使用在调用 createjob 时返回的 JobID，您可以取消仍在队列中的任务。您无法取消已开始转码的任务。

示例代码

```
$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";
}
```

列出最近的转码任务

创建包括值的参数 JSON，以指定是按升序还是降序对列表排序，要检查的任务队列的 ARN，以及要包含的任务的状态。这将返回最多 20 个任务。要检索后面的 20 个最近任务，请使用随结果返回的 nextToken 字符串。

示例代码

```
$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";
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 示例

Amazon Simple Storage Service (Amazon S3) 是提供高度可扩展的云存储的 Web 服务。Amazon S3 提供易于使用的对象存储，具有简单的 Web 服务接口，可用于从 Web 上的任何位置存储和检索任意规模的数据。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

主题

- 使用 适用于 PHP 的 Amazon SDK 版本 3 来创建和使用 Amazon S3 存储桶
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon S3 存储桶访问权限
- 使用 适用于 PHP 的 Amazon SDK 版本 3 来配置 Amazon S3 存储桶
- 在版本 3 中使用 Amazon S3 分段上 适用于 PHP 的 Amazon SDK 传
- 适用于 PHP 的 Amazon SDK 版本 3 的亚马逊 S3 预签名网址
- 使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 预签名 POST
- 使用 Amazon S3 存储桶作为具有 适用于 PHP 的 Amazon SDK 版本 3 的静态 Web 主机
- 结合使用 Amazon S3 存储桶策略与 适用于 PHP 的 Amazon SDK 版本 3
- 结合使用 S3 接入点 ARN 与 适用于 PHP 的 Amazon SDK 版本 3
- 将 Amazon S3 多区域接入点与 适用于 PHP 的 Amazon SDK 版本 3 配合使用

使用 适用于 PHP 的 Amazon SDK 版本 3 来创建和使用 Amazon S3 存储桶

以下示例演示如何：

- 使用 ListBuckets 返回已经过身份验证的请求发件人所拥有的存储桶列表。
- 使用 CreateBucket 创建新存储桶。
- 使用 PutObject 向存储桶添加对象。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

导入。

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

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

列出存储桶

使用以下代码创建 PHP 文件。首先创建用于指定 Amazon 区域和版本的 AWS.S3 客户端服务。然后调用 `listBuckets` 方法，它将以存储桶结构数组的形式返回请求发送者拥有的所有 Amazon S3 存储桶。

示例代码

```
$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";
}
```

创建存储桶

使用以下代码创建 PHP 文件。首先创建用于指定 Amazon 区域和版本的 AWS.S3 客户端服务。然后调用 `createBucket` 方法并将数组作为参数。唯一的必需字段是“Bucket”键，它包含了表示要创建的存储桶名称的字符串值。但是，您可以用“CreateBucketConfiguration”字段来指定 Amazon 区域。如果成功，此方法将返回存储桶的“Location”。

示例代码

```
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, 'amzn-s3-demo-bucket');
}

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

在存储桶中放置对象

要向新存储桶中添加文件，请使用以下代码创建一个 PHP 文件。

在命令行执行此文件，然后以字符串形式传入要将文件上传到的存储桶的名称，后跟要上传的文件的完整文件路径。

示例代码

```
$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";
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来管理 Amazon S3 存储桶访问权限

访问控制列表 (ACL) 是基于资源的访问策略选项之一，可用来管理对存储桶和对象的访问。可以使用 ACL 来向其他 Amazon 账户授予基本读/写权限。要了解更多信息，请参阅使用 ACL 管理访问。

以下示例演示如何：

- 使用 GetBucketAcl 获取存储桶的访问控制策略。
- 使用 PutBucketAcl 通过 ACL 设置存储桶权限。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

获取和设置访问控制列表策略

导入。

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

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

示例代码

```
// 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 = 'amzn-s3-demo-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";
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 来配置 Amazon S3 存储桶

跨源资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。借助 Amazon S3 中的 CORS 支持，您可以使用 Amazon S3 来构建各种富客户端 Web 应用程序，并选择性地允许跨源访问您的 Amazon S3 资源。

有关将 CORS 配置用于 Amazon S3 存储桶的更多信息，请参阅跨源资源共享 (CORS)。

以下示例演示如何：

- 使用 GetBucketCors 获取存储桶的 CORS 配置。
- 使用 PutBucketCors 设置用于存储桶的 CORS 配置。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

获取 CORS 配置

使用以下代码创建 PHP 文件。首先创建一个 AWS.S3 客户端服务，然后调用 `getBucketCors` 方法并指定使用所需 CORS 配置的存储桶。

需要的唯一参数是所选存储桶的名称。如果存储桶当前具有 CORS 配置，该配置将作为 CORSRules 对象由 Amazon S3 返回。

导入。

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

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

示例代码

```
$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());
}
```

设置 CORS 配置

使用以下代码创建 PHP 文件。首先创建一个 AWS.S3 客户端服务。然后，调用 `putBucketCors` 方法并指定要设置其 CORS 配置的存储桶，并将 CORS 配置设置为 CORSRules JSON 对象。

导入。

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

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

示例代码

```
$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());
}
```

在版本 3 中使用 Amazon S3 分段上 适用于 PHP 的 Amazon SDK 传

单独的 PutObject 操作可上传的对象大小上限为 5 GB。但使用分段上传方法（例如，CreateMultipartUpload、UploadPart、CompleteMultipartUpload、AbortMultipartUplo上传对象的大小范围可以在 5 MB 到 5 TB 之间。

以下示例演示如何：

- 使用将对象上传到 Amazon S3 ObjectUploader。
- 使用MultipartUploader为 Amazon S3 对象创建分段上传。
- 使用将对象从一个 Amazon S3 位置复制到另一个位置ObjectCopier。

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

对象上传程序

如果您不确定是 PutObject 还是 MultipartUploader 最适合该任务，请使用 ObjectUploader。ObjectUploader 是使用 PutObject 还是 MultipartUploader 将大型文件上传到 Amazon S3，取决于有效载荷大小。

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

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

示例代码

```
// 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);
```

配置

ObjectUploader 对象构造函数接受以下参数：

$client

用于执行转移的 Aws\ClientInterface 对象。它应是 Aws\S3\S3Client 的实例。

$bucket

（string，必需）对象要上传到的存储桶名称。

$key

（string，必需）上传对象使用的键。

$body

（mixed，必需）要上传的对象数据。可以是 StreamInterface、PHP 流资源或要上传的数据字符串。

$acl

(string) 上传对象设置的访问控制列表 (ACL)。默认情况下对象是私有的。

$options

用于分段上传的配置选项的关联数组。以下配置选项有效：

add_content_md5

(bool) 设置为 true 可自动计算上传的 MD5 校验和。

mup_threshold

（int，默认：int(16777216)）文件大小的字节数。如果文件大小超过此限制，则使用分段上传。

before_complete

(callable) 在 CompleteMultipartUpload 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。有关可以添加到CommandInterface对象的参数，请参阅 CompleteMultipartUpload API 参考。

before_initiate

(callable) 在 CreateMultipartUpload 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。如果文件大小超过 mup_threshold 值，SDK 会调用此回调。有关可以添加到CommandInterface对象的参数，请参阅 CreateMultipartUpload API 参考。

before_upload

(callable) 在任何 PutObject 或 UploadPart 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。如果文件大小小于或等于 mup_threshold 值，SDK 会调用此回调。有关您可以应用于PutObject请求的参数，请参阅 PutObject API 参考。有关适用于UploadPart请求的参数，请参阅 UploadPart API 参考。SDK 会忽略任何不适用于 CommandInterface 对象所表示操作的参数。

concurrency

（int，默认：int(3)）在分段上传期间允许的并发 UploadPart 操作数量上限。

part_size

（int，默认：int(5242880)）进行分段上传时使用的分段大小（以字节为单位）。此值必须在 5 MB（含）和 5 GB（含）之间。

state

(Aws\Multipart\UploadState) 表示分段上传状态的对象，用于重新开始上次上传。如果提供了此选项，将忽略 $bucket 和 $key 参数以及 part_size 选项。

params

一个关联数组，用于为每个子命令提供配置选项。例如：

```
new ObjectUploader($bucket, $key, $body, $acl, ['params' => ['CacheControl'
 => <some_value>])
```

MultipartUploader

分段上传可改善较大对象的上传体验。这种方法支持您将对象分成各自独立的部分，既可以按任何顺序上传，也可并行上传。

我们鼓励 Amazon S3 客户针对大于 100 MB 的对象使用分段上传。

MultipartUploader 对象

开发工具包中包含一个特殊的 `MultipartUploader` 对象，可简化分段上传过程。

导入

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

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

示例代码

```
$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";
}
```

这个上传工具可根据提供的源和配置创建分段数据的生成器，并尝试上传所有分段。如果某些分段上传失败，上传工具将继续上传后面的分段，直到所有源数据均被读取。然后，上传工具重新尝试上传失败的分段，或引发包含有关无法上传的分段的信息的异常。

自定义分段上传

您可以设置 CreateMultipartUpload、UploadPart 和 CompleteMultipartUpload 操作的自定义选项，分段上传工具可通过传递给构造函数的回调执行这些操作。

导入

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

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

示例代码

```
// 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';
    },
]);
```

分段上传之间的手动垃圾回收

如果您达到大型上传的内存限制，这可能是由于在达到您的内存限制时开发工具包生成的、但尚未由 PHP 垃圾回收器收集的循环引用导致的。在操作之间手动调用收集算法可允许在达到该限制之前收集循环。以下示例在每个分段上传之前使用回调来调用收集算法。请注意，调用垃圾回收器会降低性能，最佳用法将取决于您的使用案例和环境。

```
$uploader = new MultipartUploader($client, $source, [
   'bucket' => 'your-bucket',
   'key' => 'your-key',
   'before_upload' => function(\Aws\Command $command) {
      gc_collect_cycles();
   }
]);
```

从错误中恢复

如果在分段上传过程中发生错误，将引发 MultipartUploadException。可通过此异常访问 UploadState 对象，其中包含分段上传的进度信息。可使用 UploadState 重新开始未完成的上传。

导入

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

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

示例代码

```
// 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);
}
```

若通过 UploadState 继续上传，将尝试上传尚未上传的分段。状态对象会跟踪缺少的分段，即使这些分段是不连续的。上传工具会读取或搜寻提供的源文件，找到仍需上传的特定分段的字节范围。

UploadState 对象是序列化的，因此您也可以在另一进程中重新开始上传。即使不是在处理异常，您也可以通过调用 $uploader->getState() 获得 UploadState 对象。

> **Important**
>
> 作为源传递到 `MultipartUploader` 的流在上传之前不会自动倒回。如果您在与上个示例类似的循环中使用流，而不是文件路径，请重置 `catch` 块中的 `$source` 变量。

导入

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

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

示例代码

```
// 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);
```

中止分段上传

通过检索 UploadState 对象中包含的 UploadId 并将其传递给 abortMultipartUpload 可以中止分段上传。

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

异步分段上传

在 upload() 上调用 MultipartUploader 是一个阻止请求。如果在异步环境中工作，可获得分段上传的 Promise。

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

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

示例代码

```
// 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();
```

配置

MultipartUploader 对象构造函数接受以下参数：

$client

用于执行转移的 Aws\ClientInterface 对象。它应是 Aws\S3\S3Client 的实例。

$source

上传的源数据。这可以是路径或 URL（例如，/path/to/file.jpg）、资源处理（例如，fopen('/path/to/file.jpg', 'r')）或 PSR-7 流的实例。

$config

用于分段上传的配置选项的关联数组。

以下配置选项有效：

acl

(string) 上传对象设置的访问控制列表 (ACL)。默认情况下对象是私有的。

before_complete

(callable) 在 CompleteMultipartUpload 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。

before_initiate

(callable) 在 CreateMultipartUpload 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。

before_upload

(callable) 在任何 UploadPart 操作之前调用的回调。此回调应具有类似 function (Aws\Command $command) {...} 的函数签名。

bucket

（string，必需）对象要上传到的存储桶名称。

concurrency

（int，默认：int(5)）在分段上传期间允许的并发 UploadPart 操作数量上限。

key

（string，必需）上传对象使用的键。

part_size

（int，默认：int(5242880)）进行分段上传时使用的分段大小（以字节为单位）。它必须在 5 MB（含）和 5 GB（含）之间。

state

(Aws\Multipart\UploadState) 表示分段上传状态的对象，用于重新开始上次上传。如果提供了此选项，将忽略 bucket、key 和 part_size 选项。

add_content_md5

(boolean) 设置为 true 可自动计算上传的 MD5 校验和。

params

一个关联数组，用于为每个子命令提供配置选项。例如：

```
new MultipartUploader($client, $source, ['params' => ['CacheControl'
 => <some_value>]])
```

分段副本

适用于 PHP 的 Amazon SDK 还包括一个MultipartCopy对象，该对象的使用方式与类似MultipartUploader，但设计用于在 Amazon S3 中复制大小在 5 GB 到 5 TB 之间的对象。

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

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

示例代码

```
// 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";
}
```

适用于 PHP 的 Amazon SDK 版本 3 的亚马逊 S3 预签名网址

您可以通过传递请求信息作为查询字符串参数，而不是使用授权 HTTP 标头来验证特定类型的请求。这在允许第三方浏览器直接访问您的私有 Amazon S3 数据，而无需代理请求时非常有用。其概念是构造一个“预签名”的请求并将其编码为另一个用户可以使用的 URL。此外，您还可以通过指定过期时间来限制预签名请求。

为 HTTP GET 请求创建预签名 URL

以下代码示例演示了如何使用适用于 PHP 的 SDK 为 HTTP GET 请求创建预签名 URL。

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$s3Client = new S3Client([
    'region' => 'us-west-2',
]);

// Supply a CommandInterface object and an expires parameter to the
 `createPresignedRequest` method.
$request = $s3Client->createPresignedRequest(
    $s3Client->getCommand('GetObject', [
        'Bucket' => 'amzn-s3-demo-bucket',
        'Key' => 'demo-key',
```

```
    ]),
    '+1 hour'
);

// From the resulting RequestInterface object, you can get the URL.
$presignedUrl = (string) $request->getUri();

echo $presignedUrl;
```

createPresignedRequest 方法的 API 参考提供了更多详细信息。

其他人可以在接下来的一个小时内使用 $presignedUrl 值来检索对象。当发出 HTTP GET 请求时（例如使用浏览器），在 S3 服务看来，该调用来自创建了预签名 URL 的用户。

为 HTTP PUT 请求创建预签名 URL

以下代码示例演示了如何使用适用于 PHP 的 SDK 为 HTTP PUT 请求创建预签名 URL。

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$s3Client = new S3Client([
    'region' => 'us-west-2',
]);

$request = $s3Client->createPresignedRequest(
    $s3Client->getCommand('PutObject', [
        'Bucket' => 'amzn-s3-demo-bucket',
        'Key' => 'demo-key',
    ]),
    '+1 hour'
);

// From the resulting RequestInterface object, you can get the URL.
$presignedUrl = (string) $request->getUri();
```

其他人现在可以在 HTTP PUT 请求中使用预签名 URL 来上传文件：

```
use GuzzleHttp\Psr7\Request;
use GuzzleHttp\Psr7\Response;
```

```
// ...

function uploadWithPresignedUrl($presignedUrl, $filePath, $s3Client): ?Response
{
    // Get the HTTP handler from the S3 client.
    $handler = $s3Client->getHandlerList()->resolve();

    // Create a stream from the file.
    $fileStream = new Stream(fopen($filePath, 'r'));

    // Create the request.
    $request = new Request(
        'PUT',
        $presignedUrl,
        [
            'Content-Type' => mime_content_type($filePath),
            'Content-Length' => filesize($filePath)
        ],
        $fileStream
    );

    // Send the request using the handler.
    try {
        $promise = $handler($request, []);
        $response = $promise->wait();
        return $response;
    } catch (Exception $e) {
        echo "Error uploading file: " . $e->getMessage() . "\n";
        return null;
    }
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 预签名 POST

与预签名 URL 极其相似，预签名 POST 使您无需向用户提供 Amazon 凭证便可向其授予写入访问权限。可以在 AwsS3PostObjectV4 实例的帮助下创建预签名 POST 表单。

以下示例演示如何：

- 使用 PostObjectV4 获取 S3 Object POST 上传的数据。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

> **Note**
>
> PostObjectV4 不能使用来自 Amazon IAM Identity Center 的凭证。

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建 PostObjectV4

要创建 PostObjectV4 的实例，必须提供以下内容：

- Aws\S3\S3Client 的实例
- 桶
- 表单输入字段的关联数组
- 一系列策略条件（请参阅 Amazon Simple Storage Service 用户指南中的策略构建）
- 策略的过期时间字符串（可选，默认为 1 小时）。

导入。

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

use Aws\S3\PostObjectV4;
use Aws\S3\S3Client;
```

示例代码

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

use Aws\S3\PostObjectV4;
use Aws\S3\S3Client;

$client = new S3Client([
    'profile' => 'default',
    'region' => 'us-east-1',
]);
$bucket = 'amzn-s3-demo-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>
```

使用 Amazon S3 存储桶作为具有 适用于 PHP 的 Amazon SDK 版本 3 的静态 Web 主机

您可以在 Amazon S3 上托管静态网站。要了解更多信息，请参阅在 Amazon S3 上托管静态网站。

以下示例演示如何：

- 使用 GetBucketWebsite 获取存储桶的网站配置。
- 使用 PutBucketWebsite 设置存储桶的网站配置。
- 使用 DeleteBucketWebsite 从存储桶中删除网站配置。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证。请参阅 适用于 PHP 的 Amazon SDK 版本 3 的凭证。

获取、设置和删除存储桶的网站配置。

导入。

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

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

示例代码

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

// Retrieving the Bucket Website Configuration
$bucket = 'amzn-s3-demo-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";
}
```

结合使用 Amazon S3 存储桶策略与 适用于 PHP 的 Amazon SDK 版本 3

您可以使用存储桶策略来授予对 Amazon S3 资源的权限。有关更多信息，请参阅使用存储桶策略和用户策略。

以下示例演示如何：

- 使用 GetBucketPolicy 返回用于指定存储桶的策略。
- 使用 PutBucketPolicy 替换存储桶策略。
- 使用 DeleteBucketPolicy 从存储桶中删除策略。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

获取、删除和替换存储桶策略

导入。

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

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

示例代码

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

$bucket = 'amzn-s3-demo-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";
}
```

结合使用 S3 接入点 ARN 与 适用于 PHP 的 Amazon SDK 版本 3

S3 引入了接入点，这是与 S3 存储桶交互的一种新方式。接入点上可以应用唯一的策略和配置，而不是直接应用到存储桶。通过 适用于 PHP 的 Amazon SDK，您可以在存储桶字段中使用接入点 ARN 进行 API 操作，而不是明确指定存储桶名称。有关 S3 接入点和 ARN 工作原理的更多详细信息，可在此处找到。以下示例演示如何：

- 将 GetObject 与接入点 ARN 一起使用，从存储桶中提取对象。
- 将 PutoBject 与接入点 ARN 一起使用，将对象添加到存储桶中。
- 将 S3 客户端配置为使用 ARN 区域而不是客户端区域。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

导入。

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

use Aws\S3\S3Client;
```

获取对象

首先创建用于指定 Amazon 区域和版本的 AWS.S3 客户端服务。然后 getObject 使用您的密钥并在 Bucket 字段中使用 S3 接入点 ARN 调用该方法，以从与该接入点关联的存储桶中获取对象。

示例代码

```
$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'
]);
```

在存储桶中放置对象

首先创建用于指定 Amazon 区域和版本的 AWS.S3 客户端服务。然后使用所需的密钥、正文或源文件并在 putObject 字段中使用 S3 接入点 ARN 调用 Bucket 方法，这会将对象放入与该接入点关联的存储桶中。

示例代码

```
$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'
]);
```

将 S3 客户端配置为使用 ARN 区域而不是客户端区域

在 S3 客户端操作中使用 S3 接入点 ARN 时，默认情况下，客户端将确保 ARN 区域与客户端区域匹配，如果不匹配则引发异常。通过将 `use_arn_region` 配置选项设置为 `true`，可将此行为更改为接受 ARN 区域而不是客户端区域。默认情况下，该选项设置为 `false`。

示例代码

```
$s3 = new S3Client([
    'version'        => 'latest',
    'region'         => 'us-west-2',
    'use_arn_region' => true
]);
```

客户端还将按以下优先顺序检查环境变量和配置文件选项：

1. 客户端选项 `use_arn_region`，如上例所示。
2. 环境变量 `AWS_S3_USE_ARN_REGION`

```
export AWS_S3_USE_ARN_REGION=true
```

1. Amazon 共享配置文件（默认情况下位于 `~/.aws/config` 中）中的配置变量 `s3_use_arn_region`。

```
[default]
s3_use_arn_region = true
```

将 Amazon S3 多区域接入点与 适用于 PHP 的 Amazon SDK 版本 3 配合使用

Amazon Simple Storage Service（S3）多区域接入点提供了一个全局端点，用于在 Amazon Web Services 区域之间路由 Amazon S3 请求流量。

您可以使用适用于 PHP 的 SDK、另一个 Amazon SDK、S3 控制台或 Amazon CLI 来创建多区域接入点，

> **Important**
>
> 要将多区域接入点与适用于 PHP 的 SDK 配合使用，您的 PHP 环境必须安装Amazon 通用运行时 (Amazon CRT) 扩展。

当创建多区域接入点时，Amazon S3 会生成一个具有以下格式的 Amazon 资源名称（ARN）：

`arn:aws:s3::account-id:accesspoint/MultiRegionAccessPoint_alias`

您可以使用生成的 ARN 来代替 getObject() 和 putObject() 方法的存储桶名称。

```
<?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";
```

使用 Secrets Manager API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理密钥

Amazon Secrets Manager 存储和管理共享密钥，如密码、API 密钥和数据库凭证。借助 Secrets Manager 服务，开发人员可以将已部署代码中的硬编码的凭证替换为对 Secrets Manager 的嵌入式调用。

Secrets Manager 原生支持对 Amazon Relational Database Service (Amazon RDS) 数据库的自动计划凭证轮换，从而增强应用程序安全性。Secrets Manager 还可以使用 Amazon Lambda 来无缝轮换其他数据库和第三方服务的密钥，以实现特定于服务的详细信息。

以下示例演示如何：

- 使用 CreateSecret 创建密钥。
- 使用 GetSecretValue 检索密钥。
- 使用 ListSecrets 列出 Secrets Manager 存储的所有密钥。
- 使用 DescribeSecret 获取有关指定密钥的详细信息。
- 使用 PutSecretValue 更新指定密钥。
- 使用 RotateSecret 设置密钥轮换。
- 使用 DeleteSecret 标记密钥以进行删除。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

在 Secrets Manager 中创建密钥

要在 Secrets Manager 中创建密钥，请使用 CreateSecret 操作。

在此示例中，用户名和密码将存储为 JSON 字符串。

导入。

```
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

从 Secrets Manager 中检索密钥

要检索存储在 Secrets Manager 中的密钥值，请使用 GetSecretValue 操作。

在此示例中，`secret` 是一个包含存储值的字符串。如果 `username` 的值为 `<<USERNAME>>`，并且 `password` 的值为 `<<PASSWORD>>`，则 `secret` 的输出为：

```
{"username":"<<USERNAME>>","password":"<<PASSWORD>>"}
```

用 json_decode($secret, true) 来访问数组值。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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;
```

列出 Secrets Manager 中存储的密钥

使用 ListSecrets 操作获取由 Secrets Manager 存储的所有密钥的列表。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

检索有关密钥的详细信息

存储的密钥包含有关轮换规则、上次访问或更改密钥、用户创建的标签和 Amazon 资源名称 (ARN) 的元数据。要获取存储在 Secrets Manager 中的指定密钥的详细信息，请使用 DescribeSecret 操作。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

更新密钥值

要将新加密的密钥值存储在 Secrets Manager 中，请使用 PutSecretValue 操作。

这将创建密钥的一个新版本。如果密钥的某个版本已存在，则在 `VersionStages` 中添加带该值的 `AWSCURRENT` 参数，以确保在检索该值时使用新值。

导入。

```
require 'vendor/autoload.php';
use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

将值轮换到 Secrets Manager 中的现有密钥

要轮换存储在 Secrets Manager 中的现有密钥的值，请使用 Lambda 轮换函数和 RotateSecret 操作。

在开始之前，请创建一个 Lambda 函数来轮换您的密钥。Amazon 代码示例目录目前包含用于轮换 Amazon RDS 数据库凭证的多个 Lambda 代码示例。

> **Note**
>
> 有关轮换密钥的更多信息，请参阅 Amazon Secrets Manager 用户指南中的轮换 Amazon Secrets Manager 密钥。

设置您的 Lambda 函数后，配置一个新的密钥轮换。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

配置轮换时，您可以使用 RotateSecret 操作实施轮换。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

从 Secrets Manager 中删除密钥

要从 Secrets Manager 中删除指定密钥，请使用 DeleteSecret 操作。要防止意外删除密钥，将自动为密钥添加 DeletionDate 戳，用于指定您可恢复删除的恢复时间范围。如果未指定恢复时间窗口的范围，则默认时间范围为 30 天。

导入。

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

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

相关信息

适用于 PHP 的 Amazon SDK 示例使用 Amazon Secrets Manager API 参考中的以下 REST 操作：

- CreateSecret
- GetSecretValue

- ListSecrets
- DescribeSecret
- PutSecretValue
- ： RotateSecret
- DeleteSecret

有关使用 Amazon Secrets Manager 的更多信息，请参阅 Amazon Secrets Manager 用户指南。

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SES 示例

Amazon Simple Email Service (Amazon SES) 是一个易于使用且经济的电子邮件平台，便于您通过该平台，使用您自己的电子邮件地址和域来发送或接收电子邮件。有关 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

Amazon 提供了两个 Amazon SES 服务版本，相应地，适用于 PHP 的 SDK 提供了两个客户端版本：SesClient 和 SesV2Client。尽管方法调用方式或结果可能有所不同，但在很多情况下，客户端的功能是重叠的。这两个 API 还提供专有功能，因此您可以使用两个客户端来访问所有功能。

本部分中的示例都使用原始 `SesClient`。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

主题

- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来验证电子邮件身份
- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建自定义电子邮件模板
- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理电子邮件筛选条件
- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建和管理电子邮件规则
- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来监控发送活动
- 使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来向发件人授权

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来验证电子邮件身份

当您首次开始使用 Amazon Simple Email Service (Amazon SES) 账户时，您将发送电子邮件到的同一个 Amazon 区域中的所有发件人和收件人均必须经过验证。有关发送电子邮件的更多信息，请参阅使用 Amazon SES 发送电子邮件。

以下示例演示如何：

- 使用 VerifyEmailIdentity 验证电子邮件地址。
- 使用 VerifyDomainIdentity 验证电子邮件域。
- 使用 ListIdentities 列出所有电子邮件地址。
- 使用 ListIdentities 列出所有电子邮件域。
- 使用 DeleteIdentity 删除电子邮件地址。
- 使用 DeleteIdentity 删除电子邮件域。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

验证电子邮件地址

Amazon SES 只能从已验证的电子邮件地址或域发送电子邮件。通过验证电子邮件地址，您可证明您是该地址的所有者，并希望允许 Amazon SES 从该地址发送电子邮件。

运行以下代码示例时，Amazon SES 将向您指定的地址发送一封电子邮件。当您（或电子邮件的收件人）单击电子邮件中的链接时，该地址将得到验证。

要将电子邮件地址添加到您的 Amazon SES 账户，请使用 VerifyEmailIdentity 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

验证电子邮件域

Amazon SES 只能从已验证的电子邮件地址或域发送电子邮件。通过验证域，您可证明自己是该域的所有者。在验证域时，允许 Amazon SES 从该域上的任何地址发送电子邮件。

当运行以下代码示例时，Amazon SES 向您提供验证令牌。您必须将令牌添加到域的 DNS 配置。有关更多信息，请参阅 Amazon Simple Email Service 开发人员指南中的使用 Amazon SES 来验证域。

要将发送域添加到您的 Amazon SES 账户，请使用 VerifyDomainIdentity 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出电子邮件地址

要检索在当前 Amazon 区域中提交的电子邮件地址的列表，不论验证状态如何，请使用 ListIdentities 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出电子邮件域

要检索在当前 Amazon 区域中提交的电子邮件域的列表，不论验证状态如何，请使用 ListIdentities 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除电子邮件地址

要从身份列表中删除某个经过验证的电子邮件地址，请使用 DeleteIdentity 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除电子邮件域

要从经过验证的身份列表中删除某个经过验证的电子邮件地址，请使用 DeleteIdentity 操作。

导入。

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

```
use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建自定义电子邮件模板

Amazon Simple Email Service (Amazon SES) 让您可以通过使用模板，向各个收件人发送个性化的电子邮件。模板包含一个主题行以及电子邮件正文的文本和 HTML 部分。主题和正文部分还可包含针对每个收件人进行个性化设置的唯一值。

有关更多信息，请参阅 Amazon Simple Email Service 开发人员指南中的使用 Amazon SES 来发送个性化电子邮件。

以下示例演示如何：

- 使用 CreateTemplate 创建电子邮件模板。
- 使用 ListTemplates 列出所有电子邮件模板。
- 使用 CreateTemplate 检索电子邮件模板。

- 使用 UpdateTemplate 更新电子邮件模板。
- 使用 DeleteTemplate 删除电子邮件模板。
- 使用 SendTemplatedEmail 发送模板化电子邮件。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

创建电子邮件模板

要创建模板以发送个性化电子邮件，请使用 CreateTemplate 操作。在添加模板到的 Amazon 区域中，模板可由授权发送消息的任意账户使用。

> **Note**
>
> Amazon SES 不会验证您的 HTML，因此请先确认 HtmlPart 有效，然后再发送电子邮件。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

获取电子邮件模板

要查看现有电子邮件模板（包括主题行、HTML 正文和纯文本）的内容，请使用 GetTemplate 操作。只有 TemplateName 为必填。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出所有电子邮件模板

要检索当前 Amazon 区域中与您的 Amazon Web Services 账户 关联的所有电子邮件模板，请使用 ListTemplates 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

更新电子邮件模板

要更改特定电子邮件模板（包括主题行、HTML 正文和纯文本）的内容，请使用 UpdateTemplate 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除电子邮件模板

要删除特定电子邮件模板，请使用 DeleteTemplate 操作。您只需要 TemplateName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用模板发送电子邮件

要使用模板向收件人发送电子邮件，请使用 SendTemplatedEmail 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来管理电子邮件筛选条件

除了发送电子邮件外，您还可以使用 Amazon Simple Email Service (Amazon SES) 来接收电子邮件。IP 地址筛选条件让您能够选择指定是接受还是拒绝来自某个 IP 地址或 IP 地址范围的邮件。有关更多信息，请参阅为 Amazon SES 电子邮件接收管理 IP 地址筛选条件。

以下示例演示如何：

- 使用 CreateReceiptFilter 创建电子邮件筛选条件。
- 使用 ListReceiptFilters 列出所有电子邮件筛选条件。
- 使用 DeleteReceiptFilter 删除电子邮件筛选条件。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

创建电子邮件筛选条件

要允许或阻止来自特定 IP 地址的电子邮件，请使用 CreateReceiptFilter 操作。提供 IP 地址或地址范围以及唯一名称来标识此筛选条件。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出所有电子邮件筛选条件

要列出当前 Amazon 区域中与您的 Amazon Web Services 账户 关联的 IP 地址筛选条件，请使用 ListReceiptFilters 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除电子邮件筛选条件

要删除特定 IP 地址的现有筛选条件，请使用 DeleteReceiptFilter 操作。提供唯一筛选条件名称用于标识要删除的接收筛选条件。

如果您需要更改所筛选的地址范围，可以删除接收筛选条件并创建新的筛选条件。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来创建和管理电子邮件规则

除了发送电子邮件外，您还可以使用 Amazon Simple Email Service (Amazon SES) 来接收电子邮件。您可以通过接收规则来指定 Amazon SES 在为您拥有的电子邮件地址或域接收电子邮件后，将对这些邮件执行哪些操作。规则可以将电子邮件发送到其他 Amazon 服务，包括但不限于 Amazon S3、Amazon SNS 或 Amazon Lambda。

有关更多信息，请参阅为 Amazon SES 电子邮件接收管理接收规则集和为 Amazon SES 电子邮件接收管理接收规则。

以下示例演示如何：

- 使用 CreateReceiptRuleSet 创建接收规则集。
- 使用 CreateReceiptRule 创建接收规则。
- 使用 DescribeReceiptRuleSet 描述接收规则集。
- 使用 DescribeReceiptRule 描述接收规则。
- 使用 ListReceiptRuleSets 列出所有接收规则集。
- 使用 UpdateReceiptRule 更新接收规则。
- 使用 DeleteReceiptRule 删除接收规则。
- 使用 DeleteReceiptRuleSet 删除接收规则集。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

创建接收规则集

接收规则集包含接收规则的集合。您必须至少有一个接收规则集与您的账户关联，然后才能创建接收规则。要创建接收规则集，请提供唯一 RuleSetName，然后使用 CreateReceiptRuleSet 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

创建接收规则

通过添加接收规则到现有接收规则集，控制传入电子邮件。此示例展示了如何创建将传入消息发送到 Amazon S3 存储桶的接收规则，不过您也可以将消息发送到 Amazon SNS 和 Amazon Lambda。要创建接收规则，请向 CreateReceiptRule 操作提供规则和 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
```

```
}
```

描述接收规则集

每秒一次，返回指定接收规则集的详细信息。要使用 DescribeReceiptRuleSet 操作，请提供 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

描述接收规则

返回指定接收规则的详细信息。要使用 DescribeReceiptRule 操作，请提供 RuleName 和 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

列出所有接收规则集

要列出当前 Amazon 区域中您 Amazon Web Services 账户 下存在的接收规则集，请使用 ListReceiptRuleSets 操作。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

更新接收规则

此示例展示了如何更新将传入消息发送到 Amazon Lambda 函数的接收规则，不过您也可以将消息发送到 Amazon SNS 和 Amazon S3。要使用 UpdateReceiptRule 操作，请提供新接收规则和 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除接收规则集

删除当前未禁用的指定接收规则集。这还会删除其中包含的所有接收规则。要删除接收规则集，请向 DeleteReceiptRuleSet 操作提供 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

删除接收规则

要删除指定的接收规则，请向 DeleteReceiptRule 操作提供 RuleName 和 RuleSetName。

导入。

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

use Aws\Exception\AwsException;
```

示例代码

```
$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";
}
```

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来监控发送活动

Amazon Simple Email Service (Amazon SES) 提供了监控发送活动的方法。我们建议您实现这些方法，以便您可以跟踪重要的指标，如您账户的邮件退回率、投诉率和拒绝率。过高的退回邮件率和投诉率可能会影响您使用 Amazon SES 发送电子邮件的能力。

以下示例演示如何：

- 使用 GetSendQuota 检查您的发送配额。
- 使用 GetSendStatistics 监控您的发送活动。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

检查发送配额

在单个 24 小时期间，您只能发送特定数量的邮件。要检查仍然允许您发送多少邮件，请使用 GetSendQuota 操作。有关更多信息，请参阅 管理 Amazon SES 发送限制。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

监控发送活动

要检索您在过去两周内发送的邮件的指标，请使用 GetSendStatistics 操作。此示例以 15 分钟为增量，返回传送尝试的次数、退回邮件数、投诉邮件数和拒绝邮件数。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

使用 Amazon SES API 和 适用于 PHP 的 Amazon SDK 版本 3 来向发件人授权

要允许其他 Amazon Web Services 账户、Amazon Identity and Access Management 用户或 Amazon 服务可代表您通过 Amazon Simple Email Service (Amazon SES) 发送电子邮件，您需要创建发送授权策略。这是一个附加到您拥有的身份的 JSON 文档。

该策略明确列出您允许哪些人、在何种条件下使用该身份。除了您以及在策略中明确授予权限的实体之外，所有其他发件人不允许发送电子邮件。一个身份可以不附加策略、附加一个策略或附加多个策略。此外，您还可以使用一个包含多个语句的策略来实现多个策略的效果。

有关更多信息，请参阅使用 Amazon SES 的发送授权。

以下示例演示如何：

- 使用 PutIdentityPolicy 创建授权发件人。
- 使用 GetIdentityPolicies 检索授权发件人的策略。
- 使用 ListIdentityPolicies 列出授权发件人。
- 使用 DeleteIdentityPolicy 撤销授权发件人的权限。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

有关使用 Amazon SES 的更多信息，请参阅 Amazon SES 开发人员指南。

创建授权发件人

要授权其他 Amazon Web Services 账户 代表您发送电子邮件，请使用身份授权策略添加或更新授权，以允许从您经过验证的电子邮件地址或域发送电子邮件。要创建身份策略，请使用 PutIdentityPolicy 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

检索授权发件人的策略

返回与特定电子邮件身份或域身份关联的发送授权策略。要获取指定电子邮件地址或域的发送授权，请使用 GetIdentityPolicy 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

列出授权发件人

在当前 Amazon 区域中，要列出与特定电子邮件身份或域关联的发送授权策略，请使用 ListIdentityPolicies 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

撤销授权发件人的权限

使用 DeleteIdentityPolicy 操作，通过删除关联的身份策略，删除其他 Amazon Web Services 账户 通过电子邮件身份或域身份发送电子邮件的发送授权。

导入。

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

use Aws\Exception\AwsException;
use Aws\Ses\SesClient;
```

示例代码

```
$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";
}
```

使用版本 3 的亚马逊 SNS 示 适用于 PHP 的 Amazon SDK 例

Amazon Simple Notification Service (Amazon SNS) 是一项 Web 服务，用于协调和管理向订阅端点或客户端交付或发送消息的过程。

在 Amazon SNS 中有两种类型的客户端：发布者（也称为创建者）和订阅用户（也称为用户）。发布者通过创建消息并将消息发送至主题与订阅者进行异步交流，主题是一个逻辑访问点和通信渠道。订阅者（Web 服务器、电子邮件地址、Amazon SQS 队列、 Amazon Lambda 函数）在订阅主题时通过支持的协议之一（Amazon SQS HTTP/HTTPS URLs、、电子邮件 Amazon SMS、Lambda）使用或接收消息或通知。

适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码都可以在此处找到 GitHub。

主题

- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中管理主题
- 使用 适用于 PHP 的 Amazon SDK 版本 3 在 Amazon SNS 中管理订阅
- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中发送 SMS 消息

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中管理主题

要将通知发送到 Amazon Simple Queue Service (Amazon SQS)、HTTP/HTTPS URL、电子邮件、Amazon SMS 或 Amazon Lambda，您必须首先创建主题，以管理该主题的任何订阅用户的消息传送。

对于观察程序设计模式，此主题 (Topic) 类似于该主题 (Subject)。创建主题后，您可以添加在有消息发布到该主题时将自动通知的订阅者。

在在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中管理订阅中了解有关订阅主题的更多信息。

以下示例演示如何：

- 使用 CreateTopic 创建将通知发布到的主题。
- 使用 ListTopics 返回请求者的主题列表。
- 使用 DeleteTopic 删除主题及其所有订阅。
- 使用 GetTopicAttributes 返回主题的所有属性。
- 使用 SetTopicAttributes 允许主题所有者将主题的属性设置为新值。

有关使用 Amazon SNS 的更多信息，请参阅用于消息传输状态的 Amazon SNS 主题属性。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

创建主题

要创建主题，请使用 CreateTopic 操作。

您 Amazon Web Services 账户 中的各个主题名称必须唯一。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

列出主题

要列出当前 Amazon 区域中的最多 100 个现有主题，请使用 ListTopics 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

删除主题

要删除现有主题及其所有订阅，请使用 DeleteTopic 操作。

尚未传送至订阅者的任何消息也将删除。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

获取主题属性

要检索单个现有主题的属性，请使用 GetTopicAttributes 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

设置主题属性

要更新单个现有主题的属性，请使用 SetTopicAttributes 操作。

您只能设置 Policy、DisplayName 和 DeliveryPolicy 属性。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 在 Amazon SNS 中管理订阅

使用亚马逊简单通知服务 (Amazon SNS) Simple Notification 主题向亚马逊简单队列服务 (Amazon SQS)、HTTP/HTTPS、电子邮件地址、() 或发送通知。 Amazon Server Migration Service Amazon SMS Amazon Lambda

订阅将附加到某个主题，该主题管理将消息发送给订阅者。要详细了解如何在 Amazon SNS 中使用 适用于 PHP 的 Amazon SDK 版本 3 创建主题。

以下示例演示如何：

- 使用 Subscribe 订阅到现有主题。
- 使用验证订阅ConfirmSubscription。
- 使用列出现有订阅ListSubscriptionsByTopic。
- 使用 Unsubscribe 删除订阅。
- 使用 publish 将消息发送给某一主题的所有订阅者。

有关使用亚马逊 SNS 的更多信息，请参阅使用亚马逊 SNS 发送消息。 System-to-System

的所有示例代码都可以在此 适用于 PHP 的 Amazon SDK 处找到 GitHub。

凭据

在运行示例代码之前，请配置您的 Amazon 证书，如中所述the section called “使用进行身份验证 Amazon”。然后导入 适用于 PHP 的 Amazon SDK，如中所述the section called “安装”。

针对主题订阅电子邮件地址

要建立电子邮件地址订阅，请使用 Subscribe 操作。

您可以使用订阅方法，根据在所传递参数中使用的值，将多种不同的端点订阅到某个 Amazon SNS 主题。本主题中的其他示例演示了这一点。

在本示例中，端点是电子邮件地址。将向该电子邮件发送确认令牌。在收到电子邮件的 3 天内，使用此确认令牌验证订阅。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

将应用程序端点订阅到主题

要建立 Web 应用程序订阅，请使用 Subscribe 操作。

您可以使用订阅方法，根据在所传递参数中使用的值，将多种不同的端点订阅到某个 Amazon SNS 主题。本主题中的其他示例演示了这一点。

在此示例中，端点是 URL。将会向此 Web 地址发送确认令牌。在收到电子邮件的 3 天内，使用此确认令牌验证订阅。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

向主题订阅 Lambda 函数

要建立 Lambda 函数订阅，请使用 Subscribe 操作。

您可以使用订阅方法，根据在所传递参数中使用的值，将多种不同的端点订阅到某个 Amazon SNS 主题。本主题中的其他示例演示了这一点。

在此示例中，端点是 Lambda 函数。将会向此 Lambda 函数发送确认令牌。在收到电子邮件的 3 天内，使用此确认令牌验证订阅。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

将文本 SMS 订阅到主题

要同时将 SMS 消息发送到多个电话号码，请将各个号码订阅到主题。

要建立电话号码订阅，请使用 Subscribe 操作。

您可以使用订阅方法，根据在所传递参数中使用的值，将多种不同的端点订阅到某个 Amazon SNS 主题。本主题中的其他示例演示了这一点。

在此示例中，端点是 E.164 格式的电话号码，这是国际电信的标准。

将会向此电话号码发送确认令牌。在收到电子邮件的 3 天内，使用此确认令牌验证订阅。

有关使用 Amazon SNS 发送 SMS 消息的替代方法，请参阅在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中发送 SMS 消息。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

确认订阅到主题

要实际创建订阅，端点所有者必须使用在最初建立订阅时发送的令牌，确认接收来自该主题的消息的意图，如前所述。确认令牌有效期为 3 天。3 天之后，您可以通过创建新订阅来重新发送令牌。

要确认订阅，请使用ConfirmSubscription操作。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

列出对主题的订阅

要列出给定 Amazon 区域中最多 100 个现有订阅，请使用ListSubscriptions操作。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

取消订阅主题

要删除订阅到某个主题的端点，请使用 Unsubscribe 操作。

如果订阅需要身份验证才能删除，则只有订阅的所有者或主题的所有者可以取消订阅，并且需要 Amazon 签名。如果取消订阅调用无需身份验证，并且请求者不是订阅所有者，则会将一条最终取消消息传送到端点。

导入

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

向 Amazon SNS 主题发布消息

要将消息发送到订阅到某个 Amazon SNS 主题的各个端点，请使用 Publish 操作。

创建包含用于发布消息的参数的对象，包括消息文本以及 Amazon SNS 主题的 Amazon 资源名称 (ARN)。

导入

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

```
use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SNS 中发送 SMS 消息

您可以使用 Amazon Simple Notification Service (Amazon SNS) 来将文本消息或 SMS 消息发送到支持 SMS 的设备上。您可以直接向电话号码发送消息，也可以使用多个电话号码订阅主题，然后通过向该主题发送消息来一次向这些电话号码发送消息。

使用 Amazon SNS 来指定发送 SMS 消息的首选项，例如如何优化消息传输（在成本或可靠传输方面）、您的每月支出限额、如何记录消息传输以及是否要订阅每日 SMS 使用率报告。这些首选项通过检索得到，并设置为 Amazon SNS 的 SMS 属性。

在发送 SMS 消息时，请使用 E.164 格式指定电话号码。E.164 是国际电信的标准电话号码结构。采用此格式的电话号码最多可包含 15 位数字，并以加号 (+) 和国家代码作为前缀。例如，E.164 格式的美国电话号码将显示为 +1001XXX5550100。

以下示例演示如何：

- 使用 GetSMSAttributes 检索从您账户发送 SMS 消息的默认设置。
- 使用 SetSMSAttributes 更新从您账户发送 SMS 消息的默认设置。
- 使用 CheckIfPhoneNumberISOptedOut 确定指定电话所有者是否已选择不接收来自您账户的 SMS 消息。
- 使用 ListPhoneNumberOptedOut 列出其所有者已选择不接收来自您账户的 SMS 消息的电话号码。
- 使用 Publish 直接将文本消息（SMS 消息）发送到电话号码。

有关使用 Amazon SNS 的更多信息，请参阅将手机号码作为订阅用户，将 Amazon SNS 用于用户通知（发送 SMS）。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

获取 SMS 属性

要检索 SMS 消息的默认设置，请使用 GetSMSAttributes 操作。

此示例获取 `DefaultSMSType` 属性。此属性控制 SMS 消息是作为 `Promotional` 发送（这将优化消息传送以尽可能降低成本）还是作为 `Transactional` 发送（这将优化消息传送以实现最高的可靠性）。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

设置 SMS 属性

要更新 SMS 消息的默认设置，请使用 SetSMSAttributes 操作。

此示例将 DefaultSMSType 属性设置为 Transactional，这会优化消息传送以实现最高的可靠性。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

检查电话号码是否已选择不接收消息

要确定指定电话所有者是否已选择不接收来自您账户的 SMS 消息，请使用 CheckIfPhoneNumberIsOptedOut 操作。

在此示例中，电话号码为 E.164 格式，这是国际电信的标准。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
```

```
}
```

列出选择不接收消息的电话号码

要检索其所有者已选择不接收来自您账户的 SMS 消息的电话号码列表，请使用 ListPhoneNumbersOptedOut 操作。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

发布到文本消息（SMS 消息）

要将文本消息（SMS 消息）直接传送到电话号码，请使用 Publish 操作。

在此示例中，电话号码为 E.164 格式，这是国际电信的标准。

SMS 消息最多可以包含 140 个字节。单个 SMS 发布操作的大小限制为 1600 字节。

有关发送 SMS 消息的更多详细信息，请参阅发送 SMS 消息。

导入。

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

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 示例

Amazon Simple Queue Service (SQS) 是一种快速、可靠、可扩展且完全托管的消息队列服务。Amazon SQS 使您能够分离云应用程序的组件。Amazon SQS 包括具有高吞吐量和“至少一次”处理的标准队列，以及提供 FIFO（先进先出）交付和“仅一次”处理的 FIFO 队列。

GitHub 上提供了 适用于 PHP 的 Amazon SDK 版本 3 的所有示例代码。

主题

- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中启用长轮询

- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中管理可见性超时
- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中发送和接收消息
- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中使用死信队列
- 在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中使用队列

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中启用长轮询

在发送响应之前，长轮询使 Amazon SQS 等待指定的时间，以便消息在队列中变得可用，从而减少空响应的数量。此外，长轮询通过查询所有服务器而不是执行服务器采样，消除了假的空响应。要启用长轮询，请为接收的消息指定不为零的等待时间。要了解更多信息，请参阅 SQS 长轮询。

以下示例演示如何：

- 使用 SetQueueAttributes 设置 Amazon SQS 队列的属性，以启用长轮询。
- 使用 ReceiveMessage 借助长轮询检索一条或多条消息。
- 使用 CreateQueue 创建长轮询队列。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

设置队列属性以启用长轮询

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

使用长轮询检索消息

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

使用长轮询创建队列

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中管理可见性超时

可见性超时是 Amazon SQS 阻止其他使用组件接收并处理消息的一段时间。要了解更多信息，请参阅可见性超时。

以下示例演示如何：

- 使用 ChangeMessageVisibilityBatch 将队列中指定消息的可见性超时更改为新值。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

更改多条消息的可见性超时

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```

$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());
}
```

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中发送和接收消息

要了解 Amazon SQS 消息，请参阅 Service Quotas 用户指南中的向 SQS 队列发送消息和从 SQS 队列接收和删除消息。

以下示例演示如何：

- 使用 SendMessage 向指定的队列传输消息。
- 使用 ReceiveMessage 从指定的队列检索一条或多条消息（最多 10 条）。

- 使用 DeleteMessage 从队列中删除消息。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

发送消息

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

接收和删除消息

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中使用死信队列

其他（源）队列可将无法成功处理的消息转到死信队列。您可以在死信队列中留出和隔离这些消息以确定其处理失败的原因。您必须单独配置将消息发送到死信队列的每个源队列。多个队列可将一个死信队列作为目标。

要了解更多信息，请参阅使用 SQS 死信队列。

以下示例演示如何：

- 使用 SetQueueAttributes 启用死信队列。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

启用死信队列

导入。

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

use Aws\Exception\AwsException;
```

```
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

在使用 适用于 PHP 的 Amazon SDK 版本 3 的 Amazon SQS 中使用队列

要了解有关 Amazon SQS 队列的更多信息，请参阅 SQS 队列的工作原理。

以下示例演示如何：

- 使用 ListQueues 返回队列的列表。
- 使用 CreateQueue 创建新队列。
- 使用 GetQueueUrl 返回现有队列的 URL。
- 使用 DeleteQueue 删除指定的队列。

适用于 PHP 的 Amazon SDKGitHub 上提供了的所有示例代码。

凭证

运行示例代码之前，请配置您的 Amazon 凭证，如 the section called “使用进行身份验证 Amazon” 中所述。然后导入 适用于 PHP 的 Amazon SDK，如 the section called “安装” 中所述。

返回队列列表

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

创建队列

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

返回队列的 URL

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

删除队列

导入。

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

use Aws\Exception\AwsException;
use Aws\Sqs\SqsClient;
```

示例代码

```
$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());
}
```

使用 适用于 PHP 的 Amazon SDK 版本 3 向 Amazon EventBridge 全球终端节点发送事件

您可以使用 Amazon EventBridge 全球终端节点来提高事件驱动型应用程序的可用性和可靠性。

设置 EventBridge 全局端点后，您可以使用适用于 PHP 的 SDK 向其发送事件。

> **Important**
>
> 要在适用于 PHP 的 SDK 中使用 EventBridge 全局端点，您的 PHP 环境必须安装Amazon 通用运行时 (Amazon CRT) 扩展。

以下示例使用的PutEvents方法将单个事件发送`EventBridgeClient`到 EventBridge 全局终端节点。

```
<?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]
]);
```

这篇博文包含有关 EventBridge 全局端点的更多信息。

适用于 PHP 的 SDK 代码示例

本主题中的代码示例向您展示了如何使用 wit 适用于 PHP 的 Amazon SDK h Amazon。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务服务结合来完成特定任务的代码示例。

某些服务包含其他示例类别，这些类别说明如何利用特定于服务的库或函数。

Services

- 使用适用于 PHP 的 SDK 的 API Gateway 示例
- 使用适用于 PHP 的 SDK 的 Aurora 示例
- 使用适用于 PHP 的 SDK 的 Auto Scaling 示例
- 使用 SDK for PHP 的 Amazon Bedrock 示例
- 使用 SDK for PHP 的 Amazon Bedrock 运行时系统示例
- 使用适用于 PHP 的 SDK 的 Amazon DocumentDB 示例
- 使用适用于 PHP 的 SDK 的 DynamoDB 示例
- 使用适用于 PHP 的 SDK 的亚马逊 EC2 示例
- Amazon Glue 使用适用于 PHP 的 SDK 的示例
- 使用适用于 PHP 的 SDK 的 IAM 示例
- 使用适用于 PHP 的 SDK 的 Kinesis 示例
- Amazon KMS 使用适用于 PHP 的 SDK 的示例
- 使用适用于 PHP 的 SDK 的 Lambda 示例
- 使用适用于 PHP 的 SDK 的 Amazon MSK 示例
- 使用适用于 PHP 的 SDK 的 Amazon RDS 示例
- 使用适用于 PHP 的 SDK 的 Amazon RDS 数据服务示例
- 使用适用于 PHP 的 SDK 的 Amazon Rekognition 示例
- 使用适用于 PHP 的 SDK 的 Amazon S3 示例
- 使用适用于 PHP 的 SDK 的 S3 目录存储桶示例

- 使用适用于 PHP 的 SDK 的 Amazon SES 示例
- 使用适用于 PHP 的 SDK 的 Amazon SNS 示例
- 使用适用于 PHP 的 SDK 的 Amazon SQS 示例

使用适用于 PHP 的 SDK 的 API Gateway 示例

以下代码示例向您展示了如何使用 with API Gateway 来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景

操作

`GetBasePathMapping`

以下代码示例演示了如何使用 `GetBasePathMapping`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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();
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[GetBasePathMapping]中的。

`ListBasePathMappings`

以下代码示例演示了如何使用 `ListBasePathMappings`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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();
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListBasePathMappings中的。

UpdateBasePathMapping

以下代码示例演示了如何使用 UpdateBasePathMapping。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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();
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[UpdateBasePathMapping](#)中的。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

使用适用于 PHP 的 SDK 的 Aurora 示例

以下代码示例向您展示了如何通过 适用于 PHP 的 Amazon SDK 与 Aurora 一起使用来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

场景

创建 Aurora Serverless 工作项跟踪器

以下代码示例演示如何创建 Web 应用程序，来跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service（Amazon SES）发送报告。

适用于 PHP 的 SDK

演示如何使用创建一个 Web 应用程序，该 适用于 PHP 的 Amazon SDK 应用程序通过亚马逊简单电子邮件服务 (Amazon SES) Semple Service 跟踪亚马逊 RDS 数据库中的工作项目并通过电子邮件发送报告。此示例使用使用 React.js 构建的前端与 RESTful PHP 后端进行交互。

- 将 React.js 网络应用程序与 Amazon 服务集成。
- 列出、添加、更新和删除 Amazon RDS 表中的项目。
- 使用 Amazon SES 以电子邮件发送已筛选工作项的报告。
- 使用随附的 Amazon CloudFormation 脚本部署和管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

使用适用于 PHP 的 SDK 的 Auto Scaling 示例

以下代码示例向您展示了如何使用与 Auto Scaling 适用于 PHP 的 Amazon SDK 配合使用来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Auto Scaling

以下代码示例显示了如何开始使用 Auto Scaling。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function helloService()
{
    $autoScalingClient = new AutoScalingClient([
        'region' => 'us-west-2',
        'version' => 'latest',
        'profile' => 'default',
    ]);

    $groups = $autoScalingClient->describeAutoScalingGroups([]);
    var_dump($groups);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeAutoScalingGroups中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 使用启动模板和可用区创建一个 Amazon A EC2 uto Scaling 群组，并获取有关正在运行的实例的信息。
- 启用 Amazon CloudWatch 指标收集。
- 更新组的所需容量，并等待实例启动。

- 终止组中的实例。
- 列出为响应用户请求和容量变化而发生的扩缩活动。
- 获取 CloudWatch 指标的统计数据，然后清理资源。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

操作

`CreateAutoScalingGroup`

以下代码示例演示了如何使用 `CreateAutoScalingGroup`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function createAutoScalingGroup(
        $autoScalingGroupName,
        $availabilityZones,
        $minSize,
        $maxSize,
        $launchTemplateId
    ) {
        return $this->autoScalingClient->createAutoScalingGroup([
            'AutoScalingGroupName' => $autoScalingGroupName,
```

```
            'AvailabilityZones' => $availabilityZones,
            'MinSize' => $minSize,
            'MaxSize' => $maxSize,
            'LaunchTemplate' => [
                'LaunchTemplateId' => $launchTemplateId,
            ],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateAutoScalingGroup中的。

`DeleteAutoScalingGroup`

以下代码示例演示了如何使用 `DeleteAutoScalingGroup`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function deleteAutoScalingGroup($autoScalingGroupName)
    {
        return $this->autoScalingClient->deleteAutoScalingGroup([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'ForceDelete' => true,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteAutoScalingGroup中的。

`DescribeAutoScalingGroups`

以下代码示例演示了如何使用 `DescribeAutoScalingGroups`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function describeAutoScalingGroups($autoScalingGroupNames)
{
    return $this->autoScalingClient->describeAutoScalingGroups([
        'AutoScalingGroupNames' => $autoScalingGroupNames
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeAutoScalingGroups中的。

`DescribeAutoScalingInstances`

以下代码示例演示了如何使用 `DescribeAutoScalingInstances`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function describeAutoScalingInstances($instanceIds)
{
    return $this->autoScalingClient->describeAutoScalingInstances([
        'InstanceIds' => $instanceIds
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeAutoScalingInstances中的。

`DescribeScalingActivities`

以下代码示例演示了如何使用 `DescribeScalingActivities`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function describeScalingActivities($autoScalingGroupName)
{
    return $this->autoScalingClient->describeScalingActivities([
        'AutoScalingGroupName' => $autoScalingGroupName,
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeScalingActivities中的。

`DisableMetricsCollection`

以下代码示例演示了如何使用 `DisableMetricsCollection`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function disableMetricsCollection($autoScalingGroupName)
```

```
    {
        return $this->autoScalingClient->disableMetricsCollection([
            'AutoScalingGroupName' => $autoScalingGroupName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DisableMetricsCollection中的。

`EnableMetricsCollection`

以下代码示例演示了如何使用 `EnableMetricsCollection`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function enableMetricsCollection($autoScalingGroupName, $granularity)
    {
        return $this->autoScalingClient->enableMetricsCollection([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'Granularity' => $granularity,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考EnableMetricsCollection中的。

`SetDesiredCapacity`

以下代码示例演示了如何使用 `SetDesiredCapacity`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function setDesiredCapacity($autoScalingGroupName, $desiredCapacity)
{
    return $this->autoScalingClient->setDesiredCapacity([
        'AutoScalingGroupName' => $autoScalingGroupName,
        'DesiredCapacity' => $desiredCapacity,
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考SetDesiredCapacity中的。

`TerminateInstanceInAutoScalingGroup`

以下代码示例演示了如何使用 `TerminateInstanceInAutoScalingGroup`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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;
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[TerminateInstanceInAutoScalingGroup](#)中的。

`UpdateAutoScalingGroup`

以下代码示例演示了如何使用 `UpdateAutoScalingGroup`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 [Amazon 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

```
    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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考UpdateAutoScalingGroup中的。

使用 SDK for PHP 的 Amazon Bedrock 示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon Bedrock 配合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

`ListFoundationModels`

以下代码示例演示了如何使用 `ListFoundationModels`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出可用的 Amazon Bedrock 基础模型。

```
public function listFoundationModels()
{
    $bedrockClient = new BedrockClient([
        'region' => 'us-west-2',
        'profile' => 'default'
    ]);
    $response = $bedrockClient->listFoundationModels();
    return $response['modelSummaries'];
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListFoundationModels中的。

使用 SDK for PHP 的 Amazon Bedrock 运行时系统示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon Bedrock Runtime 配合使用来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 场景
- Amazon Nova
- Amazon Titan 图像生成器

- Anthropic Claude
- Stable Diffusion

场景

在 Amazon Bedrock 上调用多个基础模型

以下代码示例展示了如何在 Amazon Bedrock 上准备和向各种大型语言模型 (LLMs) 发送提示

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

LLMs 在 Amazon Bedrock 上调用多个。

```
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";
        $bedrockRuntimeService = new BedrockRuntimeService();
        $prompt = 'In one paragraph, who are you?';
        echo "\nPrompt: " . $prompt;
        echo "\n\nAnthropic Claude:\n";
        echo $bedrockRuntimeService->invokeClaude($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 image has 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-v2');
        echo "The generated image has 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;
    }
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - InvokeModel
  - InvokeModelWithResponseStream

Amazon Nova

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Amazon Nova 发送文本消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 Bedrock 的 Converse API 向 Amazon Nova 发送文本消息。

```
// Use the Conversation API to send a text message to Amazon Nova.

use Aws\BedrockRuntime\BedrockRuntimeClient;
use Aws\Exception\AwsException;
use RuntimeException;

class Converse
{
    public function converse(): string
    {
        // Create a Bedrock Runtime client in the AWS Region you want to use.
        $client = new BedrockRuntimeClient([
            'region' => 'us-east-1',
            'profile' => 'default'
        ]);

        // Set the model ID, e.g., Amazon Nova Lite.
        $modelId = 'amazon.nova-lite-v1:0';

        // Start a conversation with the user message.
        $userMessage = "Describe the purpose of a 'hello world' program in one
 line.";
        $conversation = [
            [
                "role" => "user",
                "content" => [["text" => $userMessage]]
```

```
            ]
        ];

        try {
            // Send the message to the model, using a basic inference configuration.
            $response = $client->converse([
                'modelId' => $modelId,
                'messages' => $conversation,
                'inferenceConfig' => [
                    'maxTokens' => 512,
                    'temperature' => 0.5
                ]
            ]);

            // Extract and return the response text.
            $responseText = $response['output']['message']['content'][0]['text'];
            return $responseText;
        } catch (AwsException $e) {
            echo "ERROR: Can't invoke {$modelId}. Reason: {$e-
>getAwsErrorMessage()}";
            throw new RuntimeException("Failed to invoke model: " . $e-
>getAwsErrorMessage(), 0, $e);
        }
    }
}

$demo = new Converse();
echo $demo->converse();
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Converse。

Amazon Titan 图像生成器

InvokeModel

以下代码示例演示如何在 Amazon Bedrock 上调用 Amazon Titan Image 来生成图像。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 Amazon Titan 图像生成器创建图像。

```
    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-v2:0';
            $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;
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考InvokeModel中的。

Anthropic Claude

InvokeModel

以下代码示例演示如何使用调用模型 API 向 Anthropic Claude 发送文本消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

```
    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-3-haiku-20240307-v1:0';
        // Claude requires you to enclose the prompt as follows:
            $body = [
                'anthropic_version' => 'bedrock-2023-05-31',
                'max_tokens' => 512,
```

```
                'temperature' => 0.5,
                'messages' => [[
                    'role' => 'user',
                    'content' => $prompt
                ]]
            ];
            $result = $this->bedrockRuntimeClient->invokeModel([
                'contentType' => 'application/json',
                'body' => json_encode($body),
                'modelId' => $modelId,
            ]);
            $response_body = json_decode($result['body']);
            $completion = $response_body->content[0]->text;
        } catch (Exception $e) {
            echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
        }

        return $completion;
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考InvokeModel中的。

Stable Diffusion

InvokeModel

以下代码示例演示如何在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 来生成图像。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 Stable Diffusion 创建图像。

```
    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-v1';
            $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;
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考InvokeModel中的。

使用适用于 PHP 的 SDK 的 Amazon DocumentDB 示例

以下代码示例向您展示了如何在 Amazon DocumentDB 中 适用于 PHP 的 Amazon SDK 使用来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

无服务器示例

通过 Amazon DocumentDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DocumentDB 更改流的记录而触发的事件。该函数检索 DocumentDB 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 Amazon DocumentDB 事件与 Lambda 结合使用。

```
<?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();
```

使用适用于 PHP 的 SDK 的 DynamoDB 示例

以下代码示例向您展示了如何在 DynamoDB 中使用来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建可保存电影数据的表。
- 在表中加入单一电影，获取并更新此电影。
- 向 JSON 示例文件的表中写入电影数据。
- 查询在给定年份发行的电影。
- 扫描在年份范围内发行的电影。
- 删除表中的电影后再删除表。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

由于此示例使用支持文件，因此请务必阅读指南（位于 PHP 示例 README.md 文件中）。

```
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);
    }
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

操作

`BatchExecuteStatement`

以下代码示例演示了如何使用 `BatchExecuteStatement`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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,
                ],
            ],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考BatchExecuteStatement中的。

`BatchWriteItem`

以下代码示例演示了如何使用 `BatchWriteItem`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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;
           }
       }
   }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考BatchWriteItem中的。

CreateTable

以下代码示例演示了如何使用 CreateTable。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

创建表。

```
        $tableName = "ddb_demo_table_$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],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateTable中的。

`DeleteItem`

以下代码示例演示了如何使用 `DeleteItem`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteItem中的。

DeleteTable

以下代码示例演示了如何使用 DeleteTable。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function deleteTable(string $TableName)
{
    $this->customWaiter(function () use ($TableName) {
        return $this->dynamoDbClient->deleteTable([
            'TableName' => $TableName,
        ]);
    });
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteTable中的。

ExecuteStatement

以下代码示例演示了如何使用 ExecuteStatement。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ExecuteStatement中的。

GetItem

以下代码示例演示了如何使用 GetItem。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetItem中的。

`ListTables`

以下代码示例演示了如何使用 `ListTables`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
```

```
            'Limit' => $limit,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[ListTables](#)中的。

`PutItem`

以下代码示例演示了如何使用 `PutItem`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考PutItem中的。

Query

以下代码示例演示了如何使用 Query。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$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);
    }
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Query。

Scan

以下代码示例演示了如何使用 Scan。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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);
    }
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Scan。

`UpdateItem`

以下代码示例演示了如何使用 `UpdateItem`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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
                ]
            ],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考UpdateItem中的。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

使用批量 PartiQL 语句查询表

以下代码示例展示了如何：

- 通过运行多个 SELECT 语句来获取一批项目。
- 通过运行多个 INSERT 语句来添加一批项目。
- 通过运行多个 UPDATE 语句来更新一批项目。
- 通过运行多个 DELETE 语句来删除一批项目。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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,
                ],
            ],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考BatchExecuteStatement中的。

使用 PartiQL 来查询表

以下代码示例展示了如何：

- 通过运行 SELECT 语句来获取项目。
- 通过运行 INSERT 语句来添加项目。
- 通过运行 UPDATE 语句来更新项目。
- 通过运行 DELETE 语句来删除项目。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ExecuteStatement中的。

无服务器示例

通过 DynamoDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DynamoDB 流的记录而触发的事件。该函数检索 DynamoDB 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 DynamoDB 事件与 Lambda 结合使用。

```
<?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);
```

通过 DynamoDB 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为接收来自 DynamoDB 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 PHP 通过 Lambda 进行 DynamoDB 批处理项目失败。

```
<?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);
```

使用适用于 PHP 的 SDK 的亚马逊 EC2 示例

以下代码示例向您展示了如何通过 适用于 PHP 的 Amazon SDK 与 Amazon 一起使用来执行操作和实现常见场景 EC2。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

操作

`CreateVpc`

以下代码示例演示了如何使用 `CreateVpc`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * @param string $cidr
 * @return array
 */
public function createVpc(string $cidr): array
{
    try {
        $result = $this->ec2Client->createVpc([
            "CidrBlock" => $cidr,
        ]);
        return $result['Vpc'];
```

```
        }catch(Ec2Exception $caught){
            echo "There was a problem creating the VPC: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateVpc中的。

`CreateVpcEndpoint`

以下代码示例演示了如何使用 `CreateVpcEndpoint`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /**
    * @param string $serviceName
    * @param string $vpcId
    * @param array $routeTableIds
    * @return array
    */
   public function createVpcEndpoint(string $serviceName, string $vpcId, array
$routeTableIds): array
   {
       try {
           $result = $this->ec2Client->createVpcEndpoint([
               'ServiceName' => $serviceName,
               'VpcId' => $vpcId,
               'RouteTableIds' => $routeTableIds,
           ]);

           return $result["VpcEndpoint"];
       } catch(Ec2Exception $caught){
```

```
            echo "There was a problem creating the VPC Endpoint: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateVpcEndpoint中的。

`DeleteVpc`

以下代码示例演示了如何使用 `DeleteVpc`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * @param string $vpcId
     * @return void
     */
    public function deleteVpc(string $vpcId)
    {
        try {
            $this->ec2Client->deleteVpc([
                "VpcId" => $vpcId,
            ]);
        }catch(Ec2Exception $caught){
            echo "There was a problem deleting the VPC: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteVpc中的。

`DeleteVpcEndpoints`

以下代码示例演示了如何使用 `DeleteVpcEndpoints`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * @param string $vpcEndpointId
     * @return void
     */
    public function deleteVpcEndpoint(string $vpcEndpointId)
    {
        try {
            $this->ec2Client->deleteVpcEndpoints([
                "VpcEndpointIds" => [$vpcEndpointId],
            ]);
        }catch (Ec2Exception $caught){
            echo "There was a problem deleting the VPC Endpoint: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteVpcEndpoints中的。

`DescribeRouteTables`

以下代码示例演示了如何使用 `DescribeRouteTables`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /**
    * @param array $routeTableIds
    * @param array $filters
    * @return array
    */
   public function describeRouteTables(array $routeTableIds = [], array $filters =
[]): array
   {
       $parameters = [];
       if($routeTableIds){
           $parameters['RouteTableIds'] = $routeTableIds;
       }
       if($filters){
           $parameters['Filters'] = $filters;
       }
       try {
           $paginator = $this->ec2Client->getPaginator("DescribeRouteTables",
$parameters);
           $contents = [];
           foreach ($paginator as $result) {
               foreach ($result['RouteTables'] as $object) {
                   $contents[] = $object['RouteTableId'];
               }
           }
       }catch (Ec2Exception $caught){
           echo "There was a problem paginating the results of DescribeRouteTables:
{$caught->getAwsErrorMessage()}\n";
           throw $caught;
       }
       return $contents;
   }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeRouteTables中的。

Amazon Glue 使用适用于 PHP 的 SDK 的示例

以下代码示例向您展示了如何使用with来执行操作和实现常见场景 Amazon Glue。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 基本功能
- 操作

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建爬网程序，爬取公有 Amazon S3 存储桶并生成包含 CSV 格式的元数据的数据库。
- 列出有关中数据库和表的信息 Amazon Glue Data Catalog。
- 创建任务，从 S3 存储桶提取 CSV 数据，转换数据，然后将 JSON 格式的输出加载到另一个 S3 存储桶中。
- 列出有关作业运行的信息，查看转换后的数据，并清除资源。

有关更多信息，请参阅教程： Amazon Glue Studio 入门。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
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,
        ]);
    }
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

操作

`CreateCrawler`

以下代码示例演示了如何使用 `CreateCrawler`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
                        ]]
                ],
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateCrawler中的。

CreateJob

以下代码示例演示了如何使用 CreateJob。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateJob中的。

DeleteCrawler

以下代码示例演示了如何使用 DeleteCrawler。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteCrawler中的。

`DeleteDatabase`

以下代码示例演示了如何使用 `DeleteDatabase`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
```

```
    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteDatabase中的。

DeleteJob

以下代码示例演示了如何使用 DeleteJob。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteJob中的。

DeleteTable

以下代码示例演示了如何使用 DeleteTable。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteTable中的。

`GetCrawler`

以下代码示例演示了如何使用 `GetCrawler`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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,
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetCrawler中的。

`GetDatabase`

以下代码示例演示了如何使用 `GetDatabase`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $databaseName = "doc-example-database-$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,
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetDatabase中的。

`GetJobRun`

以下代码示例演示了如何使用 `GetJobRun`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetJobRun中的。

GetJobRuns

以下代码示例演示了如何使用 GetJobRuns。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetJobRuns中的。

GetTables

以下代码示例演示了如何使用 GetTables。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
            'DatabaseName' => $databaseName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetTables中的。

`ListJobs`

以下代码示例演示了如何使用 `ListJobs`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListJobs中的。

`StartCrawler`

以下代码示例演示了如何使用 `StartCrawler`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $glueService->startCrawler($crawlerName);

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
```

```
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考StartCrawler中的。

`StartJobRun`

以下代码示例演示了如何使用 `StartJobRun`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
           ],
       ]);
   }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考StartJobRun中的。

使用适用于 PHP 的 SDK 的 IAM 示例

以下代码示例向您展示了如何使用 with IAM 来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

基本功能

了解基本功能

以下代码示例展示了如何创建用户并代入角色。

> **Warning**
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供者的联合身份验证，例如 Amazon IAM Identity Center。

- 创建没有权限的用户。
- 创建授予列出账户的 Amazon S3 存储桶的权限的角色
- 添加策略以允许用户代入该角色。
- 代入角色并使用临时凭证列出 S3 存储桶，然后清除资源。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
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";
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

操作

`AttachRolePolicy`

以下代码示例演示了如何使用 `AttachRolePolicy`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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,
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考AttachRolePolicy中的。

`CreatePolicy`

以下代码示例演示了如何使用 `CreatePolicy`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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";

    /**
     * @param string $policyName
```

```
     * @param string $policyDocument
     * @return array
     */
    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'];
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreatePolicy中的。

`CreateRole`

以下代码示例演示了如何使用 `CreateRole`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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'];
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateRole中的。

`CreateServiceLinkedRole`

以下代码示例演示了如何使用 `CreateServiceLinkedRole`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateServiceLinkedRole中的。

`CreateUser`

以下代码示例演示了如何使用 `CreateUser`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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'];
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateUser中的。

GetAccountPasswordPolicy

以下代码示例演示了如何使用 GetAccountPasswordPolicy。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$uuid = uniqid();
$service = new IAMService();

    public function getAccountPasswordPolicy()
    {
        return $this->iamClient->getAccountPasswordPolicy();
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetAccountPasswordPolicy中的。

GetPolicy

以下代码示例演示了如何使用 GetPolicy。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$uuid = uniqid();
$service = new IAMService();

    public function getPolicy($policyArn)
    {
        return $this->customWaiter(function () use ($policyArn) {
            return $this->iamClient->getPolicy(['PolicyArn' => $policyArn]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetPolicy中的。

`GetRole`

以下代码示例演示了如何使用 `GetRole`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$uuid = uniqid();
$service = new IAMService();

    public function getRole($roleName)
    {
        return $this->customWaiter(function () use ($roleName) {
            return $this->iamClient->getRole(['RoleName' => $roleName]);
```

```
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetRole中的。

`ListAttachedRolePolicies`

以下代码示例演示了如何使用 `ListAttachedRolePolicies`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListAttachedRolePolicies中的。

ListGroups

以下代码示例演示了如何使用 ListGroups。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListGroups中的。

ListPolicies

以下代码示例演示了如何使用 ListPolicies。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListPolicies中的。

`ListRolePolicies`

以下代码示例演示了如何使用 `ListRolePolicies`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[ListRolePolicies](#)中的。

`ListRoles`

以下代码示例演示了如何使用 `ListRoles`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListRoles中的。

`ListSAMLProviders`

以下代码示例演示了如何使用 `ListSAMLProviders`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$uuid = uniqid();
$service = new IAMService();

    public function listSAMLProviders()
    {
        return $this->iamClient->listSAMLProviders();
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考SAMLProviders中的列表。

`ListUsers`

以下代码示例演示了如何使用 `ListUsers`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListUsers中的。

使用适用于 PHP 的 SDK 的 Kinesis 示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Kinesis 配合使用来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

无服务器示例

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 Kinesis 事件与 Lambda 结合使用。

```
// 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);
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 Kinesis 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告通过 PHP 进行 Lambda Kinesis 批处理项目失败。

```
// 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);
```

Amazon KMS 使用适用于 PHP 的 SDK 的示例

以下代码示例向您展示了如何使用with来执行操作和实现常见场景 Amazon KMS。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

你好 Amazon KMS

以下代码示例展示了如何开始使用 Amazon Key Management Service。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
include "vendor/autoload.php";

use Aws\Kms\KmsClient;

echo "This file shows how to connect to the KmsClient, uses a paginator to get the
 keys for the account, and lists the KeyIds for up to 10 keys.\n";

$client = new KmsClient([]);

$pageLength = 10; // Change this value to change the number of records shown, or to
 break up the result into pages.

$keys = [];
$keysPaginator = $client->getPaginator("ListKeys", ['Limit' => $pageLength]);
foreach($keysPaginator as $page){
    foreach($page['Keys'] as $index => $key){
        echo "The $index index Key's ID is: {$key['KeyId']}\n";
    }
    echo "End of page one of results. Alter the \$pageLength variable to see more
 results.\n";
    break;
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListKeys中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建 KMS 密钥。
- 列出您账户的 KMS 密钥并获取有关它们的详细信息。
- 启用和禁用 KMS 密钥。
- 生成可用于客户端加密的对称数据密钥。
- 生成用于对数据进行数字签名的非对称密钥。
- 标签键。
- 删除 KMS 密钥。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
        echo "\n";
        echo "-------------------------------------\n";
        echo <<<WELCOME
Welcome to the AWS Key Management Service SDK Basics scenario.

This program demonstrates how to interact with AWS Key Management Service using the
 AWS SDK for PHP (v3).
The AWS Key Management Service (KMS) is a secure and highly available service that
 allows you to create
and manage AWS KMS keys and control their use across a wide range of AWS services
 and applications.
KMS provides a centralized and unified approach to managing encryption keys, making
 it easier to meet your
data protection and regulatory compliance requirements.

This KMS Basics scenario creates two key types:
- A symmetric encryption key is used to encrypt and decrypt data.
- An asymmetric key used to digitally sign data.

Let's get started...\n
WELCOME;
        echo "-------------------------------------\n";
```

```
        $this->pressEnter();

        $this->kmsClient = new KmsClient([]);
        // Initialize the KmsService class with the client. This allows you to
 override any defaults in the client before giving it to the service class.
        $this->kmsService = new KmsService($this->kmsClient);

        // 1. Create a symmetric KMS key.
        echo "\n";
        echo "1. Create a symmetric KMS key.\n";
        echo "First, we will create a symmetric KMS key that is used to encrypt and
 decrypt data by invoking createKey().\n";
        $this->pressEnter();

        $key = $this->kmsService->createKey();
        $this->resources['symmetricKey'] = $key['KeyId'];
        echo "Created a customer key with ARN {$key['Arn']}.\n";
        $this->pressEnter();

        // 2. Enable a KMS key.
        echo "\n";
        echo "2. Enable a KMS key.\n";
        echo "By default when you create an AWS key, it is enabled. The code checks
 to
determine if the key is enabled. If it is not enabled, the code enables it.\n";
        $this->pressEnter();

        $keyInfo = $this->kmsService->describeKey($key['KeyId']);
        if(!$keyInfo['Enabled']){
            echo "The key was not enabled, so we will enable it.\n";
            $this->pressEnter();
            $this->kmsService->enableKey($key['KeyId']);
            echo "The key was successfully enabled.\n";
        }else{
            echo "The key was already enabled, so there was no need to enable it.
\n";
        }
        $this->pressEnter();

        // 3. Encrypt data using the symmetric KMS key.
        echo "\n";
        echo "3. Encrypt data using the symmetric KMS key.\n";
        echo "One of the main uses of symmetric keys is to encrypt and decrypt data.
\n";
```

```
        echo "Next, we'll encrypt the string 'Hello, AWS KMS!' with the
 SYMMETRIC_DEFAULT encryption algorithm.\n";
        $this->pressEnter();
        $text = "Hello, AWS KMS!";
        $encryption = $this->kmsService->encrypt($key['KeyId'], $text);
        echo "The plaintext data was successfully encrypted with the algorithm:
 {$encryption['EncryptionAlgorithm']}.\n";
        $this->pressEnter();

        // 4. Create an alias.
        echo "\n";
        echo "4. Create an alias.\n";
        $aliasInput = testable_readline("Please enter an alias prefixed with
 \"alias/\" or press enter to use a default value: ");
        if($aliasInput == ""){
            $aliasInput = "alias/dev-encryption-key";
        }
        $this->kmsService->createAlias($key['KeyId'], $aliasInput);
        $this->resources['alias'] = $aliasInput;
        echo "The alias \"$aliasInput\" was successfully created.\n";
        $this->pressEnter();

        // 5. List all of your aliases.
        $aliasPageSize = 10;
        echo "\n";
        echo "5. List all of your aliases, up to $aliasPageSize.\n";
        $this->pressEnter();
        $aliasPaginator = $this->kmsService->listAliases();
        foreach($aliasPaginator as $pages){
            foreach($pages['Aliases'] as $alias){
                echo $alias['AliasName'] . "\n";
            }
            break;
        }
        $this->pressEnter();

        // 6. Enable automatic rotation of the KMS key.
        echo "\n";
        echo "6. Enable automatic rotation of the KMS key.\n";
        echo "By default, when the SDK enables automatic rotation of a KMS key,
KMS rotates the key material of the KMS key one year (approximately 365 days) from
 the enable date and every year
thereafter.";
        $this->pressEnter();
```

```
        $this->kmsService->enableKeyRotation($key['KeyId']);
        echo "The key's rotation was successfully set for key: {$key['KeyId']}\n";
        $this->pressEnter();

        // 7. Create a grant.
        echo "7. Create a grant.\n";
        echo "\n";
        echo "A grant is a policy instrument that allows Amazon Web Services
 principals to use KMS keys.
It also can allow them to view a KMS key (DescribeKey) and create and manage grants.
When authorizing access to a KMS key, grants are considered along with key policies
 and IAM policies.\n";
        $granteeARN = testable_readline("Please enter the Amazon Resource Name
 (ARN) of an Amazon Web Services principal. Valid principals include Amazon Web
 Services accounts, IAM users, IAM roles, federated users, and assumed role users.
 For help with the ARN syntax for a principal, see IAM ARNs in the Identity and
 Access Management User Guide. \nTo skip this step, press enter without any other
 values: ");
        if($granteeARN){
            $operations = [
                "ENCRYPT",
                "DECRYPT",
                "DESCRIBE_KEY",
            ];
            $grant = $this->kmsService->createGrant($key['KeyId'], $granteeARN,
 $operations);
            echo "The grant Id is: {$grant['GrantId']}\n";
        }else{
            echo "Steps 7, 8, and 9 will be skipped.\n";
        }
        $this->pressEnter();

        // 8. List grants for the KMS key.
        if($granteeARN){
            echo "8. List grants for the KMS key.\n\n";
            $grantsPaginator = $this->kmsService->listGrants($key['KeyId']);
            foreach($grantsPaginator as $page){
                foreach($page['Grants'] as $grant){
                    echo $grant['GrantId'] . "\n";
                }
            }
        }else{
            echo "Skipping step 8...\n";
        }
```

```
        $this->pressEnter();

        // 9. Revoke the grant.
        if($granteeARN) {
            echo "\n";
            echo "9. Revoke the grant.\n";
            $this->pressEnter();
            $this->kmsService->revokeGrant($grant['GrantId'], $keyInfo['KeyId']);
            echo "{$grant['GrantId']} was successfully revoked!\n";
        }else{
            echo "Skipping step 9...\n";
        }
        $this->pressEnter();

        // 10. Decrypt the data.
        echo "\n";
        echo "10. Decrypt the data.\n";
        echo "Let's decrypt the data that was encrypted before.\n";
        echo "We'll use the same key to decrypt the string that we encrypted earlier
 in the program.\n";
        $this->pressEnter();
        $decryption = $this->kmsService->decrypt($keyInfo['KeyId'],
 $encryption['CiphertextBlob'], $encryption['EncryptionAlgorithm']);
        echo "The decrypted text is: {$decryption['Plaintext']}\n";
        $this->pressEnter();

        // 11. Replace a Key Policy.
        echo "\n";
        echo "11. Replace a Key Policy.\n";
        echo "A key policy is a resource policy for a KMS key. Key policies are the
 primary way to control access to KMS keys.\n";
        echo "Every KMS key must have exactly one key policy. The statements in the
 key policy determine who has permission to use the KMS key and how they can use it.
\n";
        echo " You can also use IAM policies and grants to control access to the KMS
 key, but every KMS key must have a key policy.\n";
        echo "We will replace the key's policy with a new one:\n";
        $stsClient = new StsClient([]);
        $result = $stsClient->getCallerIdentity();
        $accountId = $result['Account'];
        $keyPolicy = <<< KEYPOLICY
{
    "Version":"2012-10-17",
    "Statement": [{
```

```
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::$accountId:root"},
        "Action": "kms:*",
        "Resource": "*"
    }]
}
KEYPOLICY;
        echo $keyPolicy;
        $this->pressEnter();
        $this->kmsService->putKeyPolicy($keyInfo['KeyId'], $keyPolicy);
        echo "The Key Policy was successfully replaced!\n";
        $this->pressEnter();

        // 12. Retrieve the key policy.
        echo "\n";
        echo "12. Retrieve the key policy.\n";
        echo "Let's get some information about the new policy and print it to the
 screen.\n";
        $this->pressEnter();
        $policyInfo = $this->kmsService->getKeyPolicy($keyInfo['KeyId']);
        echo "We got the info! Here is the policy: \n";
        echo $policyInfo['Policy'] . "\n";
        $this->pressEnter();

        // 13. Create an asymmetric KMS key and sign data.
        echo "\n";
        echo "13. Create an asymmetric KMS key and sign data.\n";
        echo "Signing your data with an AWS key can provide several benefits that
 make it an attractive option for your data signing needs.\n";
        echo "By using an AWS KMS key, you can leverage the security controls and
 compliance features provided by AWS, which can help you meet various regulatory
 requirements and enhance the overall security posture of your organization.\n";
        echo "First we'll create the asymmetric key.\n";
        $this->pressEnter();
        $keySpec = "RSA_2048";
        $keyUsage = "SIGN_VERIFY";
        $asymmetricKey = $this->kmsService->createKey($keySpec, $keyUsage);
        $this->resources['asymmetricKey'] = $asymmetricKey['KeyId'];
        echo "Created the key with ID: {$asymmetricKey['KeyId']}\n";
        echo "Next, we'll sign the data.\n";
        $this->pressEnter();
        $algorithm = "RSASSA_PSS_SHA_256";
        $sign = $this->kmsService->sign($asymmetricKey['KeyId'], $text, $algorithm);
```

```
        $verify = $this->kmsService->verify($asymmetricKey['KeyId'], $text,
$sign['Signature'], $algorithm);
        echo "Signature verification result: {$sign['signature']}\n";
        $this->pressEnter();

        // 14. Tag the symmetric KMS key.
        echo "\n";
        echo "14. Tag the symmetric KMS key.\n";
        echo "By using tags, you can improve the overall management, security,
and governance of your KMS keys, making it easier to organize, track, and control
access to your encrypted data within your AWS environment.\n";
        echo "Let's tag our symmetric key as Environment->Production\n";
        $this->pressEnter();
        $this->kmsService->tagResource($key['KeyId'], [
            [
                'TagKey' => "Environment",
                'TagValue' => "Production",
            ],
        ]);
        echo "The key was successfully tagged!\n";
        $this->pressEnter();

        // 15. Schedule the deletion of the KMS key
        echo "\n";
        echo "15. Schedule the deletion of the KMS key.\n";
        echo "By default, KMS applies a waiting period of 30 days, but you can
specify a waiting period of 7-30 days.\n";
        echo "When this operation is successful, the key state of the KMS key
changes to PendingDeletion and the key can't be used in any cryptographic
operations.\n";
        echo "It remains in this state for the duration of the waiting period.\n\n";

        echo "Deleting a KMS key is a destructive and potentially dangerous
operation. When a KMS key is deleted, all data that was encrypted under the KMS key
is unrecoverable.\n\n";

        $cleanUp = testable_readline("Would you like to delete the resources created
during this scenario, including the keys? (y/n): ");
        if($cleanUp == "Y" || $cleanUp == "y"){
            $this->cleanUp();
        }
```

```
        echo
 "--------------------------------------------------------------------------------
\n";
        echo "This concludes the AWS Key Management SDK Basics scenario\n";
        echo
 "--------------------------------------------------------------------------------
\n";



namespace Kms;

use Aws\Kms\Exception\KmsException;
use Aws\Kms\KmsClient;
use Aws\Result;
use Aws\ResultPaginator;
use AwsUtilities\AWSServiceClass;

class KmsService extends AWSServiceClass
{

    protected KmsClient $client;
    protected bool $verbose;

    /***
     * @param KmsClient|null $client
     * @param bool $verbose
     */
    public function __construct(KmsClient $client = null, bool $verbose = false)
    {
        $this->verbose = $verbose;
        if($client){
            $this->client = $client;
            return;
        }
        $this->client = new KmsClient([]);
    }


    /***
     * @param string $keySpec
     * @param string $keyUsage
     * @param string $description
     * @return array
```

```
     */
    public function createKey(string $keySpec = "", string $keyUsage = "", string
 $description = "Created by the SDK for PHP")
    {
        $parameters = ['Description' => $description];
        if($keySpec && $keyUsage){
            $parameters['KeySpec'] = $keySpec;
            $parameters['KeyUsage'] = $keyUsage;
        }
        try {
            $result = $this->client->createKey($parameters);
            return $result['KeyMetadata'];
        }catch(KmsException $caught){
            // Check for error specific to createKey operations
            if ($caught->getAwsErrorMessage() == "LimitExceededException"){
                echo "The request was rejected because a quota was exceeded. For
 more information, see Quotas in the Key Management Service Developer Guide.";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $ciphertext
     * @param string $algorithm
     * @return Result
     */
    public function decrypt(string $keyId, string $ciphertext, string $algorithm =
 "SYMMETRIC_DEFAULT")
    {
        try{
            return $this->client->decrypt([
                'CiphertextBlob' => $ciphertext,
                'EncryptionAlgorithm' => $algorithm,
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem decrypting the data: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
```

```
    }



    /***
     * @param string $keyId
     * @param string $text
     * @return Result
     */
    public function encrypt(string $keyId, string $text)
    {
        try {
            return $this->client->encrypt([
                'KeyId' => $keyId,
                'Plaintext' => $text,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "DisabledException"){
                echo "The request was rejected because the specified KMS key is not
enabled.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param int $limit
     * @return ResultPaginator
     */
    public function listAliases(string $keyId = "", int $limit = 0)
    {
        $args = [];
        if($keyId){
            $args['KeyId'] = $keyId;
        }
        if($limit){
            $args['Limit'] = $limit;
        }
        try{
            return $this->client->getPaginator("ListAliases", $args);
        }catch(KmsException $caught){
```

```
            if($caught->getAwsErrorMessage() == "InvalidMarkerException"){
                echo "The request was rejected because the marker that specifies
where pagination should next begin is not valid.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $alias
     * @return void
     */
    public function createAlias(string $keyId, string $alias)
    {
        try{
            $this->client->createAlias([
                'TargetKeyId' => $keyId,
                'AliasName' => $alias,
            ]);
        }catch (KmsException $caught){
            if($caught->getAwsErrorMessage() == "InvalidAliasNameException"){
                echo "The request was rejected because the specified alias name is
not valid.";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $granteePrincipal
     * @param array $operations
     * @param array $grantTokens
     * @return Result
     */
    public function createGrant(string $keyId, string $granteePrincipal, array
$operations, array $grantTokens = [])
    {
        $args = [
```

```
            'KeyId' => $keyId,
            'GranteePrincipal' => $granteePrincipal,
            'Operations' => $operations,
        ];
        if($grantTokens){
            $args['GrantTokens'] = $grantTokens;
        }
        try{
            return $this->client->createGrant($args);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "InvalidGrantTokenException"){
                echo "The request was rejected because the specified grant token is
not valid.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @return array
     */
    public function describeKey(string $keyId)
    {
        try {
            $result = $this->client->describeKey([
                "KeyId" => $keyId,
            ]);
            return $result['KeyMetadata'];
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "The request was rejected because the specified entity or
resource could not be found.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
```

```
     * @return void
     */
    public function disableKey(string $keyId)
    {
        try {
            $this->client->disableKey([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem disabling the key: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @return void
     */
    public function enableKey(string $keyId)
    {
        try {
            $this->client->enableKey([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "The request was rejected because the specified entity or
 resource could not be found.\n";
            }
            throw $caught;
        }
    }



    /***
     * @return array
     */
    public function listKeys()
    {
        try {
```

```
            $contents = [];
            $paginator = $this->client->getPaginator("ListKeys");
            foreach($paginator as $result){
                foreach ($result['Content'] as $object) {
                    $contents[] = $object;
                }
            }
            return $contents;
        }catch(KmsException $caught){
            echo "There was a problem listing the keys: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @return Result
     */
    public function listGrants(string $keyId)
    {
        try{
            return $this->client->listGrants([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "    The request was rejected because the specified entity or
 resource could not be found.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @return Result
     */
    public function getKeyPolicy(string $keyId)
    {
        try {
```

```
            return $this->client->getKeyPolicy([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem getting the key policy: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }


    /***
     * @param string $grantId
     * @param string $keyId
     * @return void
     */
    public function revokeGrant(string $grantId, string $keyId)
    {
        try{
            $this->client->revokeGrant([
                'GrantId' => $grantId,
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem with revoking the grant: {$caught-
>getAwsErrorMessage()}.\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param int $pendingWindowInDays
     * @return void
     */
    public function scheduleKeyDeletion(string $keyId, int $pendingWindowInDays = 7)
    {
        try {
            $this->client->scheduleKeyDeletion([
                'KeyId' => $keyId,
                'PendingWindowInDays' => $pendingWindowInDays,
            ]);
```

```
        }catch(KmsException $caught){
            echo "There was a problem scheduling the key deletion: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param array $tags
     * @return void
     */
    public function tagResource(string $keyId, array $tags)
    {
        try {
            $this->client->tagResource([
                'KeyId' => $keyId,
                'Tags' => $tags,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem applying the tag(s): {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $message
     * @param string $algorithm
     * @return Result
     */
    public function sign(string $keyId, string $message, string $algorithm)
    {
        try {
            return $this->client->sign([
                'KeyId' => $keyId,
                'Message' => $message,
                'SigningAlgorithm' => $algorithm,
            ]);
```

```
        }catch(KmsException $caught){
            echo "There was a problem signing the data: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param int $rotationPeriodInDays
     * @return void
     */
    public function enableKeyRotation(string $keyId, int $rotationPeriodInDays =
 365)
    {
        try{
            $this->client->enableKeyRotation([
                'KeyId' => $keyId,
                'RotationPeriodInDays' => $rotationPeriodInDays,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "The request was rejected because the specified entity or
 resource could not be found.\n";
            }
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $policy
     * @return void
     */
    public function putKeyPolicy(string $keyId, string $policy)
    {
        try {
            $this->client->putKeyPolicy([
                'KeyId' => $keyId,
                'Policy' => $policy,
```

```
            ]);
        }catch(KmsException $caught){
            echo "There was a problem replacing the key policy: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $aliasName
     * @return void
     */
    public function deleteAlias(string $aliasName)
    {
        try {
            $this->client->deleteAlias([
                'AliasName' => $aliasName,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem deleting the alias: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }



    /***
     * @param string $keyId
     * @param string $message
     * @param string $signature
     * @param string $signingAlgorithm
     * @return bool
     */
    public function verify(string $keyId, string $message, string $signature, string
 $signingAlgorithm)
    {
        try {
            $result = $this->client->verify([
                'KeyId' => $keyId,
                'Message' => $message,
                'Signature' => $signature,
```

```
                'SigningAlgorithm' => $signingAlgorithm,
            ]);
            return $result['SignatureValid'];
        }catch(KmsException $caught){
            echo "There was a problem verifying the signature: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }


}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CreateAlias
  - CreateGrant
  - CreateKey
  - Decrypt
  - DescribeKey
  - DisableKey
  - EnableKey
  - Encrypt
  - GetKeyPolicy
  - ListAliases
  - ListGrants
  - ListKeys
  - RevokeGrant
  - ScheduleKeyDeletion
  - Sign
  - TagResource

操作

`CreateAlias`

以下代码示例演示了如何使用 `CreateAlias`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param string $alias
     * @return void
     */
    public function createAlias(string $keyId, string $alias)
    {
        try{
            $this->client->createAlias([
                'TargetKeyId' => $keyId,
                'AliasName' => $alias,
            ]);
        }catch (KmsException $caught){
            if($caught->getAwsErrorMessage() == "InvalidAliasNameException"){
                echo "The request was rejected because the specified alias name is
not valid.";
            }
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateAlias中的。

CreateGrant

以下代码示例演示了如何使用 CreateGrant。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /***
    * @param string $keyId
    * @param string $granteePrincipal
    * @param array $operations
    * @param array $grantTokens
    * @return Result
    */
   public function createGrant(string $keyId, string $granteePrincipal, array
$operations, array $grantTokens = [])
   {
       $args = [
           'KeyId' => $keyId,
           'GranteePrincipal' => $granteePrincipal,
           'Operations' => $operations,
       ];
       if($grantTokens){
           $args['GrantTokens'] = $grantTokens;
       }
       try{
           return $this->client->createGrant($args);
       }catch(KmsException $caught){
           if($caught->getAwsErrorMessage() == "InvalidGrantTokenException"){
               echo "The request was rejected because the specified grant token is
not valid.\n";
           }
           throw $caught;
       }
   }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateGrant中的。

`CreateKey`

以下代码示例演示了如何使用 `CreateKey`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /***
    * @param string $keySpec
    * @param string $keyUsage
    * @param string $description
    * @return array
    */
   public function createKey(string $keySpec = "", string $keyUsage = "", string
$description = "Created by the SDK for PHP")
   {
       $parameters = ['Description' => $description];
       if($keySpec && $keyUsage){
           $parameters['KeySpec'] = $keySpec;
           $parameters['KeyUsage'] = $keyUsage;
       }
       try {
           $result = $this->client->createKey($parameters);
           return $result['KeyMetadata'];
       }catch(KmsException $caught){
           // Check for error specific to createKey operations
           if ($caught->getAwsErrorMessage() == "LimitExceededException"){
               echo "The request was rejected because a quota was exceeded. For
more information, see Quotas in the Key Management Service Developer Guide.";
           }
           throw $caught;
```

```
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateKey中的。

`Decrypt`

以下代码示例演示了如何使用 `Decrypt`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param string $ciphertext
     * @param string $algorithm
     * @return Result
     */
    public function decrypt(string $keyId, string $ciphertext, string $algorithm =
 "SYMMETRIC_DEFAULT")
    {
        try{
            return $this->client->decrypt([
                'CiphertextBlob' => $ciphertext,
                'EncryptionAlgorithm' => $algorithm,
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem decrypting the data: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Decrypt。

DeleteAlias

以下代码示例演示了如何使用 DeleteAlias。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $aliasName
     * @return void
     */
    public function deleteAlias(string $aliasName)
    {
        try {
            $this->client->deleteAlias([
                'AliasName' => $aliasName,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem deleting the alias: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteAlias中的。

DescribeKey

以下代码示例演示了如何使用 DescribeKey。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @return array
     */
    public function describeKey(string $keyId)
    {
        try {
            $result = $this->client->describeKey([
                "KeyId" => $keyId,
            ]);
            return $result['KeyMetadata'];
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "The request was rejected because the specified entity or
resource could not be found.\n";
            }
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DescribeKey中的。

`DisableKey`

以下代码示例演示了如何使用 `DisableKey`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @return void
     */
    public function disableKey(string $keyId)
    {
        try {
            $this->client->disableKey([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem disabling the key: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DisableKey中的。

`EnableKey`

以下代码示例演示了如何使用 `EnableKey`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @return void
     */
    public function enableKey(string $keyId)
    {
        try {
            $this->client->enableKey([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "The request was rejected because the specified entity or
resource could not be found.\n";
            }
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考EnableKey中的。

`Encrypt`

以下代码示例演示了如何使用 `Encrypt`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param string $text
     * @return Result
```

```
     */
    public function encrypt(string $keyId, string $text)
    {
        try {
            return $this->client->encrypt([
                'KeyId' => $keyId,
                'Plaintext' => $text,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "DisabledException"){
                echo "The request was rejected because the specified KMS key is not
enabled.\n";
            }
            throw $caught;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Encrypt。

`ListAliases`

以下代码示例演示了如何使用 `ListAliases`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param int $limit
     * @return ResultPaginator
     */
    public function listAliases(string $keyId = "", int $limit = 0)
    {
```

```
        $args = [];
        if($keyId){
            $args['KeyId'] = $keyId;
        }
        if($limit){
            $args['Limit'] = $limit;
        }
        try{
            return $this->client->getPaginator("ListAliases", $args);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "InvalidMarkerException"){
                echo "The request was rejected because the marker that specifies
where pagination should next begin is not valid.\n";
            }
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListAliases中的。

`ListGrants`

以下代码示例演示了如何使用 `ListGrants`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @return Result
     */
    public function listGrants(string $keyId)
    {
```

```
        try{
            return $this->client->listGrants([
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            if($caught->getAwsErrorMessage() == "NotFoundException"){
                echo "    The request was rejected because the specified entity or
resource could not be found.\n";
            }
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListGrants中的。

`ListKeys`

以下代码示例演示了如何使用 `ListKeys`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @return array
     */
    public function listKeys()
    {
        try {
            $contents = [];
            $paginator = $this->client->getPaginator("ListKeys");
            foreach($paginator as $result){
                foreach ($result['Content'] as $object) {
                    $contents[] = $object;
                }
```

```
            }
            return $contents;
        }catch(KmsException $caught){
            echo "There was a problem listing the keys: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListKeys中的。

PutKeyPolicy

以下代码示例演示了如何使用 PutKeyPolicy。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param string $policy
     * @return void
     */
    public function putKeyPolicy(string $keyId, string $policy)
    {
        try {
            $this->client->putKeyPolicy([
                'KeyId' => $keyId,
                'Policy' => $policy,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem replacing the key policy: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
```

```
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考PutKeyPolicy中的。

`RevokeGrant`

以下代码示例演示了如何使用 `RevokeGrant`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $grantId
     * @param string $keyId
     * @return void
     */
    public function revokeGrant(string $grantId, string $keyId)
    {
        try{
            $this->client->revokeGrant([
                'GrantId' => $grantId,
                'KeyId' => $keyId,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem with revoking the grant: {$caught-
>getAwsErrorMessage()}.\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考RevokeGrant中的。

ScheduleKeyDeletion

以下代码示例演示了如何使用 ScheduleKeyDeletion。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param int $pendingWindowInDays
     * @return void
     */
    public function scheduleKeyDeletion(string $keyId, int $pendingWindowInDays = 7)
    {
        try {
            $this->client->scheduleKeyDeletion([
                'KeyId' => $keyId,
                'PendingWindowInDays' => $pendingWindowInDays,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem scheduling the key deletion: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ScheduleKeyDeletion中的。

Sign

以下代码示例演示了如何使用 Sign。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param string $message
     * @param string $algorithm
     * @return Result
     */
    public function sign(string $keyId, string $message, string $algorithm)
    {
        try {
            return $this->client->sign([
                'KeyId' => $keyId,
                'Message' => $message,
                'SigningAlgorithm' => $algorithm,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem signing the data: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Sign。

`TagResource`

以下代码示例演示了如何使用 `TagResource`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /***
     * @param string $keyId
     * @param array $tags
     * @return void
     */
    public function tagResource(string $keyId, array $tags)
    {
        try {
            $this->client->tagResource([
                'KeyId' => $keyId,
                'Tags' => $tags,
            ]);
        }catch(KmsException $caught){
            echo "There was a problem applying the tag(s): {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考TagResource中的。

使用适用于 PHP 的 SDK 的 Lambda 示例

以下代码示例向您展示了如何使用 with Lambda 来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建 IAM 角色和 Lambda 函数，然后上传处理程序代码。
- 使用单个参数来调用函数并获取结果。
- 更新函数代码并使用环境变量进行配置。
- 使用新参数来调用函数并获取结果。显示返回的执行日志。
- 列出账户函数，然后清除函数。

有关更多信息，请参阅使用控制台创建 Lambda 函数。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
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";
        $bucketName = "amzn-s3-demo-bucket-$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";
    }
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

操作

`CreateFunction`

以下代码示例演示了如何使用 `CreateFunction`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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",
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateFunction中的。

`DeleteFunction`

以下代码示例演示了如何使用 `DeleteFunction`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteFunction中的。

`GetFunction`

以下代码示例演示了如何使用 `GetFunction`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function getFunction($functionName)
{
    return $this->lambdaClient->getFunction([
        'FunctionName' => $functionName,
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetFunction中的。

`Invoke`

以下代码示例演示了如何使用 `Invoke`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public function invoke($functionName, $params, $logType = 'None')
{
    return $this->lambdaClient->invoke([
        'FunctionName' => $functionName,
        'Payload' => json_encode($params),
        'LogType' => $logType,
    ]);
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Invoke。

`ListFunctions`

以下代码示例演示了如何使用 `ListFunctions`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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,
    ]);
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListFunctions中的。

`UpdateFunctionCode`

以下代码示例演示了如何使用 `UpdateFunctionCode`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
    {
        return $this->lambdaClient->updateFunctionCode([
            'FunctionName' => $functionName,
            'S3Bucket' => $s3Bucket,
            'S3Key' => $s3Key,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考UpdateFunctionCode中的。

`UpdateFunctionConfiguration`

以下代码示例演示了如何使用 `UpdateFunctionConfiguration`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
    {
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
```

```
            'Environment' => $environment,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考[UpdateFunctionConfiguration](#)中的。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

无服务器示例

使用 Lambda 函数连接到 Amazon RDS 数据库

以下代码示例显示如何实现连接到 RDS 数据库的 Lambda 函数。该函数发出一个简单的数据库请求并返回结果。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

在 Lambda 函数中使用 PHP 连接到 Amazon RDS 数据库。

```
<?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);
```

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 Kinesis 事件与 Lambda 结合使用。

```
// 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);
```

通过 DynamoDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DynamoDB 流的记录而触发的事件。该函数检索 DynamoDB 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 DynamoDB 事件与 Lambda 结合使用。

```
<?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);
```

通过 Amazon DocumentDB 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 DocumentDB 更改流的记录而触发的事件。该函数检索 DocumentDB 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 Amazon DocumentDB 事件与 Lambda 结合使用。

```
<?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();
```

通过 Amazon MSK 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 Amazon MSK 集群的记录而触发的事件。该函数检索 MSK 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 Amazon MSK 事件与 Lambda 结合使用。

```
<?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\Kafka\KafkaEvent;
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): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
```

```
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }

            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收通过将对象上传到 S3 桶而触发的事件。该函数从事件参数中检索 S3 存储桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 S3 事件与 Lambda 结合使用。

```
<?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);
```

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 SNS 事件与 Lambda 结合使用。

```
// 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();
```

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 队列的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 SQS 事件与 Lambda 结合使用。

```
// 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);
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 Kinesis 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告通过 PHP 进行 Lambda Kinesis 批处理项目失败。

```
// 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);
```

通过 DynamoDB 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为接收来自 DynamoDB 流的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 PHP 通过 Lambda 进行 DynamoDB 批处理项目失败。

```
<?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);
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 SQS 队列的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 PHP 进行 Lambda SQS 批处理项目失败。

```
// 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);
```

使用适用于 PHP 的 SDK 的 Amazon MSK 示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon MSK 配合使用来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 无服务器示例

无服务器示例

通过 Amazon MSK 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收通过接收来自 Amazon MSK 集群的记录而触发的事件。该函数检索 MSK 有效负载，并记录下记录内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 Amazon MSK 事件与 Lambda 结合使用。

```
<?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\Kafka\KafkaEvent;
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): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }

            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

使用适用于 PHP 的 SDK 的 Amazon RDS 示例

以下代码示例向您展示了如何在 Amazon RDS 中使用来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

操作

`CreateDBInstance`

以下代码示例演示了如何使用 `CreateDBInstance`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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";
}
```

- 有关 API 的详细信息，请参阅DBInstance在 适用于 PHP 的 Amazon SDK API 参考中创建。

`CreateDBSnapshot`

以下代码示例演示了如何使用 `CreateDBSnapshot`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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";
}
```

- 有关 API 的详细信息，请参阅DBSnapshot在 适用于 PHP 的 Amazon SDK API 参考中创建。

`DeleteDBInstance`

以下代码示例演示了如何使用 `DeleteDBInstance`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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";
}
```

- 有关 API 的详细信息，请参阅DBInstance《适用于 PHP 的 Amazon SDK API 参考》中的 “删除”。

`DescribeDBInstances`

以下代码示例演示了如何使用 `DescribeDBInstances`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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";
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DBInstances中的描述。

场景

创建 Aurora Serverless 工作项跟踪器

以下代码示例演示如何创建 Web 应用程序，来跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service（Amazon SES）发送报告。

适用于 PHP 的 SDK

演示如何使用创建一个 Web 应用程序，该 适用于 PHP 的 Amazon SDK 应用程序通过亚马逊简单电子邮件服务 (Amazon SES) Semple Service 跟踪亚马逊 RDS 数据库中的工作项目并通过电子邮件发送报告。此示例使用使用 React.js 构建的前端与 RESTful PHP 后端进行交互。

- 将 React.js 网络应用程序与 Amazon 服务集成。
- 列出、添加、更新和删除 Amazon RDS 表中的项目。
- 使用 Amazon SES 以电子邮件发送已筛选工作项的报告。
- 使用随附的 Amazon CloudFormation 脚本部署和管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Aurora
- Amazon RDS

- Amazon RDS 数据服务
- Amazon SES

无服务器示例

使用 Lambda 函数连接到 Amazon RDS 数据库

以下代码示例显示如何实现连接到 RDS 数据库的 Lambda 函数。该函数发出一个简单的数据库请求并返回结果。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

在 Lambda 函数中使用 PHP 连接到 Amazon RDS 数据库。

```
<?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);
```

使用适用于 PHP 的 SDK 的 Amazon RDS 数据服务示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon RDS 数据服务配合使用来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

场景

创建 Aurora Serverless 工作项跟踪器

以下代码示例演示如何创建 Web 应用程序，来跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service（Amazon SES）发送报告。

适用于 PHP 的 SDK

演示如何使用创建一个 Web 应用程序，该 适用于 PHP 的 Amazon SDK 应用程序通过亚马逊简单电子邮件服务 (Amazon SES) Semple Service 跟踪亚马逊 RDS 数据库中的工作项目并通过电子邮件发送报告。此示例使用使用 React.js 构建的前端与 RESTful PHP 后端进行交互。

- 将 React.js 网络应用程序与 Amazon 服务集成。
- 列出、添加、更新和删除 Amazon RDS 表中的项目。
- 使用 Amazon SES 以电子邮件发送已筛选工作项的报告。
- 使用随附的 Amazon CloudFormation 脚本部署和管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

使用适用于 PHP 的 SDK 的 Amazon Rekognition 示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon Rekognition 配合使用来执行操作和实现常见场景。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

使用适用于 PHP 的 SDK 的 Amazon S3 示例

以下代码示例向您展示了如何在 Amazon S3 中使用来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

基本功能是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

开始使用

开始使用 Amazon S3

以下代码示例显示了如何开始使用 Amazon S3。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
use Aws\S3\S3Client;

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

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListBuckets中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建桶并将文件上载到其中。
- 从桶中下载对象。
- 将对象复制到存储桶中的子文件夹。
- 列出存储桶中的对象。
- 删除存储桶及其对象。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
        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 = "amzn-s3-demo-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' => $this->bucketName,
            ]);
            if (isset($check['Contents']) && count($check['Contents']) > 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";
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

操作

`CopyObject`

以下代码示例演示了如何使用 `CopyObject`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

对象的简单副本。

```
        $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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CopyObject中的。

`CreateBucket`

以下代码示例演示了如何使用 `CreateBucket`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建存储桶。

```
        $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.");
```

```
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateBucket中的。

`DeleteBucket`

以下代码示例演示了如何使用 `DeleteBucket`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除空桶。

```
        $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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteBucket中的。

`DeleteObject`

以下代码示例演示了如何使用 `DeleteObject`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   public function deleteObject(string $bucketName, string $fileName, array $args =
[])
   {
       $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $fileName],
$args);
       try {
           $this->client->deleteObject($parameters);
           if ($this->verbose) {
               echo "Deleted the object named: $fileName from $bucketName.\n";
           }
       } catch (AwsException $exception) {
           if ($this->verbose) {
               echo "Failed to delete $fileName from $bucketName with error:
{$exception->getMessage()}\n";
               echo "Please fix error with object deletion before continuing.";
           }
           throw $exception;
       }
   }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteObject中的。

`DeleteObjects`

以下代码示例演示了如何使用 `DeleteObjects`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从键列表中删除一组对象。

```
        $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' => $this->bucketName,
            ]);
            if (isset($check['Contents']) && count($check['Contents']) > 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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteObjects中的。

GetObject

以下代码示例演示了如何使用 GetObject。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取对象。

```
        $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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetObject中的。

ListObjectsV2

以下代码示例演示了如何使用 ListObjectsV2。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出存储桶中的对象。

```
        $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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考中的 ListObjectsV2。

`PutObject`

以下代码示例演示了如何使用 `PutObject`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将对象上传到存储桶。

```
        $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.");
        }
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考PutObject中的。

场景

创建预签名 URL

以下代码示例展示了如何为 Amazon S3 创建预签名 URL 以及如何上传对象。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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();
```

```
namespace S3;

use Aws\CommandInterface;
use Aws\Exception\AwsException;
use Aws\Result;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use AwsUtilities\AWSServiceClass;
use DateTimeInterface;

class S3Service extends AWSServiceClass
{
    protected S3Client $client;
    protected bool $verbose;

    public function __construct(S3Client $client = null, $verbose = false)
    {
        if ($client) {
            $this->client = $client;
        } else {
            $this->client = new S3Client([
                'version' => 'latest',
                'region' => 'us-west-2',
            ]);
        }
        $this->verbose = $verbose;
    }

    public function setVerbose($verbose)
    {
        $this->verbose = $verbose;
    }

    public function isVerbose(): bool
    {
        return $this->verbose;
    }

    public function getClient(): S3Client
    {
        return $this->client;
    }
```

```
    public function setClient(S3Client $client)
    {
        $this->client = $client;
    }


    public function emptyAndDeleteBucket($bucketName, array $args = [])
    {
        try {
            $objects = $this->listAllObjects($bucketName, $args);
            $this->deleteObjects($bucketName, $objects, $args);
            if ($this->verbose) {
                echo "Deleted all objects and folders from $bucketName.\n";
            }
            $this->deleteBucket($bucketName, $args);
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete $bucketName with error: {$exception-
>getMessage()}\n";
                echo "\nPlease fix error with bucket deletion before continuing.\n";
            }
            throw $exception;
        }
    }



    public function createBucket(string $bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);
        try {
            $this->client->createBucket($parameters);
            if ($this->verbose) {
                echo "Created the bucket named: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket creation before continuing.";
            }
            throw $exception;
        }
```

```
    }



    public function putObject(string $bucketName, string $key, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key], $args);
        try {
            $this->client->putObject($parameters);
            if ($this->verbose) {
                echo "Uploaded the object named: $key to the bucket named:
 $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create $key in $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with object uploading before continuing.";
            }
            throw $exception;
        }
    }



    public function getObject(string $bucketName, string $key, array $args = []):
 Result
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key], $args);
        try {
            $object = $this->client->getObject($parameters);
            if ($this->verbose) {
                echo "Downloaded the object named: $key to the bucket named:
 $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to download $key from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object downloading before continuing.";
            }
            throw $exception;
        }
        return $object;
```

```
    }



    public function copyObject($bucketName, $key, $copySource, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key,
 "CopySource" => $copySource], $args);
        try {
            $this->client->copyObject($parameters);
            if ($this->verbose) {
                echo "Copied the object from: $copySource in $bucketName to: $key.
\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to copy $copySource in $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object copying before continuing.";
            }
            throw $exception;
        }
    }



    public function listObjects(string $bucketName, $start = 0, $max = 1000, array
 $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Marker' => $start,
 "MaxKeys" => $max], $args);
        try {
            $objects = $this->client->listObjectsV2($parameters);
            if ($this->verbose) {
                echo "Retrieved the list of objects from: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to retrieve the objects from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with list objects before continuing.";
            }
            throw $exception;
        }
```

```
        return $objects;
    }



    public function listAllObjects($bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);

        $contents = [];
        $paginator = $this->client->getPaginator("ListObjectsV2", $parameters);

        foreach ($paginator as $result) {
            if($result['KeyCount'] == 0){
                break;
            }
            foreach ($result['Contents'] as $object) {
                $contents[] = $object;
            }
        }
        return $contents;
    }



    public function deleteObjects(string $bucketName, array $objects, array $args =
[])
    {
        $listOfObjects = array_map(
            function ($object) {
                return ['Key' => $object];
            },
            array_column($objects, 'Key')
        );
        if(!$listOfObjects){
            return;
        }

        $parameters = array_merge(['Bucket' => $bucketName, 'Delete' => ['Objects'
=> $listOfObjects]], $args);
        try {
            $this->client->deleteObjects($parameters);
            if ($this->verbose) {
                echo "Deleted the list of objects from: $bucketName.\n";
```

```
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete the list of objects from $bucketName with
 error: {$exception->getMessage()}\n";
                echo "Please fix error with object deletion before continuing.";
            }
            throw $exception;
        }
    }



    public function deleteBucket(string $bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);
        try {
            $this->client->deleteBucket($parameters);
            if ($this->verbose) {
                echo "Deleted the bucket named: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket deletion before continuing.";
            }
            throw $exception;
        }
    }



    public function deleteObject(string $bucketName, string $fileName, array $args =
 [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $fileName],
 $args);
        try {
            $this->client->deleteObject($parameters);
            if ($this->verbose) {
                echo "Deleted the object named: $fileName from $bucketName.\n";
            }
        } catch (AwsException $exception) {
```

```
            if ($this->verbose) {
                echo "Failed to delete $fileName from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object deletion before continuing.";
            }
            throw $exception;
        }
    }



    public function listBuckets(array $args = [])
    {
        try {
            $buckets = $this->client->listBuckets($args);
            if ($this->verbose) {
                echo "Retrieved all " . count($buckets) . "\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to retrieve bucket list with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket lists before continuing.";
            }
            throw $exception;
        }
        return $buckets;
    }



    public function preSignedUrl(CommandInterface $command, DateTimeInterface|int|
string $expires, array $options = [])
    {
        $request = $this->client->createPresignedRequest($command, $expires,
 $options);
        try {
            $presignedUrl = (string)$request->getUri();
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create a presigned url: {$exception-
>getMessage()}\n";
                echo "Please fix error with presigned urls before continuing.";
            }
```

```
            throw $exception;
        }
        return $presignedUrl;
    }



    public function createSession(string $bucketName)
    {
        try{
            $result = $this->client->createSession([
                'Bucket' => $bucketName,
            ]);
            return $result;
        }catch(S3Exception $caught){
            if($caught->getAwsErrorType() == "NoSuchBucket"){
                echo "The specified bucket does not exist.";
            }
            throw $caught;
        }
    }

}
```

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

无服务器示例

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收通过将对象上传到 S3 桶而触发的事件。该函数从事件参数中检索 S3 存储桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 S3 事件与 Lambda 结合使用。

```
<?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);
```

使用适用于 PHP 的 SDK 的 S3 目录存储桶示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 S3 Directory Buckets 配合使用来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 基本功能

基本功能

了解基本功能

以下代码示例展示了如何：

- 设置 VPC 和 VPC 端点。
- 设置策略、角色和用户，以使用 S3 目录存储桶和 S3 Express One Zone 存储类别。
- 创建两个 S3 客户端。
- 创建两个存储桶。
- 创建一个对象并复制它。
- 演示性能差异。
- 填充存储桶以显示按字典顺序的差异。
- 提示用户查看他们是否要清除资源。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行一个用于演示 Amazon S3 目录存储桶和 S3 Express One Zone 的基础知识的场景。

```
        echo "\n";
        echo "-------------------------------------\n";
        echo "Welcome to the Amazon S3 Express Basics demo using PHP!\n";
        echo "-------------------------------------\n";

        // Change these both of these values to use a different region/availability
zone.
        $region = "us-west-2";
        $az = "usw2-az1";
```

```
        $this->s3Service = new S3Service(new S3Client(['region' => $region]));
        $this->iamService = new IAMService(new IamClient(['region' => $region]));

        $uuid = uniqid();

        echo <<<INTRO
Let's get started! First, please note that S3 Express One Zone works best when
 working within the AWS infrastructure,
specifically when working in the same Availability Zone. To see the best results in
 this example, and when you implement
Directory buckets into your infrastructure, it is best to put your Compute resources
 in the same AZ as your Directory
bucket.\n
INTRO;
        pressEnter();
        // 1. Configure a gateway VPC endpoint. This is the recommended method to
 allow S3 Express One Zone traffic without
        // the need to pass through an internet gateway or NAT device.
        echo "\n";
        echo "1. First, we'll set up a new VPC and VPC Endpoint if this program is
 running in an EC2 instance in the same AZ as your Directory buckets will be.\n";
        $ec2Choice = testable_readline("Are you running this in an EC2 instance
 located in the same AZ as your intended Directory buckets? Enter Y/y to setup a VPC
 Endpoint, or N/n/blank to skip this section.");
        if($ec2Choice == "Y" || $ec2Choice == "y") {
            echo "Great! Let's set up a VPC, retrieve the Route Table from it, and
 create a VPC Endpoint to connect the S3 Client to.\n";
            pressEnter();
            $this->ec2Service = new EC2Service(new Ec2Client(['region' =>
 $region]));
            $cidr = "10.0.0.0/16";
            $vpc = $this->ec2Service->createVpc($cidr);
            $this->resources['vpcId'] = $vpc['VpcId'];

            $this->ec2Service->waitForVpcAvailable($vpc['VpcId']);

            $routeTable = $this->ec2Service->describeRouteTables([], [
                [
                    'Name' => "vpc-id",
                    'Values' => [$vpc['VpcId']],
                ],
            ]);
```

```
            $serviceName = "com.amazonaws." . $this->ec2Service->getRegion() .
 ".s3express";
            $vpcEndpoint = $this->ec2Service->createVpcEndpoint($serviceName,
 $vpc['VpcId'], [$routeTable[0]]);
            $this->resources['vpcEndpointId'] = $vpcEndpoint['VpcEndpointId'];
        }else{
            echo "Skipping the VPC setup. Don't forget to use this in production!
\n";
        }

        // 2. Policies, user, and roles with CDK.
        echo "\n";
        echo "2. Policies, users, and roles with CDK.\n";
        echo "Now, we'll set up some policies, roles, and a user. This user will
 only have permissions to do S3 Express One Zone actions.\n";
        pressEnter();

        $this->cloudFormationClient = new CloudFormationClient([]);
        $stackName = "cfn-stack-s3-express-basics-" . uniqid();
        $file = file_get_contents(__DIR__ . "/../../../../resources/cfn/
s3_express_basics/s3_express_template.yml");
        $result = $this->cloudFormationClient->createStack([
            'StackName' => $stackName,
            'TemplateBody' => $file,
            'Capabilities' => ['CAPABILITY_IAM'],
        ]);
        $waiter = $this->cloudFormationClient->getWaiter("StackCreateComplete",
 ['StackName' => $stackName]);
        try {
            $waiter->promise()->wait();
        }catch(CloudFormationException $caught){
            echo "Error waiting for the CloudFormation stack to create: {$caught-
>getAwsErrorMessage()}\n";
            throw $caught;
        }
        $this->resources['stackName'] = $stackName;
        $stackInfo = $this->cloudFormationClient->describeStacks([
            'StackName' => $result['StackId'],
        ]);

        $expressUserName = "";
        $regularUserName = "";
        foreach($stackInfo['Stacks'][0]['Outputs'] as $output) {
            if ($output['OutputKey'] == "RegularUser") {
```

```
                $regularUserName = $output['OutputValue'];
            }
            if ($output['OutputKey'] == "ExpressUser") {
                $expressUserName = $output['OutputValue'];
            }
        }
        $regularKey = $this->iamService->createAccessKey($regularUserName);
        $regularCredentials = new Credentials($regularKey['AccessKeyId'],
$regularKey['SecretAccessKey']);
        $expressKey = $this->iamService->createAccessKey($expressUserName);
        $expressCredentials = new Credentials($expressKey['AccessKeyId'],
$expressKey['SecretAccessKey']);

        // 3. Create an additional client using the credentials with S3 Express
permissions.
        echo "\n";
        echo "3. Create an additional client using the credentials with S3 Express
permissions.\n";
        echo "This client is created with the credentials associated with the
user account with the S3 Express policy attached, so it can perform S3 Express
operations.\n";
        pressEnter();
        $s3RegularClient = new S3Client([
            'Region' => $region,
            'Credentials' => $regularCredentials,
        ]);
        $s3RegularService = new S3Service($s3RegularClient);
        $s3ExpressClient = new S3Client([
            'Region' => $region,
            'Credentials' => $expressCredentials,
        ]);
        $s3ExpressService = new S3Service($s3ExpressClient);
        echo "All the roles and policies were created an attached to the user. Then,
a new S3 Client and Service were created using that user's credentials.\n";
        echo "We can now use this client to make calls to S3 Express operations.
Keeping permissions in mind (and adhering to least-privilege) is crucial to S3
Express.\n";
        pressEnter();

        // 4. Create two buckets.
        echo "\n";
        echo "3. Create two buckets.\n";
        echo "Now we will create a Directory bucket, which is the linchpin of the S3
Express One Zone service.\n";
```

```
        echo "Directory buckets behave in different ways from regular S3 buckets,
which we will explore here.\n";
        echo "We'll also create a normal bucket, put an object into the normal
bucket, and copy it over to the Directory bucket.\n";
        pressEnter();

        // Create a directory bucket. These are different from normal S3 buckets in
subtle ways.
        $directoryBucketName = "s3-express-demo-directory-bucket-$uuid--$az--x-s3";
        echo "Now, let's create the actual Directory bucket, as well as a regular
bucket.\n";
        pressEnter();
        $s3ExpressService->createBucket($directoryBucketName, [
            'CreateBucketConfiguration' => [
                'Bucket' => [
                    'Type' => "Directory", // This is what causes S3 to create a
Directory bucket as opposed to a normal bucket.
                    'DataRedundancy' => "SingleAvailabilityZone",
                ],
                'Location' => [
                    'Name' => $az,
                    'Type' => "AvailabilityZone",
                ],
            ],
        ]);
        $this->resources['directoryBucketName'] = $directoryBucketName;

        // Create a normal bucket.
        $normalBucketName = "normal-bucket-$uuid";
        $s3RegularService->createBucket($normalBucketName);
        $this->resources['normalBucketName'] = $normalBucketName;
        echo "Great! Both buckets were created.\n";
        pressEnter();

        // 5. Create an object and copy it over.
        echo "\n";
        echo "5. Create an object and copy it over.\n";
        echo "We'll create a basic object consisting of some text and upload it to
the normal bucket.\n";
        echo "Next, we'll copy the object into the Directory bucket using the
regular client.\n";
        echo "This works fine, because Copy operations are not restricted for
Directory buckets.\n";
        pressEnter();
```

```
        $objectKey = "basic-text-object";
        $s3RegularService->putObject($normalBucketName, $objectKey, $args = ['Body'
=> "Look Ma, I'm a bucket!"]);
        $this->resources['objectKey'] = $objectKey;

        // Create a session to access the directory bucket. The SDK Client will
automatically refresh this as needed.
        $s3ExpressService->createSession($directoryBucketName);
        $s3ExpressService->copyObject($directoryBucketName, $objectKey,
"$normalBucketName/$objectKey");

        echo "It worked! It's important to remember the user permissions when
interacting with Directory buckets.\n";
        echo "Instead of validating permissions on every call as normal buckets do,
Directory buckets utilize the user credentials and session token to validate.\n";
        echo "This allows for much faster connection speeds on every call. For
single calls, this is low, but for many concurrent calls, this adds up to a lot of
time saved.\n";
        pressEnter();

        // 6. Demonstrate performance difference.
        echo "\n";
        echo "6. Demonstrate performance difference.\n";
        $downloads = 1000;
        echo "Now, let's do a performance test. We'll download the same object
from each bucket $downloads times and compare the total time needed. Note: the
performance difference will be much more pronounced if this example is run in an
EC2 instance in the same AZ as the bucket.\n";
        $downloadChoice = testable_readline("If you would like to download each
object $downloads times, press enter. Otherwise, enter a custom amount and press
enter.");
        if($downloadChoice && is_numeric($downloadChoice) && $downloadChoice <
1000000){ // A million is enough. I promise.
            $downloads = $downloadChoice;
        }

        // Download the object $downloads times from each bucket and time it to
demonstrate the speed difference.
        $directoryStartTime = hrtime(true);
        for($i = 0; $i < $downloads; ++$i){
            $s3ExpressService->getObject($directoryBucketName, $objectKey);
        }
        $directoryEndTime = hrtime(true);
```

```
        $directoryTimeDiff = $directoryEndTime - $directoryStartTime;

        $normalStartTime = hrtime(true);
        for($i = 0; $i < $downloads; ++$i){
            $s3RegularService->getObject($normalBucketName, $objectKey);
        }
        $normalEndTime = hrtime(true);
        $normalTimeDiff = $normalEndTime - $normalStartTime;

        echo "The directory bucket took $directoryTimeDiff nanoseconds, while the
 normal bucket took $normalTimeDiff.\n";
        echo "That's a difference of " . ($normalTimeDiff - $directoryTimeDiff) .
 " nanoseconds, or " . (($normalTimeDiff - $directoryTimeDiff)/1000000000) . "
 seconds.\n";
        pressEnter();

        // 7. Populate the buckets to show the lexicographical difference.
        echo "\n";
        echo "7. Populate the buckets to show the lexicographical difference.\n";
        echo "Now let's explore how Directory buckets store objects in a different
 manner to regular buckets.\n";
        echo "The key is in the name \"Directory!\"\n";
        echo "Where regular buckets store their key/value pairs in a flat manner,
 Directory buckets use actual directories/folders.\n";
        echo "This allows for more rapid indexing, traversing, and therefore
 retrieval times!\n";
        echo "The more segmented your bucket is, with lots of directories, sub-
directories, and objects, the more efficient it becomes.\n";
        echo "This structural difference also causes ListObjects to behave
 differently, which can cause unexpected results.\n";
        echo "Let's add a few more objects with layered directories as see how the
 output of ListObjects changes.\n";
        pressEnter();

        // Populate a few more files in each bucket so that we can use ListObjects
 and show the difference.
        $otherObject = "other/$objectKey";
        $altObject = "alt/$objectKey";
        $otherAltObject = "other/alt/$objectKey";
        $s3ExpressService->putObject($directoryBucketName, $otherObject);
        $s3RegularService->putObject($normalBucketName, $otherObject);
        $this->resources['otherObject'] = $otherObject;
        $s3ExpressService->putObject($directoryBucketName, $altObject);
        $s3RegularService->putObject($normalBucketName, $altObject);
```

```
        $this->resources['altObject'] = $altObject;
        $s3ExpressService->putObject($directoryBucketName, $otherAltObject);
        $s3RegularService->putObject($normalBucketName, $otherAltObject);
        $this->resources['otherAltObject'] = $otherAltObject;

        $listDirectoryBucket = $s3ExpressService->listObjects($directoryBucketName);
        $listNormalBucket = $s3RegularService->listObjects($normalBucketName);

        // Directory bucket content
        echo "Directory bucket content\n";
        foreach($listDirectoryBucket['Contents'] as $result){
            echo $result['Key'] . "\n";
        }

        // Normal bucket content
        echo "\nNormal bucket content\n";
        foreach($listNormalBucket['Contents'] as $result){
            echo $result['Key'] . "\n";
        }

        echo "Notice how the normal bucket lists objects in lexicographical order,
 while the directory bucket does not. This is because the normal bucket considers
 the whole \"key\" to be the object identifies, while the directory bucket actually
 creates directories and uses the object \"key\" as a path to the object.\n";
        pressEnter();

        echo "\n";
        echo "That's it for our tour of the basic operations for S3 Express One
 Zone.\n";
        $cleanUp = testable_readline("Would you like to delete all the resources
 created during this demo? Enter Y/y to delete all the resources.");
        if($cleanUp){
            $this->cleanUp();
        }



namespace S3;

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

```
use AwsUtilities\AWSServiceClass;
use DateTimeInterface;

class S3Service extends AWSServiceClass
{
    protected S3Client $client;
    protected bool $verbose;

    public function __construct(S3Client $client = null, $verbose = false)
    {
        if ($client) {
            $this->client = $client;
        } else {
            $this->client = new S3Client([
                'version' => 'latest',
                'region' => 'us-west-2',
            ]);
        }
        $this->verbose = $verbose;
    }

    public function setVerbose($verbose)
    {
        $this->verbose = $verbose;
    }

    public function isVerbose(): bool
    {
        return $this->verbose;
    }

    public function getClient(): S3Client
    {
        return $this->client;
    }

    public function setClient(S3Client $client)
    {
        $this->client = $client;
    }


    public function emptyAndDeleteBucket($bucketName, array $args = [])
    {
```

```
        try {
            $objects = $this->listAllObjects($bucketName, $args);
            $this->deleteObjects($bucketName, $objects, $args);
            if ($this->verbose) {
                echo "Deleted all objects and folders from $bucketName.\n";
            }
            $this->deleteBucket($bucketName, $args);
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete $bucketName with error: {$exception-
>getMessage()}\n";
                echo "\nPlease fix error with bucket deletion before continuing.\n";
            }
            throw $exception;
        }
    }



    public function createBucket(string $bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);
        try {
            $this->client->createBucket($parameters);
            if ($this->verbose) {
                echo "Created the bucket named: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket creation before continuing.";
            }
            throw $exception;
        }
    }



    public function putObject(string $bucketName, string $key, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key], $args);
        try {
            $this->client->putObject($parameters);
```

```
            if ($this->verbose) {
                echo "Uploaded the object named: $key to the bucket named:
 $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create $key in $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with object uploading before continuing.";
            }
            throw $exception;
        }
    }



    public function getObject(string $bucketName, string $key, array $args = []):
 Result
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key], $args);
        try {
            $object = $this->client->getObject($parameters);
            if ($this->verbose) {
                echo "Downloaded the object named: $key to the bucket named:
 $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to download $key from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object downloading before continuing.";
            }
            throw $exception;
        }
        return $object;
    }



    public function copyObject($bucketName, $key, $copySource, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $key,
 "CopySource" => $copySource], $args);
        try {
```

```
            $this->client->copyObject($parameters);
            if ($this->verbose) {
                echo "Copied the object from: $copySource in $bucketName to: $key.
\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to copy $copySource in $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object copying before continuing.";
            }
            throw $exception;
        }
    }



    public function listObjects(string $bucketName, $start = 0, $max = 1000, array
 $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Marker' => $start,
 "MaxKeys" => $max], $args);
        try {
            $objects = $this->client->listObjectsV2($parameters);
            if ($this->verbose) {
                echo "Retrieved the list of objects from: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to retrieve the objects from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with list objects before continuing.";
            }
            throw $exception;
        }
        return $objects;
    }



    public function listAllObjects($bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);
```

```
        $contents = [];
        $paginator = $this->client->getPaginator("ListObjectsV2", $parameters);

        foreach ($paginator as $result) {
            if($result['KeyCount'] == 0){
                break;
            }
            foreach ($result['Contents'] as $object) {
                $contents[] = $object;
            }
        }
        return $contents;
    }



    public function deleteObjects(string $bucketName, array $objects, array $args =
[])
    {
        $listOfObjects = array_map(
            function ($object) {
                return ['Key' => $object];
            },
            array_column($objects, 'Key')
        );
        if(!$listOfObjects){
            return;
        }

        $parameters = array_merge(['Bucket' => $bucketName, 'Delete' => ['Objects'
=> $listOfObjects]], $args);
        try {
            $this->client->deleteObjects($parameters);
            if ($this->verbose) {
                echo "Deleted the list of objects from: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete the list of objects from $bucketName with
error: {$exception->getMessage()}\n";
                echo "Please fix error with object deletion before continuing.";
            }
            throw $exception;
        }
```

```
    }



    public function deleteBucket(string $bucketName, array $args = [])
    {
        $parameters = array_merge(['Bucket' => $bucketName], $args);
        try {
            $this->client->deleteBucket($parameters);
            if ($this->verbose) {
                echo "Deleted the bucket named: $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete $bucketName with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket deletion before continuing.";
            }
            throw $exception;
        }
    }



    public function deleteObject(string $bucketName, string $fileName, array $args =
 [])
    {
        $parameters = array_merge(['Bucket' => $bucketName, 'Key' => $fileName],
 $args);
        try {
            $this->client->deleteObject($parameters);
            if ($this->verbose) {
                echo "Deleted the object named: $fileName from $bucketName.\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to delete $fileName from $bucketName with error:
 {$exception->getMessage()}\n";
                echo "Please fix error with object deletion before continuing.";
            }
            throw $exception;
        }
    }
```

```
    public function listBuckets(array $args = [])
    {
        try {
            $buckets = $this->client->listBuckets($args);
            if ($this->verbose) {
                echo "Retrieved all " . count($buckets) . "\n";
            }
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to retrieve bucket list with error: {$exception-
>getMessage()}\n";
                echo "Please fix error with bucket lists before continuing.";
            }
            throw $exception;
        }
        return $buckets;
    }



    public function preSignedUrl(CommandInterface $command, DateTimeInterface|int|
string $expires, array $options = [])
    {
        $request = $this->client->createPresignedRequest($command, $expires,
 $options);
        try {
            $presignedUrl = (string)$request->getUri();
        } catch (AwsException $exception) {
            if ($this->verbose) {
                echo "Failed to create a presigned url: {$exception-
>getMessage()}\n";
                echo "Please fix error with presigned urls before continuing.";
            }
            throw $exception;
        }
        return $presignedUrl;
    }



    public function createSession(string $bucketName)
    {
```

```
        try{
            $result = $this->client->createSession([
                'Bucket' => $bucketName,
            ]);
            return $result;
        }catch(S3Exception $caught){
            if($caught->getAwsErrorType() == "NoSuchBucket"){
                echo "The specified bucket does not exist.";
            }
            throw $caught;
        }
    }

}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的以下主题。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObject
  - GetObject
  - ListObjects
  - PutObject

使用适用于 PHP 的 SDK 的 Amazon SES 示例

以下代码示例向您展示了如何在 Amazon SES 中使用来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 场景

场景

创建 Aurora Serverless 工作项跟踪器

以下代码示例演示如何创建 Web 应用程序，来跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service（Amazon SES）发送报告。

适用于 PHP 的 SDK

演示如何使用创建一个 Web 应用程序，该 适用于 PHP 的 Amazon SDK 应用程序通过亚马逊简单电子邮件服务 (Amazon SES) Semple Service 跟踪亚马逊 RDS 数据库中的工作项目并通过电子邮件发送报告。此示例使用使用 React.js 构建的前端与 RESTful PHP 后端进行交互。

- 将 React.js 网络应用程序与 Amazon 服务集成。
- 列出、添加、更新和删除 Amazon RDS 表中的项目。
- 使用 Amazon SES 以电子邮件发送已筛选工作项的报告。
- 使用随附的 Amazon CloudFormation 脚本部署和管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

使用适用于 PHP 的 SDK 的 Amazon SNS 示例

以下代码示例向您展示了如何在 Amazon SNS 中使用来执行操作和实现常见场景。 适用于 PHP 的 Amazon SDK

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 Amazon Web Services 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

操作

CheckIfPhoneNumberIsOptedOut

以下代码示例演示了如何使用 CheckIfPhoneNumberIsOptedOut。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-sending-sms.html#check-if-a-phone-number-has-opted-out。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CheckIfPhoneNumberIsOptedOut中的。

`ConfirmSubscription`

以下代码示例演示了如何使用 `ConfirmSubscription`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ConfirmSubscription中的。

`CreateTopic`

以下代码示例演示了如何使用 `CreateTopic`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-managing-topics.html#create-a-topic。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考CreateTopic中的。

DeleteTopic

以下代码示例演示了如何使用 DeleteTopic。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考DeleteTopic中的。

GetSMSAttributes

以下代码示例演示了如何使用 GetSMSAttributes。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-sending-sms.html#get-sms-attributes。
- 有关 API 的详细信息，请参阅 Get SMSAttributes in 适用于 PHP 的 Amazon SDK API 参考。

`GetTopicAttributes`

以下代码示例演示了如何使用 `GetTopicAttributes`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考GetTopicAttributes中的。

`ListPhoneNumbersOptedOut`

以下代码示例演示了如何使用 `ListPhoneNumbersOptedOut`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-sending-sms.html#list-opted-out-phone-numbers。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListPhoneNumbersOptedOut中的。

`ListSubscriptions`

以下代码示例演示了如何使用 `ListSubscriptions`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListSubscriptions中的。

`ListTopics`

以下代码示例演示了如何使用 `ListTopics`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考ListTopics中的。

`Publish`

以下代码示例演示了如何使用 `Publish`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-subscribing-unsubscribing-topics.html#publish-a-message-to-an-sns-topic。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考中的 Publish。

`SetSMSAttributes`

以下代码示例演示了如何使用 `SetSMSAttributes`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
$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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-sending-sms.html#set-sms-attributes。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考SMSAttributes中的设置。

`SetTopicAttributes`

以下代码示例演示了如何使用 `SetTopicAttributes`。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考SetTopicAttributes中的。

Subscribe

以下代码示例演示了如何使用 Subscribe。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将电子邮件地址订阅到主题。

```
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());
}
```

将 HTTP 端点订阅到主题。

```
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());
}
```

- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Subscribe。

Unsubscribe

以下代码示例演示了如何使用 Unsubscribe。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-subscribing-unsubscribing-topics.html#unsubscribe-from-a-topic。
- 有关 API 详细信息，请参阅《适用于 PHP 的 Amazon SDK API Reference》中的 Unsubscribe。

场景

创建无服务器应用程序来管理照片

以下代码示例演示如何创建无服务器应用程序，让用户能够使用标签管理照片。

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

发布 SMS 文本消息

以下代码示例展示了如何使用 Amazon SNS 来发布 SMS 消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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());
}
```

- 有关更多信息，请参阅《适用于 PHP 的 Amazon SDK 开发人员指南》https://docs.amazonaws.cn/sdk-for-php/v3/developer-guide/sns-examples-sending-sms.html#publish-to-a-text-message-sms-message。
- 有关 API 的详细信息，请参阅 适用于 PHP 的 Amazon SDK API 参考中的 Publish。

无服务器示例

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

通过 PHP 将 SNS 事件与 Lambda 结合使用。

```
// 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();
```

使用适用于 PHP 的 SDK 的 Amazon SQS 示例

以下代码示例向您展示了如何使用 适用于 PHP 的 Amazon SDK 与 Amazon SQS 配合使用来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 无服务器示例

无服务器示例

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例展示了如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 队列的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

使用 PHP 将 SQS 事件与 Lambda 结合使用。

```
// 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);
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例展示了如何为接收来自 SQS 队列的事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在无服务器示例存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 PHP 进行 Lambda SQS 批处理项目失败。

```
// 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);
```

从版本 2 迁移 适用于 PHP 的 Amazon SDK

本主题展示如何迁移代码，以使用 适用于 PHP 的 Amazon SDK 的版本 3，并介绍新版本与开发工具包版本 2 的不同之处。

> **Note**
>
> 开发工具包的基本使用模式（即 `$result = $client->operation($params);`）从版本 2 到版本 3 并未改变，这样可确保顺利迁移。

简介

版本 3 适用于 PHP 的 Amazon SDK 代表了为改进 SDK 的功能、整合两年多的客户反馈、升级我们的依赖关系、提高性能和采用最新的 PHP 标准所做的巨大努力。

版本 3 中有哪些新功能？

的第 3 版 适用于 PHP 的 Amazon SDK 遵循 PSR-4 和 PSR-7 标准，今后将遵循该SemVer标准。

其他新增功能包括：

- 用于自定义服务客户端行为的中间件系统
- 灵活的分页工具，用于迭代分页结果
- 能够通过以下方式从结果和分页器对象中查询数据 JMESPath
- 通过 `'debug'` 配置选项轻松调试

分离的 HTTP 层

- 默认使用 Guzzle 6 发送请求，但也支持 Guzzle 5。
- 在 cURL 不可用的环境中开发工具包也可以工作。
- 也支持自定义 HTTP 处理程序。

异步请求

- Waiter 和分段上传工具等功能也可异步使用。

- 使用 Promise 和协同程序可创建异步工作流程。
- 提升了并发或批处理请求的性能。

与版本 2 有何不同之处？

更新了项目依赖项

开发工具包在此版本中更改了依赖项。

- 开发工具包现在需要 PHP 8.1 及以上。可以自由地在开发工具包代码中使用生成器。
- 我们已将 SDK 升级为使用 Guzzle 6（或 5），它提供了 SDK 用来向服务发送请求的底层 HTTP 客户端实现。 Amazon Guzzle 的最新版本引入了一系列改进，包括异步请求、可交换 HTTP 处理程序、PSR-7 合规性、更出色的性能等。
- 来自 PHP-FIG (`psr/http-message`) 的 PSR-7 包定义了用于表示 HTTP 请求、HTTP 响应和流的接口。 URLs这些接口是开发工具包和 Guzzle 通用的，可与兼容 PSR-7 的其他程序包互通。
- Guzzle 的 PSR-7 实现 (`guzzlehttp/psr7`) 在 PSR-7 中提供接口实现，还提供若干有用的类和函数。开发工具包和 Guzzle 6 均在很大程度上依赖这个程序包。
- Guzzle 的 Promises/A+ 实现 (`guzzlehttp/promises`) 是开发工具包和 Guzzle 通用的，可提供管理异步请求和协同程序的接口。虽然 Guzzle 的 Multi-curl HTTP 处理程序最终实现了允许异步请求的非阻塞 I/O 模型，但该软件包提供了在该范式中进行编程的能力。有关更多详细信息，请参阅适用于 PHP 的 Amazon SDK 版本 3 中的 Promis es。
- SDK 中使用 JMESPath(`mtdowling/jmespath.php`) 的 PHP 实现来提供`Aws\Result::search()`和`Aws\ResultPaginator::search()`方法的数据查询能力。有关更多详细信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 中的JMESPath 表达式。

现在要求提供区域和版本选项

如果要将任何服务的客户端实例化，请指定 `'region'` 和 `'version'` 选项。在的版本 2 中 适用于 PHP 的 Amazon SDK，`'version'`是完全可选的，有时`'region'`是可选的。在版本 3 中，这两个选项都是必需的。明确说明这两个选项可以让你锁定你要编码的 API 版本和 Amazon 区域。当新的 API 版本创建或新的 Amazon 区域可用时，在您准备好显式更新配置之前，您将与可能发生的重大更改隔离开来。

> **Note**
>
> 如果您不关心所用的 API 版本，只需将 `'version'` 选项设为 `'latest'`。但是，我们建议您针对生产代码显式设置 API 版本号。
> 并非所有服务在所有 Amazon 地区都可用。您可以参考区域和端点，找到可用区域的列表。对于只能通过单个全球终端节点提供的服务（例如 Amazon Route 53 和 Amazon CloudFront），请实例化客户端，并将配置的区域设置为。 Amazon Identity and Access Managementus-east-1

> **Important**
>
> SDK 还包括多区域客户端，它们可以根据作为命令参数提供的参数 (`@region`) 将请求分发到不同的 Amazon 区域。这些客户端默认使用的区域是由客户端构造函数提供的 `region` 选项指定的。

使用构造函数进行客户端实例化

在的版本 3 中 适用于 PHP 的 Amazon SDK，实例化客户端的方式发生了变化。您只需使用 `factory` 关键字即可将客户端实例化，而在版本 2 中需使用 `new` 方法。

```
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**
>
> 仍可使用 `factory()` 方法将客户端实例化。但这种方法已被弃用。

客户端配置已更改

版本 3 中的客户端配置选项与版本 2 相比略 适用于 PHP 的 Amazon SDK 有变化。有关所有支持的选项的描述，请参阅 适用于 PHP 的 Amazon SDK 版本 3 的配置页面。

> **Important**
>
> 在版本 3 中，`'key'` 和 `'secret'` 不再是根级别的有效选项，但可以作为 `'credentials'` 选项的一部分进行传递。我们这样做的原因之一是为了阻止开发人员将 Amazon 凭证硬编码到他们的项目中。

Sdk 对象

的版本 3 适用于 PHP 的 Amazon SDK 引入了该`Aws\Sdk`对象作为替换对象`Aws\Common\Aws`。`Sdk` 对象可作为客户端工厂，用于管理多个客户端的共享配置选项。

开发工具包版本 2 中的 `Aws` 类像一个服务定位器（它总是会返回客户端的同一实例），而版本 3 中的 `Sdk` 类在每次使用时都会返回客户端的一个新实例。

`Sdk` 对象也不支持与开发工具包版本 2 相同的配置文件格式。原来的配置格式是 Guzzle 3 的特定格式，现已过时。可利用基本数组更轻松地进行配置，配置方法在使用 Sdk 类中进行了记录。

更改了一些 API 结果

为了保持软件开发工具包解析 API 操作结果的方式的一致性，亚马逊 ElastiCache、亚马逊 RDS 和 Amazon Redshift 现在在某些 API 响应上增加了包装元素。

例如，在版本 3 中调用 Amazon RDS DescribeEngineDefaultParameters结果现在包括一个封装“EngineDefaults” 元素。在版本 2 中没有此元素。

```
$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'];
```

以下操作会受到影响，在结果输出中会包含包装元素（在下文的括号中提供）：

- Amazon ElastiCache
  - AuthorizeCacheSecurityGroupIngress (CacheSecurityGroup)
  - CopySnapshot（快照）
  - CreateCacheCluster (CacheCluster)
  - CreateCacheParameterGroup (CacheParameterGroup)
  - CreateCacheSecurityGroup (CacheSecurityGroup)
  - CreateCacheSubnetGroup (CacheSubnetGroup)
  - CreateReplicationGroup (ReplicationGroup)
  - CreateSnapshot（快照）
  - DeleteCacheCluster (CacheCluster)
  - DeleteReplicationGroup (ReplicationGroup)
  - DeleteSnapshot（快照）
  - DescribeEngineDefaultParameters (EngineDefaults)
  - ModifyCacheCluster (CacheCluster)
  - ModifyCacheSubnetGroup (CacheSubnetGroup)
  - ModifyReplicationGroup (ReplicationGroup)
  - PurchaseReservedCacheNodesOffering (ReservedCacheNode)
  - RebootCacheCluster (CacheCluster)
  - RevokeCacheSecurityGroupIngress (CacheSecurityGroup)
- Amazon RDS
  - AddSourceIdentifierToSubscription (EventSubscription)
  - 授权 DBSecurityGroupIngress（DBSecurity群组）
  - 复制DBParameter组（DBParameter组）
  - 复制 DBSnapshot (DBSnapshot)
  - CopyOptionGroup (OptionGroup)

- 创建 DBInstance (DBInstance)
- 创建 DBInstance ReadReplica (DBInstance)
- 创建DBParameter群组（DBParameter群组）
- 创建DBSecurity群组（DBSecurity群组）
- 创建 DBSnapshot (DBSnapshot)
- 创建DBSubnet群组（DBSubnet群组）
- CreateEventSubscription (EventSubscription)
- CreateOptionGroup (OptionGroup)
- 删除 DBInstance (DBInstance)
- 删除 DBSnapshot (DBSnapshot)
- DeleteEventSubscription (EventSubscription)
- DescribeEngineDefaultParameters (EngineDefaults)
- 修改 DBInstance (DBInstance)
- 修改DBSubnet群组（DBSubnet群组）
- ModifyEventSubscription (EventSubscription)
- ModifyOptionGroup (OptionGroup)
- PromoteReadReplica (DBInstance)
- PurchaseReservedDBInstances发行（已保留DBInstance）
- 重启 DBInstance (DBInstance)
- RemoveSourceIdentifierFromSubscription (EventSubscription)
- DBInstance从 DBSnapshot (DBInstance) 恢复
- 恢复 DBInstance ToPointInTime (DBInstance)
- 撤销 DBSecurityGroupIngress（DBSecurity群组）

- Amazon Redshift
  - AuthorizeClusterSecurityGroupIngress (ClusterSecurityGroup)
  - AuthorizeSnapshotAccess（快照）
  - CopyClusterSnapshot（快照）
  - CreateCluster（集群）
  - CreateClusterParameterGroup (ClusterParameterGroup)

  - CreateClusterSecurityGroup (ClusterSecurityGroup)

- CreateClusterSnapshot（快照）
- CreateClusterSubnetGroup (ClusterSubnetGroup)
- CreateEventSubscription (EventSubscription)
- CreateHsmClientCertificate (HsmClientCertificate)
- CreateHsmConfiguration (HsmConfiguration)
- DeleteCluster（集群）
- DeleteClusterSnapshot（快照）
- DescribeDefaultClusterParameters (DefaultClusterParameters)
- DisableSnapshotCopy（集群）
- EnableSnapshotCopy（集群）
- ModifyCluster（集群）
- ModifyClusterSubnetGroup (ClusterSubnetGroup)
- ModifyEventSubscription (EventSubscription)
- ModifySnapshotCopyRetentionPeriod（集群）
- PurchaseReservedNodeOffering (ReservedNode)
- RebootCluster（集群）
- RestoreFromClusterSnapshot（集群）
- RevokeClusterSecurityGroupIngress (ClusterSecurityGroup)
- RevokeSnapshotAccess（快照）
- RotateEncryptionKey（集群）

已删除 Enum 类

我们已删除 适用于 PHP 的 Amazon SDK版本 2 中的 Enum 类（例如 `Aws\S3\Enum\CannedAcl`）。Enum 是开发工具包公共 API 中的具体类，其中包含的常量代表有效参数值的组合。由于这些枚举是特定于某一 API 版本的，是可变的，可能与 PHP 保留关键字冲突，因此并不是非常有用，所以我们在版本 3 中删除了它们，从而支持版本 3 的数据驱动特性，以及与 API 版本无关的特性。

请直接使用文本值，而不是使用 Enum 对象的值（例如，`CannedAcl::PUBLIC_READ` → `'public-read'`）。

已删除精细异常类

我们已基于与删除 Enum 非常类似的原因，删除了每项服务的命名空间中存在的精细异常类（例如 `Aws\Rds\Exception\{SpecificError}Exception`）。服务或操作引发的异常依赖于所用的 API 版本（各版本可能有变化）。另外，也无法提供给定操作可引发异常的完整列表，从而使版本 2 的精细异常类不完整。

通过捕获每个服务的根异常类来处理错误（例如 `Aws\Rds\Exception\RdsException`）。您可以使用异常的 `getAwsErrorCode()` 方法查看具体的错误代码。它的功能与抓住不同的异常类等效，但不会为开发工具包增加多余内容。

已删除静态 Facade 类

在的版本2中 适用于 PHP 的 Amazon SDK，有一个受Laravel启发的模糊功能，它允许你调用该Aws类来启用`enableFacades()`对各种服务客户端的静态访问。此功能不符合 PHP 最佳实践，我们在一年多前停止为此功能提供文档。在版本 3 中完全删除了此功能。请从 `Aws\Sdk` 对象中检索客户端对象，并将其用作对象实例，而不是静态类。

分页器取代迭代器

的版本 2 适用于 PHP 的 Amazon SDK 有一个名为*迭代器*的功能。这些对象用于迭代分页结果。我们收到反馈，表示它们不够灵活，因为迭代器只会发出每个结果的特定值。如果您需要结果的其他值，只能通过事件侦听器进行检索。

在版本 3 中，分页工具替代了迭代器。它们的用途类似，但分页工具更灵活。这是因为分页工具可生成结果对象，而不是响应值。

以下示例演示了在版本 2 和版本 3 中如何检索 S3 `ListObjects` 操作的分页结果，从而展示了分页工具与迭代器的不同之处。

```
// Version 2
$objects = $s3Client->getIterator('ListObjects', ['Bucket' => 'amzn-s3-demo-bucket']);
foreach ($objects as $object) {
    echo $object['Key'] . "\n";
}
```

```
// Version 3
$results = $s3Client->getPaginator('ListObjects', ['Bucket' => 'amzn-s3-demo-bucket']);
foreach ($results as $result) {
    // You can extract any data that you want from the result.
```

```
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
}
```

Paginator 对象具有一种search()方法，可让您使用JMESPath表达式更轻松地从结果集中提取数据。

```
$results = $s3Client->getPaginator('ListObjects', ['Bucket' => 'amzn-s3-demo-bucket']);
foreach ($results->search('Contents[].Key') as $key) {
    echo $key . "\n";
}
```

> **Note**
>
> 我们仍支持 getIterator() 方法，从而确保更顺畅地过渡到版本 3，但我们鼓励您迁移代码，以使用分页工具。

更改了许多高级抽象

总体而言，我们改进或更新了许多高级抽象（除客户端之外，特定服务的帮助程序对象）。还删除了一些。

- 已更新：
  - Amazon S3 分段上传的使用方式已更改。Amazon Glacier 分段上传也进行了类似更改。
  - 创建预签名的 Amazon S3 的方式 URLs已经改变。
  - Aws\S3\Sync 命名空间已替换为 Aws\S3\Transfer 类。S3Client::uploadDirectory() 和 S3Client::downloadBucket() 方法仍然可用，但有不同的选项。请参阅适用于 PHP 的 Amazon SDK 版本 3 的 Amazon S3 传输管理器的文档。
  - Aws\S3\Model\ClearBucket 和 Aws\S3\Model\DeleteObjectsBatch 已替换为 Aws\S3\BatchDelete 和 S3Client::deleteMatchingObjects()。
  - 在版本 3 中使用 DynamoDB 会话处理程序的选项和行为 适用于 PHP 的 Amazon SDK 略有变化。
  - Aws\DynamoDb\Model\BatchRequest 命名空间已替换为 Aws\DynamoDb\WriteRequestBatch。请参阅 DynamoDB WriteRequestBatch 的文档。

    - `Aws\Ses\SesClient` 现可在使用 `SendRawEmail` 操作时处理对 `RawMessage` 的 base64 编码。
- 已删除：
    - Amazon `DynamoDBItem`、`Attribute` 和 `ItemIterator` 类 — 之前在版本 2.7.0 中已弃用。
    - Amazon SNS 消息验证程序 — 现在已成为单独的轻量级项目，不需要 SDK 作为依赖项。但此项目包含在开发工具包的 Phar 和 ZIP 分发版中。您可以在 Amazon PHP 开发博客上找到入门指南。
    - Amazon `S3AcpBuilder` 及其相关对象均已删除。

比较 SDK 这两个版本的代码示例

以下示例展示了 适用于 PHP 的 Amazon SDK 版本 3 与版本 2 在使用方式上的一些不同之处。

示例：Amazon S3 **`ListObjects`** 操作

SDK 版本 2

```
<?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' => 'amzn-s3-demo-bucket',
        'Key'    => 'my-object-key'
    ]);

    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
```

```
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

SDK 版本 3

主要区别：

- 使用 new 将客户端实例化，而不是 factory()。
- 在实例化的过程中需要 'version' 和 'region' 选项。

```
<?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' => 'amzn-s3-demo-bucket',
        'Key'    => 'my-object-key'
    ]);

    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

示例：实例化具有全局配置的客户端

SDK 版本 2

```
<?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.
```

SDK 版本 3

主要区别：

- 使用 Aws\Sdk 类，而不是 Aws\Common\Aws。
- 没有配置文件。使用一组配置。

- 在实例化期间需要 'version' 选项。
- 使用 create<Service>() 方法，而不是 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.
```

的安全性 适用于 PHP 的 Amazon SDK

云安全性一直是 Amazon Web Services（Amazon）的重中之重。作为 Amazon 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。安全是双方共同承担 Amazon 的责任。责任共担模式将其描述为云的安全性和云中的安全性。

云安全 — Amazon 负责保护运行 Amazon 云中提供的所有服务的基础架构，并为您提供可以安全使用的服务。我们的安全责任是重中之重 Amazon，作为Amazon 合规计划的一部分，第三方审计师定期测试和验证我们安全的有效性。

云端安全 — 您的责任由您使用的 Amazon 服务以及其他因素决定，包括数据的敏感性、组织的要求以及适用的法律和法规。

主题

- 中的数据保护 适用于 PHP 的 Amazon SDK 版本 3
- 身份和访问管理
- 为此进行合规性验证 Amazon 产品或服务
- 为此具有韧性 Amazon 产品或服务
- 为此提供基础设施安全 Amazon 产品或服务
- Amazon S3 加密客户端迁移（从 V1 到 V2）适用于 PHP 的 Amazon SDK 版本 3
- Amazon S3 加密客户端迁移（从 V2 到 V3）适用于 PHP 的 Amazon SDK 版本 3

中的数据保护 适用于 PHP 的 Amazon SDK 版本 3

Amazon 责任共担模式 适用于保护 中的数据。如本模型所述 Amazon ，负责保护运行所有内容的全球基础架构 Amazon Web Services 云。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 Amazon Web Services 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅中国地区法律条款。

出于数据保护目的，我们建议您保护 Amazon Web Services 账户 凭证并使用 Amazon IAM Identity Center 或 Amazon Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 Amazon 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。

- 使用设置 API 和用户活动日志 Amazon CloudTrail。有关使用 CloudTrail 跟踪捕获 Amazon 活动的信息，请参阅《Amazon CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 Amazon 加密解决方案以及其中的所有默认安全控件 Amazon Web Services 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 Amazon 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://www.amazonaws.cn/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括使用控制台、API 或 Amazon SDK Amazon Web Services 服务 使用 适用于 PHP 的 Amazon SDK 版本 3 或其他版本的情况。 Amazon CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

身份和访问管理

Amazon Identity and Access Management (IAM) Amazon Web Services 服务 可帮助管理员安全地控制对 Amazon 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 Amazon 资源。您可以使用 IAM Amazon Web Services 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- 操作方法 Amazon Web Services 服务 使用 IAM
- 问题排查 Amazon 身份和访问权限

受众

您的使用方式 Amazon Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 Amazon。

服务用户-如果您 Amazon Web Services 服务 曾经完成工作，则您的管理员会为您提供所需的凭证和权限。当你使用更多 Amazon 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有

助于您向管理员请求适合的权限。如果您无法访问中的功能 Amazon，请参阅问题排查 Amazon 身份和访问权限或 Amazon Web Services 服务 您正在使用的用户指南。

服务管理员-如果您负责公司的 Amazon 资源，则可能拥有完全访问权限 Amazon。您的工作是确定您的服务用户应访问哪些 Amazon 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何使用 IAM Amazon，请参阅 Amazon Web Services 服务 您正在使用的用户指南。

IAM 管理员：如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon的访问权限的详细信息。要查看您可以在 IAM 中使用的 Amazon 基于身份的策略示例，请参阅 Amazon Web Services 服务 您正在使用的用户指南。

使用身份进行身份验证

身份验证是您 Amazon 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 Amazon Web Services 账户根用户，或者通过担任 IAM 角色进行身份验证。

对于编程访问， Amazon 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的Amazon 签名版本 4。

Amazon Web Services 账户 根用户

创建时 Amazon Web Services 账户，首先会有一个名为 Amazon Web Services 账户 root 用户的登录身份，该身份可以完全访问所有资源 Amazon Web Services 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户使用与身份提供商的联合身份验证才能 Amazon Web Services 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Amazon Directory Service ，或者 Amazon Web Services 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 Amazon 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 Amazon CLI 或 Amazon API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 Amazon 通过创建策略并将其附加到 Amazon 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 Amazon 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 Amazon 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 Amazon 托管策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 Amazon WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

Amazon 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（SCP）– 指定 Amazon Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《Amazon Organizations 用户指南》中的服务控制策略。
- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《Amazon Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 Amazon 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

操作方法 Amazon Web Services 服务 使用 IAM

要全面了解如何 Amazon Web Services 服务 使用大多数 IAM 功能，请参阅 IAM 用户指南中的与 IAM 配合使用的Amazon 服务。

要了解如何在 IAM 中 Amazon Web Services 服务 使用特定的，请参阅相关服务的《用户指南》的安全部分。

问题排查 Amazon 身份和访问权限

使用以下信息来帮助您诊断和修复在使用 Amazon 和 IAM 时可能遇到的常见问题。

主题

我无权在以下位置执行操作 Amazon

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 awes:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 awes:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给。 Amazon

有些 Amazon Web Services 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人进入 Amazon Web Services 账户 访问我的 Amazon 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 Amazon 支持这些功能，请参阅操作方法 Amazon Web Services 服务 使用 IAM。
- 要了解如何提供对您拥有的资源的访问权限 Amazon Web Services 账户 ，请参阅 IAM 用户指南中的向您拥有 Amazon Web Services 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 Amazon Web Services 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 Amazon Web Services 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

为此进行合规性验证 Amazon 产品或服务

要了解是否属于特定合规计划的范围，请参阅Amazon Web Services 服务 “” Amazon Web Services 服务 中的 “按合规计划划分的范围”，然后选择您感兴趣的合规计划。 Amazon Web Services 服务 有关一般信息，请参阅合规计划。

您可以使用下载第三方审计报告 Amazon Artifact。有关更多信息，请参阅中的 “下载报告” Amazon Artifact。

您在使用 Amazon Web Services 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 Amazon Web Services 服务，请参阅Amazon 安全文档。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

为此具有韧性 Amazon 产品或服务

Amazon 全球基础设施是围绕 Amazon Web Services 区域 可用区构建的。

Amazon Web Services 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。

利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅Amazon 全球基础设施。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

为此提供基础设施安全 Amazon 产品或服务

本 Amazon 产品或服务使用托管服务，因此受到 Amazon 全球网络安全的保护。有关 Amazon 安全服务以及如何 Amazon 保护基础设施的信息，请参阅Amazon 云安全。要使用基础设施安全的最佳实践来设计您的 Amazon 环境，请参阅 S Amazon ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 Amazon 已发布的 API 调用通过网络访问此 Amazon 产品或服务。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 Amazon Security Token Service（Amazon STS）生成临时安全凭证来对请求进行签名。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

Amazon S3 加密客户端迁移（从 V1 到 V2）适用于 PHP 的 Amazon SDK 版本 3

> **Note**
>
> 如果您使用的是 Amazon S3 加密客户端的版本 2 (V2)，并且想要迁移到版本 3 (V3)，请参阅Amazon S3 加密客户端迁移（从 V2 到 V3）适用于 PHP 的 Amazon SDK 版本 3。

此主题介绍了如何将应用程序从 Amazon Simple Storage Service (Amazon S3) 加密客户端的版本 1 (V1) 迁移到版本 2 (V2)，并确保应用程序在整个迁移过程中的可用性。

迁移概述

此迁移分为两个阶段：

1. 更新现有客户端以读取新格式。首先，将 适用于 PHP 的 Amazon SDK 的已更新版本部署到应用程序中。这允许现有 V1 加密客户端解密由新的 V2 客户端写入的对象。如果您的应用程序使用多个 Amazon SDK，则必须单独升级每个 SDK。

2. 将加密和解密客户端迁移到 V2。一旦所有 V1 加密客户端都能读取新格式，就可以将现有加密和解密客户端迁移到各自的 V2 版本。

更新现有客户端以读取新格式

V2 加密客户端使用旧版本客户端不支持的加密算法。迁移的第一步是将 V1 解密客户端更新到最新 SDK 版本。完成此步骤后，您的应用程序的 V1 客户端将能够解密由 V2 加密客户端加密的对象。有关适用于 PHP 的 Amazon SDK的每个主要版本，请参阅下文的详细信息。

Upgrading 适用于 PHP 的 Amazon SDK 版本 3

版本 3 是 适用于 PHP 的 Amazon SDK的最新版本。必须使用版本 3.148.0 或更高版本的 `aws/aws-sdk-php` 包才能完成此迁移。

从命令行安装

对于使用 Composer 来安装的项目，在 Composer 文件中，将 SDK 包更新到 SDK 的 3.148.0 版本，然后运行以下命令。

```
composer update aws/aws-sdk-php
```

使用 Phar 或 Zip 文件进行安装

使用以下方法之一。请务必将已更新的 SDK 文件放在代码所需位置，该位置由 require 语句确定。

对于使用 Phar 文件来安装的项目，请下载已更新的文件：aws.phar。

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

对于使用 Zip 文件来安装的项目，请下载已更新的文件：。

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

将加密和解密客户端迁移到 V2

更新客户端以读取新的加密格式后，您可以将应用程序更新到 V2 加密和解密客户端。以下步骤展示了如何成功地将代码从 V1 迁移到 V2。

更新到 V2 客户端的要求

1. 必须将 Amazon KMS 加密上下文传递到`S3EncryptionClientV2::putObject`和`S3EncryptionClientV2::putObjectAsync`方法中。 Amazon KMS 加密上下文是密钥值对的关联数组，必须将其添加到加密上下文中才能进行密钥加密。 Amazon KMS 如果不需要其他上下文，则可以传递空数组。

2.必须将 `@SecurityProfile` 传递到 `S3EncryptionClientV2` 中的 `getObject` 和 `getObjectAsync` 方法中。`@SecurityProfile` 是这些 `getObject...` 方法的新必填参数。如果设置为`'V2'`，则只能解密以 V2-compatible 格式加密的对象。将此参数设置为`'V2_AND_LEGACY'`还允许对以 V1-compatible格式加密的对象进行解密。要支持迁移，请将 `@SecurityProfile` 设置为`'V2_AND_LEGACY'`。`'V2'` 仅用于新应用程序开发。

3.（可选）将 `@KmsAllowDecryptWithAnyCmk` 参数包含在 `S3EncryptionClientV2::getObject` 和 `S3EncryptionClientV2::getObjectAsync*` `methods.` 中。添加了一个名为 `@KmsAllowDecryptWithAnyCmk` 的新参数。将此参数设置为 `true`，可在不提供 KMS 密钥的情况下进行解密。默认值为 `false`。

4. 要使用 V2 客户端进行解密，如果未将 @KmsAllowDecryptWithAnyCmk 参数设置为 true，则对于 "getObject..." 方法调用，必须为 KmsMaterialsProviderV2 构造函数提供 kms-key-id。

迁移示例

示例 1：迁移到 V2 客户端

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',
    ])
);
```

示例 2：使用 Amazon KMS 带有 kms-key-id

> **Note**
>
> 这些示例使用在示例 1 中定义的导入和变量。例如 $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,
]);
```

Amazon S3 加密客户端迁移（从 V2 到 V3）适用于 PHP 的 Amazon SDK 版本 3

> **Note**
>
> 如果您使用的是 Amazon S3 加密客户端的版本 1 (V1)，则必须先迁移到版本 2 (V2)，然后才能迁移到版本 3 (V3)。请参阅Amazon S3 加密客户端迁移（从 V1 到 V2）适用于 PHP 的 Amazon SDK 版本 3。

本主题介绍如何将您的应用程序从亚马逊简单存储服务 (Amazon S3) Simple Service 加密客户端的版本 2 (V2) 迁移到版本 3 (V3)，并确保应用程序在整个迁移过程中的可用性。版本 3 引入了 AES GCM 以及密钥承诺和承诺策略，以增强安全性并防止数据密钥被篡改。

迁移概述

此迁移分为两个阶段：

1. 更新现有客户端以读取新格式。首先，将 适用于 PHP 的 Amazon SDK 的已更新版本部署到应用程序中。这允许现有的 V2 加密客户端解密新 V3 客户端写入的对象。如果您的应用程序使用多个 Amazon SDK，则必须单独升级每个 SDK。

2. 将加密和解密客户端迁移到 V3。一旦您的所有 V2 加密客户端都能读取新格式，您就可以将现有的加密和解密客户端迁移到各自的 V3 版本。

理解 V3 的概念

Amazon S3 加密客户端第 3 版引入了两项关键安全增强功能：承诺策略和带密钥承诺算法的 AES GCM。了解这些概念对于成功迁移至关重要。

承诺政策

承诺策略控制加密客户端在加密和解密操作期间如何处理密钥承诺。版本 3 提供了三个策略选项：

`FORBID_ENCRYPT_ALLOW_DECRYPT`

加密行为：无需密钥承诺即可加密对象。

解密行为：允许对使用或不使用密钥承诺进行加密的对象进行解密。

安全影响：此策略不对新加密的对象强制执行密钥承诺，这可能会允许篡改数据密钥。仅在需要与 V2 客户端保持兼容性的初始迁移阶段使用此策略。

版本兼容性：所有 V2 和 V3 实现均可读取使用此策略加密的对象。

`REQUIRE_ENCRYPT_ALLOW_DECRYPT`

加密行为：使用`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`算法对具有密钥承诺的对象进行加密。

解密行为：允许对使用或不使用密钥承诺进行加密的对象进行解密。

安全影响：此策略为新加密的对象提供了增强的安全性，同时保持了读取现有对象的能力。这是大多数迁移场景的推荐策略。

版本兼容性：使用此策略加密的对象只能由 V3 和最新的 V2 实现读取。

迁移注意事项：在使用此策略之前，请确保所有需要读取加密对象的客户端都已升级到 V3 或最新的 V2。

`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`

加密行为：使用`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`算法对具有密钥承诺的对象进行加密。

解密行为：仅允许解密使用密钥承诺加密的对象。未经密钥承诺加密的对象将无法解密。

安全影响：该策略通过强制加密和解密密钥承诺来提供最高级别的安全性。只有在所有对象都已迁移后才使用此策略以使用密钥承诺。

版本兼容性：只有 V3 实现才能使用此政策。尝试使用此策略解密 V1 或 V2 加密对象将失败。

迁移注意事项：只有在完成完整迁移并使用密钥承诺重新加密所有现有对象之后，才应使用此策略。

带有关键承诺的 AES GCM

带密钥承诺的 AES GCM (`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`) 算法是 V3 中引入的一种新的加密算法，可防止数据密钥篡改攻击。

安全性增强：通过加密方式将数据密钥绑定到加密内容，`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`防止数据密钥被篡改。这可以防止攻击者在解密过程中替换不同的数据密钥，这可能会导致意外数据被解密。

版本兼容性：使用加密的对象`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`只能通过 V3 和 Amazon S3 加密客户端的最新 V2 实现进行解密。V1 客户端无法解密使用此算法加密的对象。

> **Important**
>
> 升级要求：在启用加密之前`ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY`（使用 REQUIRE_ENCRYPT_ALLOW_DECRYPT 或 REQUIRE_ENCRYPT 策略），必须确保所有需要读取加密对象的客户端都已升级到 V3。未能升级所有读取器将导致使用密钥承诺加密的对象解密失败。

更新现有客户端以读取新格式

V3 加密客户端使用旧版本的客户端不支持的加密算法和密钥承诺功能。迁移的第一步是将 V2 解密客户端更新到最新的 SDK 版本。完成此步骤后，您的应用程序的 V2 客户端将能够解密由 V3 加密客户端加密的对象。有关每种安装方法的详细信息，请参阅下文 适用于 PHP 的 Amazon SDK。

构建并安装最新的 SDK 版本

要完成此迁移，您必须使用`aws/aws-sdk-php`包含 V3 加密客户端支持的最新版本的软件包。

从 Composer 安装

对于使用 Composer 安装的项目，请在 Composer 文件中将 SDK 包更新到最新版本的 SDK，然后运行以下命令。

```
composer update aws/aws-sdk-php
```

使用 Phar 或 Zip 文件进行安装

使用以下方法之一。请务必将已更新的 SDK 文件放在代码所需位置，该位置由 require 语句确定。

对于使用 Phar 文件来安装的项目，请下载已更新的文件：`aws.phar`。

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

对于使用 Zip 文件来安装的项目，请下载已更新的文件：。

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

构建、安装和部署应用程序

更新 SDK 后，重新构建并重新部署您的应用程序，以确保所有组件都使用更新的版本。此步骤对于确保您的 V2 客户端可以读取由 V3 客户端加密的对象至关重要。

按照贵组织的标准部署程序推出更新的应用程序。在继续将加密和解密客户端迁移到 V3 之前，请确保应用程序的所有实例都已更新。

部署后，请验证您的应用程序是否仍然可以解密现有对象，并且在正常操作期间没有出现错误。这可以确认 SDK 更新已成功并且您的应用程序已准备好进入下一阶段的迁移。

将加密和解密客户端迁移到 V3

更新客户端以读取新的加密格式后，您可以将应用程序更新为 V3 加密和解密客户端。以下示例向您展示了如何成功地将代码从 V2 迁移到 V3。

使用 V3 加密客户端

V3 引入了该`S3EncryptionClientV3`类并`KmsMaterialsProviderV3`取代了 V2 的等效项。V3 的主要区别在于：

- V3 使用`KmsMaterialsProviderV3`（与 V2 相同），但在解密调用中的对象时会验证加密上下文。`GetObject`
- V3 引入了承诺策略来控制加密和解密行为。

示例：使用 KMS 加密从 V2 迁移到 V3

Pre-migration (V2)

```
use Aws\S3\Crypto\S3EncryptionClientV2;
use Aws\S3\S3Client;
use Aws\Crypto\KmsMaterialsProviderV2;
use Aws\Kms\KmsClient;

$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,
];

$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',
    '@CommitmentPolicy' => 'FORBID_ENCRYPT_ALLOW_DECRYPT',
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

迁移期间（具有向后兼容性的 V3）

```
use Aws\S3\Crypto\S3EncryptionClientV3;
use Aws\S3\S3Client;
use Aws\Crypto\KmsMaterialsProviderV3;
use Aws\Kms\KmsClient;

// Create V3 encryption client
$encryptionClient = new S3EncryptionClientV3(
    new S3Client([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ])
```

```
);

// Create encryption materials
$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV3(
   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,
   '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_ALLOW_DECRYPT',
   '@KmsEncryptionContext' => ['context-key' => 'context-value'],
   'Bucket' => $bucket,
   'Key' => $key,
   'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

$result = $encryptionClient->getObject([
   '@SecurityProfile' => 'V3_AND_LEGACY',
   '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_ALLOW_DECRYPT',
   '@MaterialsProvider' => $materialsProvider,
   '@CipherOptions' => $cipherOptions,
   'Bucket' => $bucket,
   'Key' => $key,
]);
```

Post-migration（带有关键承诺的 V3）

```
use Aws\S3\Crypto\S3EncryptionClientV3;
use Aws\S3\S3Client;
```

```
use Aws\Crypto\KmsMaterialsProviderV3;
use Aws\Kms\KmsClient;

// Create V3 encryption client
$encryptionClient = new S3EncryptionClientV3(
   new S3Client([
      'profile' => 'default',
      'region' => 'us-east-1',
      'version' => 'latest',
   ])
);

// Create encryption materials
$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV3(
   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,
   // Use the commitment policy (REQUIRE_ENCRYPT_REQUIRE_DECRYPT)
   // This encrypts with key commitment and does not decrypt V2 objects
   '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
   '@KmsEncryptionContext' => ['context-key' => 'context-value'],
   'Bucket' => $bucket,
   'Key' => $key,
   'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

$result = $encryptionClient->getObject([
   '@SecurityProfile' => 'V3',
```

```
    // Use the commitment policy (REQUIRE_ENCRYPT_REQUIRE_DECRYPT)
    // This encrypts with key commitment and does not decrypt V2 objects
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

V3 的主要区别：

- 使用 KmsMaterialsProviderV3 代替 KmsMaterialsProviderV2
- 该@KmsEncryptionContext参数仍然是putObject操作所必需的
- 该@KmsEncryptionContext参数对于getObject操作来说是可选的，它将验证提供的加密上下文是否与对象中的加密上下文相匹配。
- 该@SecurityProfile参数控制可以解密哪些加密版本。设置'V3_AND_LEGACY'为支持在迁移期间读取 V1 和 V2 加密的对象
- 该@CommitmentPolicy参数控制此操作的承诺策略。设置'FORBID_ENCRYPT_ALLOW_DECRYPT'为支持在迁移期间读取非承诺加密对象

其他示例

以下示例演示了 V3 中可用的其他配置选项，这些选项可以帮助您管理迁移过程和控制加密行为。

启用旧版 支持

在迁移过程中，您可能需要解密使用 Amazon S3 加密客户端 V1 或 V2 加密的对象。该@SecurityProfile参数控制您的 V3 客户端可以解密哪些加密版本。

何时使用此配置：当您的应用程序需要读取由 V1 或 V2 客户端加密的对象时，请使用'V3_AND_LEGACY'安全配置文件。在迁移期间，当您的存储桶中混合使用新旧加密对象时，这种情况很常见。

```
use Aws\S3\Crypto\S3EncryptionClientV3;
use Aws\S3\S3Client;
use Aws\Crypto\KmsMaterialsProviderV3;
use Aws\Kms\KmsClient;
```

```
$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV3(
    new KmsClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ]),
    $kmsKeyId
);

$encryptionClient = new S3EncryptionClientV3(
    new S3Client([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ])
);

$bucket = 'the-bucket-name';
$key = 'the-file-name';
$cipherOptions = [
    'Cipher' => 'gcm',
    'KeySize' => 256,
];

// Decrypt objects encrypted with V1, V2, or V3
$result = $encryptionClient->getObject([
    '@SecurityProfile' => 'V3_AND_LEGACY',
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_ALLOW_DECRYPT',
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    'Bucket' => $bucket,
    'Key' => $key,
]);
```

`@SecurityProfile` 参数接受以下值：

- `'V3'`（默认）：仅使用密钥承诺解密使用 V3 加密的对象
- `'V3_AND_LEGACY'`: 解密使用 V1、V2 或 V3 加密的对象

> **Important**
>
> 完成迁移并使用 V3 重新加密所有对象后，应删除该@SecurityProfile参数或将其设置为，'V3'以确保最大限度地提高安全性。

配置存储方法

Amazon S3 加密客户端可以通过两种方式存储加密元数据：存储在对象的元数据标头中或单独的指令文件中。该@MetadataStrategy参数控制使用哪种存储方法。

何时使用此配置：'INSTRUCTION_FILE'在需要保留原始对象元数据或处理有元数据大小限制的对象时使用。使用'METADATA'（默认）进行更简单的部署，在这种部署中，加密元数据可以存储在对象旁边。

```
use Aws\S3\Crypto\S3EncryptionClientV3;
use Aws\S3\S3Client;
use Aws\Crypto\KmsMaterialsProviderV3;
use Aws\Kms\KmsClient;

$kmsKeyId = 'kms-key-id';
$materialsProvider = new KmsMaterialsProviderV3(
    new KmsClient([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ]),
    $kmsKeyId
);

$encryptionClient = new S3EncryptionClientV3(
    new S3Client([
        'profile' => 'default',
        'region' => 'us-east-1',
        'version' => 'latest',
    ])
);

$bucket = 'the-bucket-name';
$key = 'the-file-name';
$cipherOptions = [
    'Cipher' => 'gcm',
```

```
    'KeySize' => 256,
];

// Store encryption metadata in a separate instruction file
$encryptionClient->putObject([
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@MetadataStrategy' => 'INSTRUCTION_FILE',
    '@KmsEncryptionContext' => ['context-key' => 'context-value'],
    'Bucket' => $bucket,
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);

// Store encryption metadata in object headers (default)
$encryptionClient->putObject([
    '@MaterialsProvider' => $materialsProvider,
    '@CipherOptions' => $cipherOptions,
    '@CommitmentPolicy' => 'REQUIRE_ENCRYPT_REQUIRE_DECRYPT',
    '@MetadataStrategy' => 'METADATA',
    '@KmsEncryptionContext' => ['context-key' => 'context-value'],
    'Bucket' => $bucket,
    'Key' => $key,
    'Body' => fopen('file-to-encrypt.txt', 'r'),
]);
```

`@MetadataStrategy` 参数接受以下值：

- `'METADATA'`（默认）：将加密元数据存储在对象的元数据标头中
- `'INSTRUCTION_FILE'`: 将加密元数据存储在带有后缀的单独指令文件中 `.instruction`

> **Note**
>
> 使用时`'INSTRUCTION_FILE'`，带有密钥承诺的 AES GCM 算法可提供额外的保护，防止数据密钥被篡改。使用`'METADATA'`存储空间的对象无法从这种额外保护中受益。

适用于 PHP 的 Amazon SDK 版本 3 的常见问题

在客户端上可以使用哪些方法？

适用于 PHP 的 Amazon SDK使用服务描述和动态 magic __call() 方法执行 API 操作。您可以在 Web 服务客户端的 API 文档中找到该客户端提供的所有方法的完整列表。

遇到 cURL SSL 证书错误该怎么办？

如果包含 cURL 和 SSL 的 CA 捆绑包已过期，使用它会发生此问题。更新服务器上的 CA 捆绑包，或直接从 cURL 网站下载最新的 CA 捆绑包，可解决此问题。

默认情况下，适用于 PHP 的 Amazon SDK 将使用编译 PHP 时配置的 CA 捆绑包。您可以修改 `openssl.cafile` PHP .ini 配置设置（设为磁盘上 CA 文件的路径），更改 PHP 使用的默认 CA 捆绑包。

客户端提供哪些 API 版本？

创建客户端时必须提供 `version` 选项。在每个客户端的 API 文档页面 ::aws-php-class:<index.html> 上均提供了可用 API 版本的列表。如果您无法加载特定 API 版本，可能需要更新 适用于 PHP 的 Amazon SDK 的副本。

您可以为“version”配置值提供字符串 `latest`，以使用客户端的 API 提供程序可以找到的最新可用 API 版本（默认 api_provider 会扫描开发工具包的 `src/data` 目录来查找 API 模型）。

> **Warning**
>
> 我们不建议在生产应用程序中使用 `latest`，因为拉取包含 API 更新的开发工具包新次要版本会使您的生产应用程序崩溃。

客户端提供哪些区域的版本？

创建客户端时需提供 `region` 选项，使用字符串值指定。有关可用 Amazon 区域和端点的列表，请参阅 Amazon Web Services 一般参考 中的 Amazon 区域和端点。

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

为什么我无法上传或下载超过 2 GB 的文件？

因为 PHP 的整数类型经过签名，许多平台使用 32 位整数，适用于 PHP 的 Amazon SDK 无法在 32 位体系（此处所指的“体系”包含 CPU、操作系统、Web 服务器和 PHP 二进制文件）中正确处理超过 2 GB 的文件。这是一个常见的 PHP 问题。对于 Microsoft Windows，只有 PHP 7 的版本支持 64 位整数。

建议的解决方案是使用 64 位 Linux 体系，例如 64 位 Amazon Linux AMI，并安装最新版本的 PHP。

有关更多信息，请参阅 PHP 文件大小：返回值。

如何查看已通过线路发送的数据？

您可以在客户端构造函数中使用 debug 选项，获得调试信息，其中包括通过线路发送的数据。如果此选项设为 `true`，执行的所有命令更改、发送的所有请求、接收到的所有响应以及处理的所有结果均会发送给 STDOUT。其中包括通过线路发送和接收的数据。

```
$s3Client = new Aws\S3\S3Client([
    'region'  => 'us-standard',
    'version' => '2006-03-01',
    'debug'   => true
]);
```

如何为请求设置任意标题？

您可以将自定义中间件添加到 `Aws\HandlerList` 或 `Aws\CommandInterface` 的 `Aws\ClientInterface`，为服务操作添加任意标题。以下示例展示了如何使用 `Aws\Middleware::mapRequest` 帮助程序方法来为特定 Amazon S3`PutObject` 操作添加 `X-Foo-Baz` 标题。

有关更多信息，请参阅 mapRequest。

如何针对任意请求签名？

您可使用 SDK 的 :aws-php-class: SignatureV4 class <class-Aws.Signature.SignatureV4.html> 来针对一个任意 :aws-php-class: PSR-7 request <class-Psr.Http.Message.RequestInterface.html> 签名。

请参阅使用 适用于 PHP 的 Amazon SDK 版本 3 来针对自定义 Amazon CloudSearch 域请求签名，了解操作方法的完整示例。

如何在发送命令之前修改命令？

您可以将自定义中间件添加到 `Aws\HandlerList` 或 `Aws\CommandInterface` 的 `Aws\ClientInterface`，在发送命令之前修改命令。以下示例展示了如何在发送命令之前为其添加自定义命令参数，即添加默认选项。此示例使用 `Aws\Middleware::mapCommand` 帮助程序方法。

有关更多信息，请参阅 mapCommand。

什么是 CredentialsException？

如果您在使用`Aws\Exception\CredentialsException`时看到 适用于 PHP 的 Amazon SDK，这就意味着没有为开发工具包提供任何凭证，也无法在环境中找到凭证。

如果您将一个没有凭证的客户端实例化，则在首次执行服务操作时，开发工具包会尝试寻找凭证。它首先会检查一些特定的环境变量，然后寻找实例配置文件凭证（仅在经过配置的 Amazon EC2 实例上提供）。如果绝对没有提供或无法找到凭证，将会引发 `Aws\Exception\CredentialsException`。

如果看到这种错误，且希望使用实例配置文件凭证，则需要确保运行 SDK 的 Amazon EC2 实例配置了适当的 IAM 角色。

如果您看到这种错误，且不希望使用实例配置文件凭证，则需要为开发工具包提供适当的凭证。

有关更多信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 的凭证。

适用于 PHP 的 Amazon SDK 是否适用于 HHVM？

适用于 PHP 的 Amazon SDK目前无法在 HHVM 上运行，并且在 HHVM 中的生成语义问题得到解决之前，将无法支持 HHVM。

如何禁用 SSL？

将客户端工厂方法的 `scheme` 参数设为“http”，可禁用 SSL。请务必注意，并不是所有服务都支持 `http` 访问。请参阅 Amazon Web Services 一般参考 中的 Amazon 区域和端点，了解区域、端点和支持方案的列表。

```
$client = new Aws\DynamoDb\DynamoDbClient([
    'version' => '2012-08-10',
    'region'  => 'us-west-2',
    'scheme'  => 'http'
]);
```

> **Warning**
>
> 与 TCP 相比，SSL 要求所有数据均需加密，完成连接握手需要的 TCP 数据包数也更多，因此禁用 SSL 可能会小幅提升性能。但禁用 SSL 后，通过线路发送的所有数据均不会加密。在禁用 SSL 之前，您必须仔细考虑安全隐患，以及通过网络窃取的潜在可能。

出现“解析错误”该怎么办？

PHP 引擎如果遇到不理解的语法，就会引发解析错误。如果尝试运行使用其他版本的 PHP 编写的代码，总会遇到这种错误。

如果您遇到解析错误，请检查系统，确保其满足 SDK 的 适用于 PHP 的 Amazon SDK 版本 3 的要求和建议。

为什么 Amazon S3 客户端会解压缩使用 gzipp 进行压缩的文件？

一些 HTTP 处理程序，包括默认的 Guzzle 6 HTTP 处理程序，在默认情况下会扩大经过压缩的响应正文。您可以将 decode_content HTTP 选项设置为 `false`，覆盖这一行为。为了向后兼容，此默认值无法改变，但我们建议您在 S3 客户端级别禁用内容解码。

请参阅 decode_content，了解如何禁用内容自动解码的示例。

如何在 Amazon S3 中禁用正文签名?

您可以将命令对象中的 ContentSHA256 参数设置为 Aws\Signature \S3SignatureV4::UNSIGNED_PAYLOAD，禁用正文签名。然后适用于 PHP 的 Amazon SDK将在规范请求中用它作为“x-amz-content-sha-256”标题和正文校验和。

```
$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);
```

如何重试在 适用于 PHP 的 Amazon SDK 中处理的方案？

适用于 PHP 的 Amazon SDK的 RetryMiddleware 可处理重试行为。对于 5xx HTTP 服务器错误状态代码，开发工具包会重试 500、502、503 和 504。

限制异常也会进行重试，包括 RequestLimitExceeded、Throttling、ProvisionedThroughputExceededException、Throttl 和 BandwidthLimitExceeded。

适用于 PHP 的 Amazon SDK 还在重试方案中将指数延迟与退避和抖动算法集成。而且，所有服务的默认重试行为均配置为 3，只有 Amazon DynamoDB 配置为 10。

如何处理具有错误代码的异常？

除了 适用于 PHP 的 Amazon SDK 自定义的 `Exception` 类以外，每个 Amazon 服务客户端都有其各自的异常类，该类继承自 AwsExceptionAwsException。您可以根据每种方法 `Errors` 部分列出的特定于 API 的错误，确定要捕获的更多具体错误类型。

的 getAwsErrorCode()`Aws\Exception\AwsException` 可提供错误代码信息。

```
$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;
    }
}
```

术语表

API 版本

服务拥有一个或多个 API 版本，您所用的版本指示哪些操作和参数有效。API 版本采用类似于日期的格式。例如，Amazon S3 的最新 API 版本为 `2006-03-01`。配置客户端对象时，指定版本。

客户端

客户端对象用于执行服务的操作。开发工具包中支持的每项服务都有一个对应的客户端对象。客户端对象具有与服务操作一一对应的方法。要开始使用客户端，请参阅the section called “创建基本服务客户端”。

命令

命令对象封装操作的执行。当您按照发出请求部分所述执行服务操作时，将不会直接处理命令对象。可以使用客户端的 `getCommand()` 方法访问命令对象，以便使用开发工具包的高级功能，如并发请求和批处理。有关更多详细信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 中的命令对象指南。

处理程序

处理程序是一个将命令和请求实际转换为结果的函数。处理程序通常发送 HTTP 请求。处理程序可由中间件组成，以增强行为。处理程序是一个函数，它接受 `Aws\CommandInterface` 和 `Psr\Http\Message\RequestInterface`，并返回用 `Aws\ResultInterface` 执行或因 `Aws\Exception\AwsException` 原因而被拒绝的 Promise。

JMESPath

JMESPath 是一种针对类 JSON 数据的查询语言。适用于 PHP 的 Amazon SDK 使用 JMESPath 表达式查询 PHP 数据结构。JMESPath 表达式可直接通过 `Aws\Result` 方法用于 `Aws\ResultPaginator` 和 `search($expression)` 对象。

中间件

中间件是一类特殊的高级函数，可对传输命令的行为进行增强，并委托给“下一个”处理程序。中间件函数接受 `Aws\CommandInterface` 和 `Psr\Http\Message\RequestInterface`，并返回用 `Aws\ResultInterface` 执行或因 `Aws\Exception\AwsException` 原因而被拒绝的 Promise。

操作

指的是服务 API 范围内的单个操作（如适用于 DynamoDB 的 `CreateTable`、适用于 Amazon EC2 的 `RunInstances`）。在开发工具包中，通过对相应服务的客户端对象调用相同名称的方法来

执行操作。执行操作涉及准备 HTTP 请求并发送至服务以及解析响应。这种执行操作的过程由开发工具包通过命令对象提取。

Paginator

某些 Amazon 服务操作会分页，并以截断的结果进行响应。例如，Amazon S3 的 `ListObjects` 操作一次最多只能返回 1000 个对象。此类操作要求使用令牌（或标记）参数发出后续请求，以检索整个结果集。Paginator 是开发工具包的一种功能，充当此流程的抽象层，使开发人员能够更轻松地使用分页的 API。可通过客户端的 `getPaginator()` 方法访问它们。有关更多详细信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 中的 Paginator 指南。

Promise

Promise 表示异步操作的最终结果。与 Promise 交互的主要方式是通过其 then 方法，该方式注册回调以接收 Promise 的最终值或无法执行该 Promise 的原因。

区域

一个或多个地理区域支持这些服务。每个区域的服务可能拥有不同的端点/URL，用于减少应用程序中的数据延迟。配置客户端对象时，提供区域，以便开发工具包确定对该服务使用的端点。

SDK

“开发工具包”一词既可指整个适用于 PHP 的 Amazon SDK库，也可指 `Aws\Sdk` 类（文档），充当每项服务的客户端对象的工厂。您还可以通过 `Sdk` 类提供一组应用于其所创建的所有客户端对象的全局配置值。

服务

参阅任何 Amazon 服务（例如 Amazon S3、Amazon DynamoDB、Amazon OpsWorks 等）的一般方式。每项服务在开发工具包中都有一个对应的客户端对象支持一个或多个 API 版本。每项服务还有一个或多个操作构成其 API。一个或多个区域支持这些服务。

Signature

执行操作时，开发工具包使用您的凭证创建请求的数字签名。该服务随后将验证签名，然后再处理您的请求。签名过程由开发工具包封装，并使用您为客户端配置的凭证自动执行。

Waiter

Waiter 是开发工具包的一种功能，使您能够更轻松地处理改变资源状态并且本质上具有最终一致性或异步的操作。例如，Amazon DynamoDB`CreateTable` 操作会立即发回响应，但要访问该表可能需要等待几秒钟。执行 Waiter 可让您一直等到资源进入特定状态（通过休眠和轮询资源的状态）。可使用客户端的 `waitUntil()` 方法访问 Waiter。有关更多详细信息，请参阅 适用于 PHP 的 Amazon SDK 版本 3 中的 Waiter 指南。

有关最新 Amazon 术语，请参阅 Amazon Web Services 一般参考 中的 Amazon 术语表。

文档历史记录

下表说明了自上次发行 适用于 PHP 的 Amazon SDK 开发人员指南以来的重要更改。

最近更改：

变更	说明	日期
目录更新	修订了目录，使其与其他 SDK 指南更加一致。	2025 年 7 月 28 日
使用校验和实现数据完整性保护	更新内容：有关自动校验和计算的详细信息。	2025 年 1 月 16 日
凭证主题修订	重新组织了主题。提供了有关默认凭证提供程序链的更多详细信息。	2025 年 1 月 10 日
亚马逊 S3 加密客户端 V3 迁移	添加了有关 Amazon S3 加密客户端从 V2 迁移到 V3 的主题	2024 年 12 月 4 日
Amazon S3 分段上传	记录可用于配置 `ObjectUploader` 和 `MultipartUploader` 子命令的“params”数组	2024 年 11 月 6 日
对象上传器	澄清 `ObjectUploader` 上可用于 S3 上传的回调的用法	2024 年 10 月 11 日
更新 Amazon S3 存储桶名称	更新了整个指南中的 S3 存储桶名称。	2024 年 9 月 30 日
Amazon EventBridge 全球终端节点	添加显示如何使用 Amazon EventBridge 全球终端节点的代码示例	2023 年 12 月 22 日
Amazon 公共运行时 (Amazon CRT)	添加一个主题，讨论适用于 PHP 的 SDK 使用 Amazon 公	2023 年 11 月 17 日

	共运行时 (Amazon CRT) 的情况。	
StreamWrapper mkdir () 更新	添加了有关通过使用 `mkdir()` 来使用桶和文件夹对象的信息。	2023 年 11 月 2 日
创建服务客户端	由于默认值为“最新”，因此通过删除“版本”参数来更新代码段。	2023 年 8 月 31 日
基本客户端创建	由于默认值为“最新”，因此通过删除“版本”参数来更新代码段。	2023 年 8 月 31 日
目录	更新了目录，使代码示例更易于访问。	2023 年 6 月 1 日
IAM 最佳实践更新	更新了指南，使其符合 IAM 最佳实践。有关更多信息，请参阅 IAM 安全最佳实践。入门更新。	2023 年 5 月 20 日
Amazon S3 分段上传	包含了同步上传的配置信息。添加了异步上传的 `add_content_md5` 上传选项。	2023 年 4 月 13 日
亚马逊 S3 目录传输	添加了 `add_content_md5` 传输选项。	2023 年 4 月 13 日
参考信息	在《工具参考指南》 Amazon SDKs 和《工具参考指南》中添加了多个指向相关详细内容的链接。更新了指南格式。	2022 年 9 月 14 日
一般清理	添加了对 Amazon SDKs 和工具参考指南的引用。更新了 Amazon Key Management Service 章节以反映术语更新。	2022 年 8 月 23 日

使用 Amazon 服务	包括上可用的代码示例列表 GitHub。	2022 年 4 月 1 日
启用 SDK 指标	删除了有关启用 SDK 指标的信息，该指标已于 2021 年 12 月 20 日弃用。	2022 年 1 月 27 日
Amazon S3 加密客户端迁移	添加了有关 Amazon S3 加密客户端迁移的主题	2020 年 8 月 7 日

较早的更改：

更改	描述	发行日期
Secrets Manager 示例	添加更多服务示例	2019 年 3 月 27 日
端点发现	终端节点发现的配置	2019 年 2 月 15 日
Amazon CloudFront	添加更多服务示例	2019 年 1 月 25 日
服务功能	开发工具包指标	2018 年 1 月 11 日
Amazon Kinesis、Amazon SNS	添加更多服务示例	2018 年 12 月 14 日
Amazon SES 示例	添加更多服务示例	2018 年 10 月 5 日
Amazon KMS 例子	添加更多服务示例	2018 年 8 月 8 日
凭据	理顺和简化凭证指南	2018 年 6 月 30 日
MediaConvert 例子	添加更多服务示例	2018 年 6 月 15 日
新的 Web 布局	文档切换到 Amazon 样式	2018 年 9 月 5 日
Amazon S3 加密	客户端加密	2017 年 11 月 17 日
Amazon S3、Amazon SQS	添加更多服务示例	2017 年 3 月 26 日
亚马逊 S3、IAM、亚马逊 EC2	添加更多服务示例	2017 年 3 月 17 日

更改	描述	发行日期
添加 凭证	添加了对 AssumeRole 和 ini 的支持	2017 年 1 月 17 日
S3; 示例	S3 多区域和预签名文章	2016 年 3 月 18 日
OpenSearch 服务和 Amazon CloudSearch	添加更多服务示例	2015 年 12 月 28 日
命令行	添加命令参数	2015 年 8 月 13 日
服务功能	为 S3 添加服务功能和 Amazon	2015 年 4 月 30 日
新开发工具包版本	已 适用于 PHP 的 Amazon SDK 发布的版本 3。	2015 年 26 月 5 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。