AWS SDK for PHP

Developer Guide

Version v1.0.0

AWS SDK for PHP: Developer Guide

AWS Developer Guide for PHP

The AWS SDK for PHP provides a PHP API for AWS infrastructure services. Using the SDK, you can build applications on top of Amazon Simple Storage Service (Amazon S3), Amazon Elastic Compute Cloud (Amazon EC2), Amazon SimpleDB, and more.

The AWS SDK for PHP includes:

AWS PHP Library
: Build PHP applications on top of APIs that take the complexity out of coding directly against a web service interface. The library provides APIs that hide much of the lower-level plumbing, including authentication, request retries, and error handling.

Examples
: Examples that show how to use the library to build applications.

Documentation
: Information about how to use the library and code samples as well as online video tutorials and reference documentation.

Getting Started (p. 4)

If you are just starting out with the AWS SDK for PHP, you should first read through the Getting Started (p. 4) section. It will guide you through setting up your development environment and introduce the samples that are included with the SDK.

Tutorials: Accessing AWS Services from PHP

- **Tutorial: Player Manager**

  The first tutorial explains how to use Amazon SimpleDB and Amazon S3 to store information about a group of people—such as players at a tennis club.
- **Tutorial: Amazon EC2 Spot Instances (p. 22)**

  Explains how to set up a request for Amazon EC2 Spot Instances, how to determine when they have completed, and how to clean up afterwards.

How-To: Useful Code for Programming AWS

These topics are shorter than the above tutorials and deal with discreet programming tasks related to Start an Amazon EC2 Instance (p. 31).

Supported Services

The AWS SDK for PHP supports the following AWS infrastructure products:

- **Compute**
  - Amazon EC2
  - Auto Scaling
  - Elastic Load Balancing
  - Amazon Elastic MapReduce
- **Content Delivery**
  - Amazon CloudFront
- **Database**
  - Amazon SimpleDB
  - Amazon RDS
- **Deployment & Management**
  - AWS Elastic Beanstalk
  - AWS CloudFormation
- **Messaging**
  - Amazon SNS
  - Amazon SQS
  - Amazon SES
- **Monitoring**
  - Amazon CloudWatch
- **Networking**
  - Amazon VPC
- **Security**
  - Identity and Access Management
  - AWS Secure Token Service
- **Storage**
  - Amazon S3
  - Import/Export

Revision History for the AWS SDK for PHP

We regularly release updates to the AWS SDK for PHP to support new services and new service features. To see what changed with a given release, you can check the release notes history.

Also, each release of the AWS SDK for PHP is published to GitHub. The comments in the commit history provide information about what changed in each commit. To view the comments associated with a commit, click on the plus sign next to that commit.

Additional Resources

The Further Resources section has pointers to other resources to assist you in programming AWS.

About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model. You are charged only for the services that you--or your applications--use. Also, to make AWS more approachable as a platform for prototyping and experimentation, AWS offers a free usage tier. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier go to Test-Driving AWS in the Free Usage Tier. To obtain an AWS account, go to to the AWS home page and click the Sign Up Now button.

Getting Started

To get started with the AWS SDK for PHP, you need to set up the following:

- AWS Account and Credentials
- PHP Development Environment
- AWS SDK for PHP

AWS Account and Credentials

To access AWS, you will need to sign up for an AWS account.

To sign up for an AWS account

1. Go to http://aws.amazon.com, and then click **Sign Up Now**.
2. Follow the on-screen instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

AWS sends you a confirmation e-mail after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to http://aws.amazon.com and clicking **My Account/Console**.

When you sign up, AWS provides you with security credentials that are specific to your account. Two of these credentials, your *access key ID* and your *secret key*, are used by the SDK whenever it accesses the services provided by AWS. The security credentials authenticate requests to the service and identify you as the sender of the request. Examples of these credentials are shown below.

- Access Key ID Example: AKIAIOSFODNN7EXAMPLE
- Secret Access Key Example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

To view your AWS access credentials

1. Go to the Amazon Web Services website at http://aws.amazon.com.
2. Click **My Account/Console**, and then click **Security Credentials**.

3. Under **Your Account**, click **Security Credentials**.
4. In the spaces provided, type your user name and password, and then click **Sign in using our secure server**.
5. Under **Access Credentials**, on the **Access Keys** tab, your access key ID is displayed. To view your secret key, under **Secret Access Key**, click **Show**.

Your secret key must remain a secret that is known only by you and AWS. Keep it confidential in order to protect your account. Store it securely in a safe place, and never email it. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

PHP Development Environment

Aside from a baseline understanding of object-oriented programming in PHP, there are a few minimum requirements to start using the AWS SDK for PHP. All the following requirements are common in PHP environments, and most extensions come installed with PHP 5.2 by default.

The minimum requirements are:

- PHP 5.2 or later. We recommend either 5.2.14 or the latest 5.3.x release.
- Support for the following enabled in your PHP environment.
  - SimpleXML extension for PHP
  - JavaScript Object Notation (JSON)
  - Perl Compatible Regular Expressions (PCRE)
  - The Standard PHP Library (SPL)
  - Client URL Library (cURL)

The cURL extension should be compiled with the OpenSSL libraries for HTTPS support.

The ability to read from, and write to, the file system via file_get_contents and file_put_contents.

If you're not sure if your PHP installation supports these requirements, you can run the AWS SDK for PHP Compatibility Test.

Alternatively, you can check by either running **phpinfo()** in a browser or by running **`php -i`** from the command line.)

To install your PHP environment, choose one of the installations listed below that corresponds to the operating system that you are using. If you are running PHP on an Amazon EC2 instance, you should choose an installation based on the operating system that the Amazon EC2 instance is running. Each operating system shown below lists multiple possible installations. For example, for Linux, you could install any one of Apache Friends XAMPP, BitName LAMPStack, or BitNami LAPPStack. Click through the links for more information and to determine the most appropriate installation for your needs.

Mac OS X
: MacPorts, MAMP, Apache Friends XAMPP (MacOS), BitNami MAMPStack, BitNami MAPPStack

Linux
: Depending on your distribution, you can likely install everything you need via yum or apt-get; Apache Friends XAMPP, BitNami LAMPStack, bitnami lappstack

windows
: wamp, Apache Friends XAMPP (Windows), BitNami WAMPStack, BitNami WAPPStack

To use Amazon CloudFront, you must have the OpenSSL PHP extension (which is different from any low-level OpenSSL libraries installed on your system) to sign private CloudFront URLs.

To use response caching with the following tools, you must install the associated extensions:

- Alternative PHP Cache (APC)—APC PHP extension
- XCache—XCache PHP extension
- Memcache—Either the older Memcache PHP extension, or the newer Memcached PHP extension
- SQLite—If the PHP Data Objects (PDO) extension support isn't already enabled, install the SQLite3 PHP extension and the PDO-SQLite driver

Install the AWS SDK for PHP

There are a number of ways that you can install the AWS SDK for PHP. Each option is described below.

- Download the install file from Amazon Web Services website http://aws.amazon.com/sdkforphp
- Install from GitHub.

```
git clone git://github.com/amazonwebservices/aws-sdk-for-php.git AWSSDKforPHP
cd ./AWSSDKforPHP
```

For more information on Git, explore Git for the lazy, or Pro Git.

- Install from Subversion

```
svn co http://svn.github.com/amazonwebservices/aws-sdk-for-php.git AWSSDKforPHP
cd ./AWSSDKforPHP
```

- Install from PEAR

```
sudo pear channel-discover pear.amazonwebservices.com
sudo pear install aws/sdk
```

Configure PHP Environment with Your AWS Security Credentials

There are two different ways to configure your credentials: one approach is geared for simplicity, whereas the other is geared for flexibility. The simple approach is preferred for all the installation approaches described above—except for PEAR. For PEAR installations, the flexible approach is preferred. You can find a copy of the config-sample.inc.php file referenced below on GitHub.

To configure your AWS credentials for a non-PEAR installation.

Be sure to back up your **`config.inc.php`** file when upgrading to avoid having to re-enter your credentials.

1. Navigate to the **`config-sample.inc.php`** file in the SDK directory, and rename it to **`config.inc.php`**.
2. Open **`config.inc.php`** for editing. There are instructions in this file for each configuration value.
3. When the SDK is loaded, it will look for **`config.inc.php`** in the same directory as **`sdk.class.php`**. You are now ready to run PHP code that accesses AWS.

To configure your AWS credentials for a PEAR installation.

1. Create a new configuration file at **`~/.aws/sdk/config.inc.php.`**
2. Either download and copy the contents of a **`config-sample.inc.php`** file, or grab it directly from the above GitHub link.
3. Open **`config.inc.php`** for editing. There are instructions in this file for what each configuration value.
4. Enter

```
echo getenv('HOME');
```

to verify that PHP can access the **`HOME`** environment variable. If this command doesn't return the correct file path to your user directory, configure it with:

```
putenv('HOME=<your-user-path>');.
```

AWS Region Selection

AWS Regions allow you to access AWS services that reside physically in a specific geographic region. This is useful both for redundancy and to keep your data and applications running close to where you and your users will access them. To select a particular region, configure the AWS client object with an endpoint that corresponds to that region.

For example, to configure an Amazon EC2 client to connect to the Asia Pacific (Singapore) region, use the following code:

```
$ec2 = new AmazonEC2();
$ec2->set_hostname('ec2.ap-southeast-1.amazonaws.com');
$response = $ec2->describe_instances();
```

Be aware that regions are logically isolated from each other, so for example, you won't be able to access US East resources when communicating with the EU West endpoint. If you're code accesses multiple AWS regions, we recommend that you instantiate a specific client for each region.

Run the Amazon S3 Sample for PHP

The AWS SDK for PHP contains sample code that demonstrates how to use the AWS SDK for PHP with Amazon S3. The sample creates an Amazon S3 bucket, uploads a number of files from a **`test-files/`** directory in to the bucket, then displays a URL for each file.

Note Before you run the sample you must configure the SDK with your AWS security credentials as described above.

To run the Amazon S3 Sample, run the **`cli-s3_get_urls_for_uploads.php`** file located in the **`samples`** folder of the SDK installation directory.

Where Do I Go From Here?

From here, you can check out the tutorials included in this developer guide.

- **Tutorial: Player Manager**

  The first tutorial explains how to use Amazon S3 and Amazon SimpleDB to store information about a group of people—such as players at a tennis club.
- **Tutorial: Amazon EC2 Spot Instances (p. 22)**

  The tutorial explains how to set up a requests for Amazon EC2 Spot Instances, how to determine when they have completed, and how to clean up afterwards.
- **SDK Reference Documentation**

  The SDK reference documentation includes the ability to browse and search across all code included with the SDK. It provides thorough documentation, usage examples, and even the ability to browse method source. You can find it at http://docs.amazonwebservices.com/AWSSDKforPHP/latest.

Additional Resources

The Further Resources has pointers to other resources to assist you in programming AWS.

Using IAM Roles for EC2 Instances with the SDK for PHP

[For in-depth information about IAM roles for EC2 instances, go to the IAM User Guide.]

Securely managing authentication credentials is one of the challenges developers face when writing software that accesses Amazon Web Services (AWS). All requests to AWS must be cryptographically signed using credentials issued by AWS. For software that runs on Amazon Elastic Compute Cloud (EC2) instances, developers need to store these credentials in a way that keeps them secure but also accessible to the software which needs them in order to make requests.

IAM roles for EC2 instances provides an effective way to manage credentials for AWS software running on EC2 instances. This section will describe IAM roles for EC2 instances and show how it works with a sample PHP program. But first, let's examine some common strategies for managing credentials and the issues that arise when using them.

One strategy is to first launch an Amazon EC2 instance and then securely transfer the credentials to the instance using a utility such as SCP (secure copy). However, this strategy doesn't scale well to large numbers of instances. It also doesn't work well for instances that are created by AWS on behalf of the customer, such as Spot Instances or instances in Auto Scaling groups.

Another strategy is to embed the credentials as literal strings in the software itself. However, this means that anyone who comes into possession of the software can scan through the code and retrieve the credentials.

Yet another strategy is to create a custom AMI (Amazon Machine Image) with the credentials, perhaps stored in a file on the AMI. However, with this approach anyone with access to the AMI automatically has access to the credentials—which again creates an unnecessary security risk.

All of the above strategies also make it cumbersome to rotate (update) the credentials. The new credentials either have to be re-copied to the EC2 instance or compiled into a new build of the software or incorporated into the creation of a new AMI.

Use IAM Roles for EC2 Instances to Manage Your Credentials

IAM roles for EC2 instances provides a solution. With IAM roles, a developer can develop software and deploy it to an EC2 instance without having to manage the credentials the software is using.

You use the IAM console to create the IAM role and configure it with all the permissions that the software requires. Permissions for IAM roles are specified in a way that is similar to permissions for IAM users. For more information about specifying permissions, go to the IAM Users Guide.

Amazon EC2 instances support the concept of an instance profile which is a logical container for the IAM role. At the time that you launch an EC2 instance, you can associate the instance with an instance profile, which in turn corresponds to the IAM role. Any software that runs on the EC2 instance is able to access AWS using the permissions associated with the IAM role.

If you are using the AWS Management Console you don't need to worry about instance profiles. The IAM console creates one for you in the background whenever you create an IAM role.

To use the permissions associated with the IAM role, the software constructs a client object for an AWS service, say Amazon Simple Storage Service (Amazon S3), without providing credentials to the constructor. When the constructor executes it first searches for credentials in a file called `config.inc.php`. If that file is not found, or it contains no credentials, the client constructor retrieves temporary credentials that have the same permissions as those associated with the IAM role. The temporary credentials are retrieved from the Instance Meta Data Service (IMDS). The credentials are cached by the SDK on behalf of the application software and used to make calls to AWS from the client object. Although the credentials are temporary and eventually expire, the SDK client periodically refreshes them so that they continue to enable access. This periodic refresh is completely transparent to the application software.

If the client is not provided credentials through the constructor or the `config.inc.php` file, and the client is not able to obtain credentials from the IMDS, then the constructor throws an CFCredentials_Exception.

Note that IAM roles are not currently supported for EC2 instances running under AWS Elastic Beanstalk.

Walkthrough: Using IAM Roles to Retrieve an Amazon S3 Object from an EC2 Instance

In this walkthrough, we consider a program that retrieves an object from Amazon S3 using regular account credentials. We then modify the program so that it uses IAM Roles for EC2 Instances.

Sample Program with Credentials

Here is our starting program, which retrieves an object from an Amazon S3 bucket. The code as shown below explicitly specifies credentials in the call to the Amazon S3 client constructor.

```
<?php

require_once 'sdk.class.php';

// Set timezone to your local timezone (if it is not set in your php.ini)
```

```
date_default_timezone_set('America/Los_Angeles');

$bucket   = 'text-content';
$keyname  = 'text-object.txt';

// Instantiate the class.
$s3 = new AmazonS3(array(
    'key'    => 'AKIAIOSFODNN7EXAMPLE',
    'secret' => 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'
));

// Get object
$response = $s3->get_object($bucket, $keyname);

// Success?
if($response->isOK())
{
 header('Content-Type: ' . $response->header['content-type']);
 echo $response->body;
}
```

You can test this program, by filling in your own credentials in the following lines located near the top of the file.

```
'key' => 'AKIAIOSFODNN7EXAMPLE',
'secret' => 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'
```

You should also specify the name of an Amazon S3 bucket and text object associated with your account.

```
$bucket   = 'text-content';
$keyname  = 'text-object.txt';
```

For instructions on how to create an Amazon S3 bucket and upload an object, go to the Amazon Simple Storage Service Getting Started Guide.

After you have configured the program for your environment (credentials and target object to retrieve), you can run it. Here is a sample commandline to run the program, assuming that the program file is named, get-object.php

```
php get-object.php
```

Update the Sample Program to Use IAM Roles

Our next stage is to update this program to run from an EC2 instance using IAM roles. Here are the high-level steps.

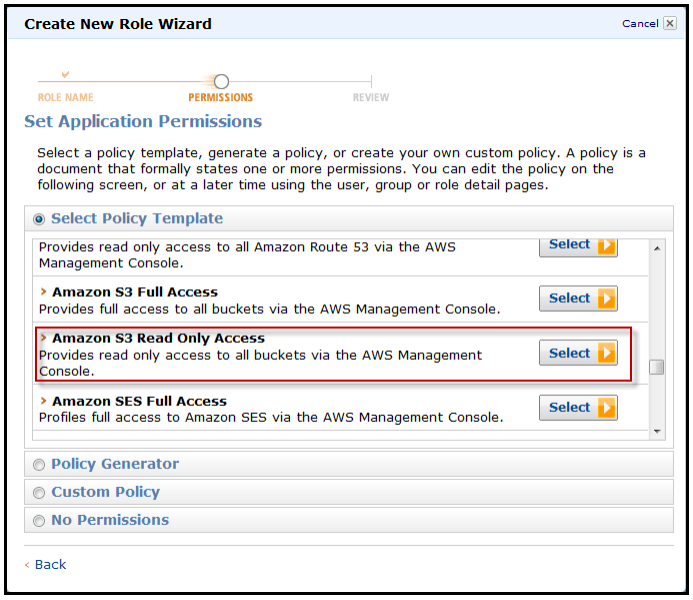
1. **Create an IAM Role using the IAM Management Console**
2. **Launch an EC2 Instance with the Role**
3. **Edit the Source File to Remove the Credentials**
4. **Transfer the Compiled Program to the EC2 Instance**

5. **Run the Program**

Create the Role

The first step is to create a role in IAM that has the appropriate permissions. To create the role, follow the procedure **Creating a Role** in the IAM User Guide. When you create the role, specify that it should have read access to Amazon S3.

The IAM console provides ready-made policy templates for specific AWS services. When you create the IAM role, specify the **Amazon S3 Read Only Access** policy template. The following screen shot from the IAM role creation wizard, shows this policy template.



Policies can also be represented in JSON format. The following JSON block describes the policy for Amazon S3 Ready Only Access.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

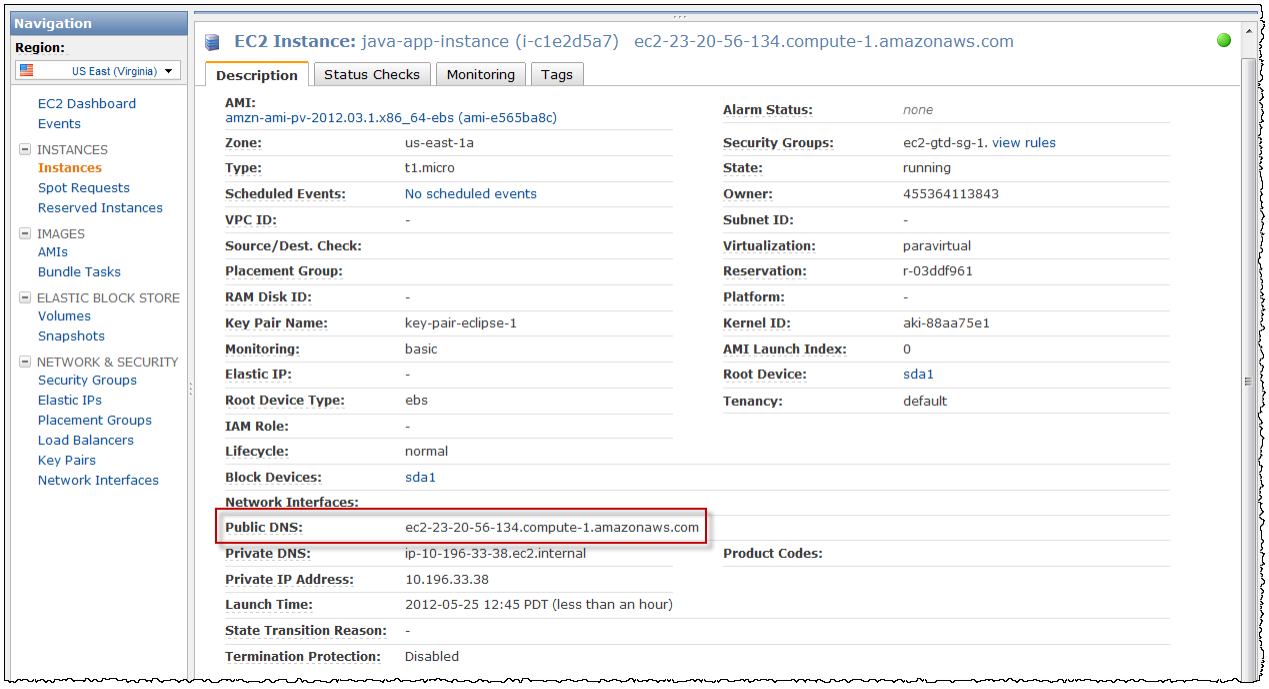
Note down the name of the role that you create so that you can specify it when you create your EC2 Instance in the next step.

Launch an EC2 Instance with the Corresponding Instance Profile

To create an EC2 instance, follow the procedure **Running an Instance** in the Amazon EC2 User Guide. We recommend that you specify a recent **Amazon Linux AMI** for your EC2 instance. When you create the EC2 instance, specify the IAM role that you created previously in the IAM console.

When you create your EC2 instance, you will also need to specify a keypair and a security group. Specify a keypair for which you have the private key (PEM file) stored on your local computer. Specify a security group that will enable you to connect to your EC2 instance using SSH (port 22). Information about keypairs and security groups is provided in the Amazon EC2 User Guide referenced above.

After you create the EC2 instance, go to the EC2 Instances area of the AWS Management Console and view the instance. After the instance transitions to a state of **Running**, record its public DNS name. You will use this DNS name later to connect to the instance using SSH and SCP.



Edit the Source File to Remove the Credentials

Edit the source for the program so that it does **not** specify any credentials in the call that creates the Amazon S3 client.

```
<?php

require_once 'sdk.class.php';

// Set timezone to your local timezone (if it is not set in your php.ini)
date_default_timezone_set('America/Los_Angeles');

$bucket   = 'text-content';
$keyname  = 'text-object.txt';
```

```
// Instantiate the class.
$s3 = new AmazonS3(array(
  'default_cache_config' => '/tmp/secure-dir'
));

// Get object
$response = $s3->get_object($bucket, $keyname);

// Success?
if($response->isOK())
{
  header('Content-Type: ' . $response->header['content-type']);
  echo $response->body;
}
```

In this new version of the program, the constructor for the Amazon S3 client no longer takes any credentials. Instead it takes the the `default_cache_config` option. This option is required when using IAM role credentials. The PHP process typically runs only briefly, such as just for the time that a web page is being rendered. The `default_cache_config` option tells the SDK how to cache the IAM role credentials so that they do not have to be retrieved every time PHP runs.

In the example, the value provided to `default_cache_config` is the location where the credentials will be cached, in this case, a directory called `secure-dir` in `/tmp`. The directory specified should be configured with appropriate permissions so that the credentials are secure. Note that the IAM role credentials are temporary and expire after a period of time which provides additional security. In practice, we recommend that, instead of a directory path, you specify either `apc` (for Alternative PHP Cache) or `xcache`, wich are two supported PHP accelerators. For more information about `default_cache_config`, go to the PHP reference page for the Amazon S3 constructor. All AWS PHP client constructors support this option.

```
$s3 = new AmazonS3(
  array (
    'default_cache_config' => '/tmp'
  )
```

You might actually run the program on your local computer to verify that it does **not** work without credentials; you will get an Amazon Service Exception.

Transfer the Compiled Program to Your EC2 Instance

Transfer the program to your EC2 instance using SCP. Use the public DNS name for the instance that you obtained from the EC2 console earlier. The command will look similar to the following.

```
scp -i key-pair-eclipse-1.pem get-object.php  \
        ec2-user@ec2-23-20-56-134.compute-1.amazonaws.com:get-object.php
```

Note that if you launched an AMI other than the Amazon Linux AMI recommended earlier, you may need to use "root" instead of "ec2-user" when connecting to the instance using SSH or SCP.

Run the Program

To run the program, use SSH to connect to your EC2 instance.

```
ssh -i key-pair-eclipse-1.pem ec2-user@ec2-23-20-56-134.compute-1.amazonaws.com
```

The Amazon Linux AMI does not have PHP installed by default. You can install PHP and the AWS SDK for PHP using any of the methods described in the Getting Started section under PHP Development Environment (p. 5).

Run the program.

```
php get-object.php
```

The program should retrieve the text object from Amazon S3 and display its contents to your console.

Tutorial: Player Manager Application

This tutorial demonstrates using Amazon SimpleDB and Amazon S3 together to build a small part of an AWS application. SimpleDB is designed to store textual information (e.g., name and email address) and S3 is designed to store blob data (e.g., photos). In this example, SimpleDB is used to store registered users (in this case, players) and Amazon S3 is used to store a photo of each player.

Requirements

- The AWS SDK for PHP
- An IDE or text editor. Here are some free suggestions:
  - Notepad++ (Windows)
  - Geany or GEdit (Linux)
  - TextWrangler (Mac)

Getting Started

The file that contains your application should have a `.php` extension, for example, `aws_tutorial.php`. We recommend using UTF-8 encoding and Unix-style end-of-line mode for the file. Here are the first few lines of the application.

```
<?php
// Enable full-blown error reporting.
error_reporting(-1);

// Set HTML headers
header('Content-type: text/html; charset=utf-8');

// Include the SDK
require_once 'AWSSDKforPHP/sdk.class.php';
```

Notice the path in the `require_once` call. The path should point to your install location of the AWS SDK for PHP. Also, the `config.inc.php` file in that folder needs to contain your AWS credentials, as explained in the Getting Started section.

The end of the file shows the method calls that do the work of the application: setting up the storage and registering the players.

```
$pm = new PlayerManager();
$pm->setup_storage();

$pm->register_player('arthur@example.com', 'Arthur', 'sample.png');
$pm->register_player('maria@example.com', 'Maria', 'sample.png');
```

Let's take a look at the code that implements this functionality.

The PlayerManager class

The `PlayerManager` class has two features. The first is that it initializes the storage we'll use in Amazon SimpleDB and Amazon S3. Secondly, it registers individual users.

This app uses a single SimpleDB domain. Each player is represented by a SimpleDB item and each item has a name and two attributes. The item name acts like a key and must be unique for each player.

We use the player's email address for the name and we store two attributes with each player. One attribute is the player's name, the other is the value of an Amazon S3 key which serves as a pointer into an Amazon S3 bucket where the player's picture is stored.

SimpleDB would contain a domain which in the sample code I've defined as "tutorial-players". The storage would look like this after two players are registered:

SimpleDB domain: tutorial-players

Item Name	PlayerName	PictureS3Key
arthur@example.com	Arthur	tutorialphotos/arthur_at_example_dot_com.png
maria@example.com	Maria	tutorialphotos/maria_at_example_dot_com.png

Amazon S3 bucket: tutorial-playerfiles-php-abcd123

Key	Contents
tutorial-photos/arthur_at_example_dot_com.png	Player's picture
tutorial-photos/maria_at_example_dot_com.png	Player's picture

The code which supports this data storage model starts with a class called `PlayerManager`.

```
class PlayerManager
{
  private $sdb_players;
  private $s3_player_files;
```

```
  private $s3_player_photos;

  public function __construct()
  {
    $this->sdb_players = 'tutorial-players';
    $this->s3_player_files = 'tutorial-playerfiles-php-abcd123';
    $this->s3_player_photos = 'photos/';
  }
}
```

This code starts our `PlayerManager` class. For the implementation of this class, we have some decisions to make. There are three values we need to configure which will define the names of the SimpleDB domain, the Amazon S3 bucket, and the prefix to the player photo files.

Decisions, decisions...

The SimpleDB domain name is straightforward. It can be any unique name within your account. The Amazon S3 bucket name has a requirement in that it must be unique across all of Amazon S3, and it is good practice to follow the rules of valid Internet domain names.

The implementation here appends some text that might not be unique and you will likely need to change it. Change the text I've provided (`tutorial-playerfiles-php-abcd123`) with a string likely to be unique across all of Amazon S3. If you own a registered domain name you might choose to use that in your S3 bucket names instead. Alternatively, your AWS Access Key ID also works well.

Lastly, the Amazon S3 folder name can be anything you choose. It doesn't need to end with a trailing slash but if it does, the AWS Console will display the contents of the bucket like folder of files, which can help organization.

Setting up storage

With these decisions made the `PlayerManager` class is shown below. Not shown yet are the implementations of the `setup_storage` and `register_player`.

```
class PlayerManager
{
  private $sdb_players;
  private $s3_player_files;
  private $s3_player_photos;

  private $s3;
  private $sdb;

  public function __construct()
  {
    $this->sdb_players = 'tutorial-players';
    $this->s3_player_files = 'tutorial-playerfiles-php-abcd123';
    $this->s3_player_photos = 'photos/';

    // Create clients for the services we're using.
    $this->s3 = new AmazonS3();
    $this->sdb = new AmazonSDB();
```

```
  }

  public function setup_storage()
  {
  }

  public function register_player($email, $player_name, $photo)
  {
  }
}
```

The `PlayerManager` class now has two new private member variables and a constructor that initializes them. The `PlayerManager::__construct()` method creates instances of the two AWS clients our program uses; one for SimpleDB and one for Amazon S3. These client instances only need to be created once. In this example, they're created using the AWS credentials stored in the `config.inc.php` file.

Now that we have AWS client instances we can put them to use. The implementation for `setup_storage()` is shown below.

```
public function setup_storage()
{
  // Create Players domain in SimpleDB.
  $r = $this->sdb->create_domain($this->sdb_players);

  // Debug information
  if (!$r->isOK())
  {
    print_r($r);
    return;
  }

  // Create the S3 bucket for player photos
  $r = $this->s3->create_bucket($this->s3_ player_files, AmazonS3::REGION_US_E1);


  // Debug information
  if (!$r->isOK())
  {
    print_r($r);
    return;
  }
}
```

Let's take a look at how the `setup_storage()` method works. The first thing it does is create the SimpleDB domain for players by calling the `create_domain()` method of the SimpleDB client.

```
$r = $this->sdb->create_domain($this->sdb_players);
```

Next, the S3 bucket is created using the `create_bucket()` function of the S3 client.

```
$r = $this->s3->create_bucket($this->s3_player_files, AmazonS3::REGION_US_E1);
```

At this point you can run the program and view the new Amazon S3 bucket in the AWS Console on the web.

Idempotence

You might wonder what happens if you run the `setup_storage()` method many times. Go ahead and try it. You should find that nothing bad happens. It simply runs with no error. This is because the `create_domain()` and `create_bucket()` actions are designed to work the same way whether the domain or bucket existed before or not. When a function or method has the same effect whether it is called once or multiple times (with the same parameters), we say that it is idempotent.

You might also wonder whether any data in the domain or bucket would still be there. In order to see that the data is not deleted, we'll need to look at the code for the `register_player()` method which we'll do next.

Registering players

Here is the code for the `register_player()` method.

```
public function register_player($email, $player_name, $photo)
{
  // Build S3 key for player photo from the photo folder name and player email

  $picture_key = str_replace(
    array('.', '@'),
    array('_dot_', '_at_'),
    $this->s3_player_photos . $email
  ). '.png';

  // Store the player's picture in S3
  $this->s3->create_object($this->s3_player_files, $photo, array(
    'fileUpload' => 'sample.png'
  ));

  // Send the player information to SimpleDB
  $this->sdb->put_attributes($this->sdb_players, $email, array(
    'PlayerName' => $player_name,
    'PictureS3Key' => $photo
  ));
}
```

The `register_player()` method does the work of writing the player's information to storage. It takes the player's email address, name, and the full path to picture file. The `register_player()` method uses the `create_object()` Amazon S3 action and the `put_attributes()` action of SimpleDB.

This method stores the player's picture in Amazon S3 first. To do this it needs a key and the picture file. The picture file is passed in to the method call but the key name is assembled using the configured prefix followed by a transformation of the player's email address.

```
$picture_key = str_replace(
  array('.', '@'),
  array('_dot_', '_at_'),
  $this->s3_player_photos . $email
) . '.png';
```

Now that we have a key for the Amazon S3 object we can upload the photo using the `create_object()` method of the Amazon S3 client. The AWS SDK for PHP does some helpful work for us by letting us

provide a file path to the player's picture and then internally opening the file, reading the bytes, and assembling them into a proper request to Amazon S3.

```
$this->s3->create_object($this->s3_player_files, $photo, array(
  'fileUpload' => 'sample.png'
));
```

That creates the Amazon S3 object for the player's picture. If you run the program at this point you can view the file exploring the bucket in the AWS Console on the web.

Registration is complete when an item is added to the SimpleDB domain. The item and its attributes are created in a single SimpleDB action (`put_attributes()`). Here the domain name, item name, and an array containing two pairs of values are used to define the player item in SimpleDB.

```
$this->sdb->put_attributes($this->sdb_players, $email, array(
  'PlayerName' => $player_name,
  'PictureS3Key' => $photo
));
```

Running the app

That completes the implementation of the `PlayerManager` class. The program will now run. The sample code after the setup section demonstrates using the `PlayerManager`.

```
$pm = new PlayerManager();
$pm->setup_storage();

$pm->register_player('arthur@example.com', 'Arthur', 'sample.png');
$pm->register_player('maria@example.com', 'Maria', 'sample.png');
```

First the `PlayerManager` instance is created, which initializes the SimpleDB and Amazon S3 client objects. The storage is then configured by creating the SimpleDB domain and creating an Amazon S3 bucket. Finally, we register a couple players.

Tutorial: Amazon EC2 Spot Instances

Overview

Spot Instances allow you to bid on unused Amazon Elastic Compute Cloud (Amazon EC2) capacity and run the acquired instances for as long as your bid exceeds the current *Spot Price*. Amazon EC2 changes the Spot Price periodically based on supply and demand, and customers whose bids meet or exceed it gain access to the available Spot Instances. Like On-Demand Instances and Reserved Instances, Spot Instances provide another option for obtaining more compute capacity.

Spot Instances can significantly lower your Amazon EC2 costs for applications such as batch processing, scientific research, image processing, video encoding, data and web crawling, financial analysis, and testing. Additionally, Spot Instances are an excellent option when you need large amounts of computing capacity but the need for that capacity is not urgent.

To use Spot Instances, place a Spot Instance request specifying the maximum price you are willing to pay per instance hour; this is your bid. If your bid exceeds the current Spot Price, your request is fulfilled and your instances will run until either you choose to terminate them or the Spot Price increases above your bid (whichever is sooner). You can terminate a Spot Instance programmatically as shown in this tutorial or by using the AWS Console.

It's important to note two points:

(1) You will often pay less per hour than your bid. Amazon EC2 adjusts the Spot Price periodically as requests come in and available supply changes. Everyone pays the same Spot Price for that period regardless of whether their bid was higher. Therefore, you might pay less than your bid, but you will never pay more than your bid.

(2) If you're running Spot Instances and your bid no longer meets or exceeds the current Spot Price, your instances will be terminated. This means that you will want to make sure that your workloads and applications are flexible enough to take advantage of this opportunistic—but potentially transient—capacity.

Spot Instances perform exactly like other Amazon EC2 instances while running, and like other Amazon EC2 instances, Spot Instances can be terminated when you no longer need them. If you terminate your instance, you pay for any partial hour used (as you would for On-Demand or Reserved Instances). However, if your instance is terminated by Amazon EC2 because the Spot Price goes above your bid, you will not be charged for any partial hour of usage.

This tutorial provides an overview of how to use the PHP programming environment to do the following.

- Submit a Spot Request
- Determine when the Spot Request becomes fulfilled
- Cancel the Spot Request
- Terminate associated instances

Prerequisites

This tutorial assumes that you have signed up for AWS, set up your PHP development environment, and installed the AWS SDK for PHP.

Step 1: Setting Up Your Credentials

To begin using this code sample, you need to populate the **`config.inc.php`** file with your credentials, that is, your Access Key ID and your Secret Key. You can find instructions for adding your credentials to **`config.inc.php`** in the **Getting Started** section of this guide under Configure PHP Environment with Your AWS Security Credentials (p. 6).

To view your AWS access credentials

1. Go to the Amazon Web Services website at http://aws.amazon.com.
2. Click **My Account/Console**, and then click **Security Credentials**.
3. Under **Your Account**, click **Security Credentials**.
4. In the spaces provided, type your user name and password, and then click **Sign in using our secure server**.
5. Under **Access Credentials**, on the **Access Keys** tab, your access key ID is displayed. To view your secret key, under **Secret Access Key**, click **Show**.

Step 2: Setting Up a Security Group

A *security group* acts as a firewall that controls the traffic allowed in and out of a group of instances. By default, an instance is started without any security group, which means that all incoming IP traffic, on any TCP port will be denied. So, before submitting your Spot Request, you will set up a security group that allows the necessary network traffic. For the purposes of this tutorial, we will create a new security group called "GettingStarted" that allows connection using SSH from the IP address of the local computer, that is, the computer where you are running the application.

To set up a new security group, you need to include or run the following code sample that sets up the security group programmatically. You only need to run this code once to create the new security group. However, the code is designed so that it is safe to run even if the security group already exists. In this case, the code catches and ignores the "InvalidGroup.Duplicate" exception.

In the code below, we first use instantiate an **AmazonEC2** client object. We then call the **create_security_group** action with the name, "GettingStarted" and a description for the security group.

```
1
  // Create the AmazonEC2 object so we can call various APIs.
```

```
    $ec2 = new AmazonEC2();

  5 // Create a new security group.
    $response = $ec2->create_security_group ( 'GettingStartedGroup', 'Getting
Started Security Group');
    if (!$response->isOK())
    {
     if (((string) $response->body->Errors->Error->Code) === 'InvalidGroup.Du
plicate')
 10  {
      // This means that the group is already created, so ignore.
      echo 'create_security_group returned an acceptable error: ' . $response-
>body->Errors->Error->Message . PHP_EOL;
     } else {
      print_r($response);
 15   exit();
     }
    }
```

To enable access to the group, we create an associative array ("$ingress_opt") to store the access configuration. We set the IP address to the CIDR representation of the IP address of the local computer. The "/32" suffix on the IP address indicates that the security group should accept traffic *only* from the local computer. We also configure the **ipPermission** object with the TCP protocol and port 22 (SSH). You will need to fill in the IP address of the local computer. If your connection to the Internet is mediated by a firewall or some other type of proxy, you will need to determine the external IP address that is used by the proxy. One technique is to query a search engine such as Google or Bing with the string: "what is my IP address".

```
  1
    // TODO - Change the code below to use your external ip address.
    $ip_source = 'XXX.XXX.XXX.XX/32';

  5 // Open up port 22 for TCP traffic to the associated IP
    // from above (e.g. ssh traffic).
    $ingress_opt = array(
     'GroupName' => 'GettingStartedGroup',
     'IpPermissions' => array(
 10   array(
       'IpProtocol' => 'tcp',
       'FromPort' => '22',
       'ToPort' => '22',
       'IpRanges' => array(
 15     array('CidrIp' => $ip_source),
       )
        )
     )
    );
 20
```

The final step is to call **authorize_security_group_ingress** with the name of our security group and the configuration array.

```
  1
    // Authorize the ports to be used.
```

```
    $response = $ec2->authorize_security_group_ingress($ingress_opt);
    if (!$response->isOK())
  5 {
     if (((string) $response->body->Errors->Error->Code) === 'InvalidPermis
sion.Duplicate')
     {
      echo 'authorize_security_group_ingress returned an acceptable error: ' .
 $response->body->Errors->Error->Message . PHP_EOL;
     } else {
 10   print_r($response);
      exit();
     }
    }
```

You can also create the security group using the AWS Toolkit for Eclipse. Go to the toolkit documentation for more information.

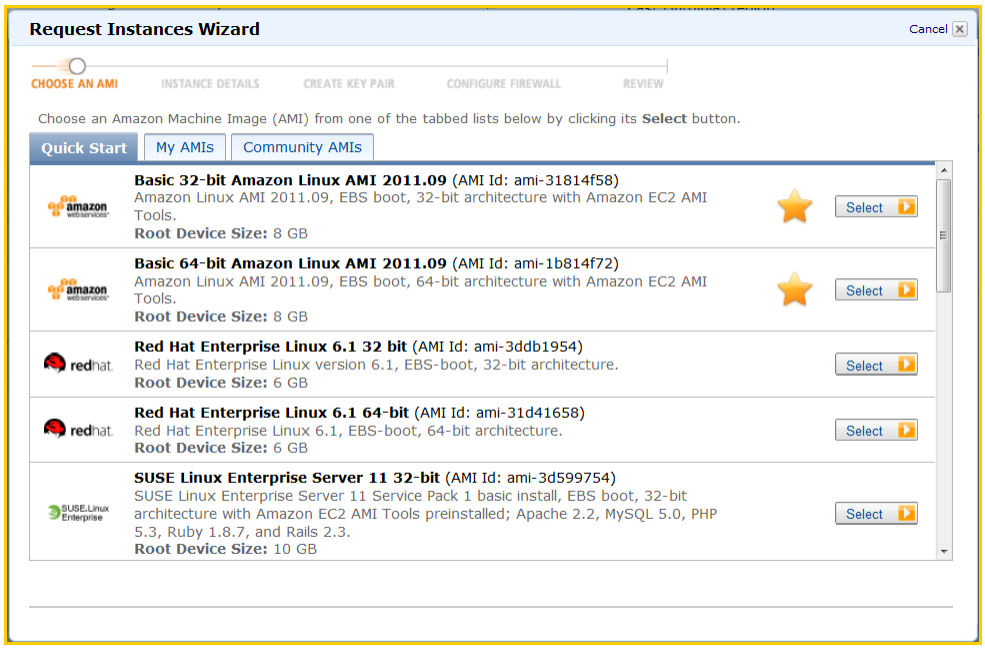
Step 3: Submitting Your Spot Request

To submit a Spot request, you first need to determine the instance type, Amazon Machine Image (AMI), and maximum bid price you want to use. You must also include the security group we configured above, so that you can log into the instance if desired.

There are several instance types to choose from; go to Amazon EC2 Instance Types for a complete list. For this tutorial, we will use t1.micro, the least expensive instance type available. Next, we will determine the type of AMI we would like to use. We'll use ami-31814f58, the most up-to-date Linux AMI available when we wrote this tutorial. The latest AMI may change over time, but you can always determine the latest version AMI by:

1. Logging into the AWS Management Console, clicking the EC2 tab, and, from the EC2 Console Dashboard, attempting to launch an instance.

2. In the window that displays AMIs, just use the AMI ID as shown below. Alternatively, you can use the **DescribeImages** API, but leveraging that command is outside the scope of this tutorial.



There are many ways to approach bidding for Spot instances; to get a broad overview of the various approaches you should view the Bidding for Spot Instances video. However, to get started, we'll describe three common strategies: bid to ensure cost is less than on-demand pricing; bid based on the value of the resulting computation; bid so as to acquire computing capacity as quickly as possible.

- **Reduce Cost below On-Demand** You have a batch processing job that will take a number of hours or days to run. However, you are flexible with respect to when it starts and when it completes. You want to see if you can complete it for less cost than with On-Demand Instances. You examine the Spot Price history for instance types using either the AWS Management Console or the Amazon EC2 API. For more information, go to Viewing Spot Price History. After you've analyzed the price history for your desired instance type in a given Availability Zone, you have two alternative approaches for your bid:
  - You could bid at the upper end of the range of Spot Prices (which are still below the On-Demand price), anticipating that your one-time Spot request would most likely be fulfilled and run for enough consecutive compute time to complete the job.
  - Or, you could bid at the lower end of the price range, and plan to combine many instances launched over time through a persistent request. The instances would run long enough--in aggregate--to complete the job at an even lower total cost. (We will explain how to automate this task later in this tutorial.)
- **Pay No More than the Value of the Result** You have a data processing job to run. You understand the value of the job's results well enough to know how much they are worth in terms of computing costs. After you've analyzed the Spot Price history for your instance type, you choose a bid price at which the cost of the computing time is no more than the value of the job's results. You create a persistent bid and allow it to run intermittently as the Spot Price fluctuates at or below your bid.
- **Acquire Computing Capacity Quickly** You have an unanticipated, short-term need for additional capacity that is not available through On-Demand Instances. After you've analyzed the Spot Price history for your instance type, you bid above the highest historical price to provide a high likelihood that your request will be fulfilled quickly and continue computing until it completes.

After you choose your bid price, you are ready to request a Spot Instance. For the purposes of this tutorial, we will set our bid price equal to the On-Demand price ($0.03) to maximize the chances that the bid will

be fulfilled. You can determine the types of available instances and the On-Demand prices for instances by going to Amazon EC2 Pricing page.

To request a Spot Instance, you simply need to build your request with the parameters we have specfied so far. We start by creating a associative array ("$spot_opt") to hold the configuration for the instance. The request requires the number of instances you want to start (2) and the bid price ($0.03). Additionally, we need to set the **LaunchSpecification** for the request, which includes the instance type, AMI ID, and security group you want to use. Once the request is populated, we call the **request_spot_instances** method on the **$ec2** object. An example of how to request a Spot Instance is shown below.

```
 1
   // Setup the specifications of the launch. This includes the
   // instance type (e.g. t1.micro) and the latest Amazon Linux
   // AMI id available. Note, you should always use the latest
 5 // Amazon Linux AMI id or another of your choosing.
   $spot_opt = array(
    'InstanceCount' => 2
    'LaunchSpecification' => array(
     'ImageId' => 'ami-31814f58',
10   'SecurityGroup' => 'GettingStartedGroup',
     'InstanceType' => 't1.micro'
    )
   );

15 // Request 1 x t1.micro instance with a bid price of $0.03.
   $response = $ec2->request_spot_instances('0.03', $spot_opt);
   if (!$response->isOK())
   {
    print_r($response);
20  exit();
   }
```

There are other options you can use to configure your Spot Requests. To learn more, visit the request_spot_instances method in the PHP SDK.

Running this code will launch a new Spot Instance Request.

Note

You will be charged for any Spot Instances that are actually launched, so make sure that you cancel any requests and terminate any instances you launch to reduce any associated fees. Code for cancelling requests and terminating instances is show below.

Step 4: Determining the State of Your Spot Request

Next, we want to create code to wait until the Spot request reaches the "active" state before proceeding to the last step. To determine the state of our Spot request, we poll the describe_spot_instance_requests method for the state of the Spot request ID we want to monitor.

The request ID created in Step 2 is embedded in the response to our **request_spot_instances** request. The following example code gathers request IDs from the **request_spot_instances** response and stores them in an array called $spot_instance_request_ids.

```
  1
    $spot_instance_request_ids = array();
    for ($i=0; $i < $response->body->spotInstanceRequestSet->item->count();
$i++)
    {
  5  $spot_instance_request_id = (string)$response->body->spotInstanceRequestSet-
>item[$i]->spotInstanceRequestId;
     $spot_instance_request_ids[] = $spot_instance_request_id;
    }
```

To monitor the request IDs, we loop through them calling the **describe_spot_instance_requests** method to determine the state of each request. We loop until none of the requests are in the "open" state. Note that we monitor for a state of not "open", rather a state of, say, "active", because the request can go straight to "closed" if there is a problem with your request arguments. The code example below provides the details of how to accomplish this task.

```
  1
    // Initialize a variable that will track whether there are any
    // requests still in the open state.
    $any_open = false;
  5
   // Initialize an array to hold any instances we activate so we can terminate
 them later.
    $instance_ids = array();

    do {
 10  // Call describe_spot_instance_requests with all of the request ids to
     // monitor (e.g. that we started).
     $describe_opt = array(
      'SpotInstanceRequestId' => $spot_instance_request_ids
     );
 15  $response = $ec2->describe_spot_instance_requests($describe_opt);
     if (!$response->isOK())
     {
      print_r($response);
      exit();
 20  }

     // Reset the any_open variable to false - which assumes there
     // are no requests open unless we find one that is still open.
     $any_open = false;
 25
     // Look through each request and determine if they are all in
     // the active state.
     foreach ($response->body->spotInstanceRequestSet->item as $item)
     {
 30   echo "spotInstanceRequestId = $item->spotInstanceRequestId, state = $item-
>state" . PHP_EOL;

      // If the state is open, it hasn't changed since we attempted
      // to request it. There is the potential for it to transition
      // almost immediately to closed or cancelled so we compare
 35   // against open instead of active.
      if (((string)$item->state) === 'open')
      {
       $any_open = true;
```

```
      break;
 40   }

      if (((string)$item->state) === 'active')
      {
       // Get the instanceId once the spot instance request is active
 45    $instance_id = (string)$item->instanceId;
       echo 'Instance $instanceId is active.' . PHP_EOL;

      // Store the instanceId for any instances we've started so we can termin
ate them later.
       if (!in_array($instanceId, $instanceIds))
 50    {
        $instance_ids[] = (string)$item->instanceId;
       }
      }
     }
 55
     if ($any_open)
     {
      echo 'Requests still in open state, will retry in 60 seconds.' . PHP_EOL;
       sleep(60);
 60  }
    }
    while($any_open);
```

If you just ran the code up to this point, your, your Spot Instance Request would complete—or possibly fail with an error. For the purposes of this tutorial, we'll add some code that cleans up after all of the requests have transitioned out of the open state.

Step 5: Cleaning up Your Spot Requests and Instances

The final step is to clean up our requests and instances. It is important to both cancel any outstanding requests *and* terminate any instances. Just canceling your requests will not terminate your instances, which means that you will continue to pay for them. If you terminate your instances, your Spot requests may be canceled, but there are some scenarios--such as if you use persistent bids--where terminating your instances is not sufficient to stop your request from being re-fulfilled. Therefore, it is a best practice to both cancel any active bids and terminate any running instances.

The following code demonstrates how to cancel your requests.

```
 1
   $response = $ec2->cancel_spot_instance_requests($spot_instance_request_ids);
    if ($response->isOK())
    {
 5   echo 'Canceled spot instance requests.' . PHP_EOL;
    } else {
     print_r($response);
     exit();
    }
 10
```

To terminate any outstanding instances, we use the instance_ids array, which we populated with the instance IDs of those instances that transitioned to the active state. We terminate these instances by paste this array to the **terminate_instances** method.

```
1
  if (count($instance_ids) > 0)
  {
   $response = $ec2->terminate_instances($instanceIds);
5  if ($response->isOK())
   {
    echo 'Terminated requested spot instances.' . PHP_EOL;
   } else {
    print_r($response);
10  exit();
   }
  }
```

Conclusion

Congratulations! You have just completed the getting started tutorial for developing Spot Instance software with the AWS SDK for PHP.

Start an Amazon EC2 Instance

This section demonstrates how to use the AWS SDK for PHP to start an Amazon Elastic Compute Cloud (Amazon EC2) instance.

Topics

Create an Amazon EC2 Client

You will need an Amazon EC2 client in order to create security groups and key pairs, and start Amazon EC2 instances. Before configuring your client, you must create a PHP file named `config.inc.php` to store your AWS Access Key and your Secret Key. This PHP file must be placed in the same parent directory as your PHP SDK.

The file looks like this:

```
<?php if (!class_exists('CFRuntime')) die('No direct access allowed.');
  CFCredentials::set(array(
      '@default' => array(
          'key' => 'MyAccessKey',
          'secret' => 'MySecretKey',
          'default_cache_config' => 'C:\wamp\www',
          'certificate_authority' => true
      )
  ));
```

You must provide your AWS security credentials, the path to your local PHP cache, and a `certificate_authority` value of `true` or `false`. Other configuration values can be optionally defined.

As an example to follow, use the `config-sample.inc.php` file in the same directory as your configuration file.

To learn more about your AWS credentials, including where to find them, go to About AWS Security Credentials.

After you create this file, you are ready to create and initialize your Amazon EC2 client.

To create and initialize an Amazon EC2 client

1. Create and initialize an AmazonEC2 instance, as follows:

```
$ec2 = new AmazonEC2();
```

2. Use the set_region method with a constant to specify the service endpoint you wish to use, as follows:

```
$ec2->set_region(AmazonEC2::REGION_OREGON);
```

   By default, the service endpoint is `ec2.us-east-1.amazonaws.com`. For a list of Amazon EC2 service endpoints, go to Regions and Endpoints.

Before running an Amazon EC2 instance, you will need to create a Amazon EC2 security group, authorize security group ingress, and create a key pair to allow you to log into your instance.

For information about creating a security group, see Create a Security Group (p. 32).

For information about authorizing security group ingress, see Authorize Security Group Ingress (p. 33).

For information about creating a key pair, see Create a Key Pair (p. 34).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 34).

Create a Security Group

An Amazon EC2 *security group* controls traffic through your Amazon EC2 instances, much like a firewall. If you do not create a security group, Amazon EC2 provides a default security group that allows no inbound traffic. For more information about security groups, go to Security Group Concepts.

If you want to allow inbound traffic, create a security group and assign a *rule* to it that allows the ingress that you want. Then associate the new security group with an Amazon EC2 instance. For more information, see Authorize Security Group Ingress (p. 33).

To create a security group, use the create_security_group method. Pass a security group name and description as parameters. The method returns a CFResponse object, as follows:

```
$response = $ec2->create_security_group('my-php-security-group', 'This is a PHP
 security group.');
```

You can create up to 500 security groups per AWS account.

The security group name must be unique within the AWS region in which you initialize your Amazon EC2 client. You must use US-ASCII characters for the security group name and description. If you attempt to create a security group with the same name as an existing security group, an error will occur.

Before starting an Amazon EC2 instance, you next need to authorize security group ingress and create a key pair to allow you to log into your instance.

For information about authorizing security group ingress, see Authorize Security Group Ingress (p. 33).

For information about creating a key pair, see Create a Key Pair (p. 34).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 34).

Authorize Security Group Ingress

By default, a new security group does not allow any inbound traffic to your Amazon EC2 instance. To allow inbound traffic, you must explicitly authorize security group ingress. You can authorize ingress for individual IP addresses, for a range of IP addresses, for a specific protocol, and for TCP/UDP ports.

To authorize security group ingress, use the authorize_security_group_ingress method. Specify the security group name, the IP protocol, the port range (optional), and IP addresses to authorize ingress for in CIDR notation. The method returns a CFResponse object, as follows:

```
$response = $ec2 -> authorize_security_group_ingress(
    array('GroupName' => 'my-php-security-group',
        'IpPermissions' => array(
            array('IpProtocol' => 'tcp',
            'FromPort' => '80',
            'ToPort' => '80',
            'IpRanges' => array(
                array('CidrIp' => '111.111.111.111/32'),
                )
            )
        )
    )
);
```

If you specify the protocol as TCP/UDP, you must provide a source port and destination port. You can authorize ports only if you specify TCP or UDP.

If you call `authorize_security_group_ingress` with IP addresses that have already been authorized, an error will occur.

Whenever you use the `authorize_security_group_ingress` or authorize_security_group_egress methods, a *rule* is added to your security group. You can add up to 100 rules per security group. For more information about security groups, go to Security Group Concepts.

Before starting an Amazon EC2 instance, you need to create a key pair to allow you to log into your instance. For information about creating a key pair, see Create a Key Pair (p. 34).

Create a Key Pair

Public AMI instances have no default password. To log into your Amazon EC2 instance, you must generate an Amazon EC2 key pair. The key pair consists of a public key and a private key, and is not the same as your AWS access credentials. For more information about Amazon EC2 key pairs, go to Getting an SSH Key Pair.

To create a key pair and obtain the private key

1. Use the create_key_pair method and specify the key pair name. The method returns a CFResponse object, as follows:

```
$response = $ec2->create_key_pair('my-php-key-pair');
```

   Key pair names must be unique. If you attempt to create a key pair with the same key name as an existing key pair, an error will occur.
2. Use the response object's `body->keyMaterial` property to obtain the unencrypted PEM-encoded private key, as follows:

```
(string) $private_key = $response->body->keyMaterial;
```

   Calling `create_key_pair` is the only way to obtain the private key programmatically. You can always access your private key through the AWS Management Console.

Before logging onto an Amazon EC2 instance, you must create the instance and ensure that it is running. For information on how to run an Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 34).

For information on how to use your key pair to connect to your Amazon EC2 instance, see Connect to Your Amazon EC2 Instance (p. 35).

Run an Amazon EC2 Instance

Before running an Amazon EC2 instance, ensure that you have created a security group and a key pair for your instance. For information about creating a key pair, see Create a Key Pair (p. 34). For information about creating a security group, see Create a Security Group (p. 32).

To run an Amazon EC2 instance, use the run_instances method. Specify the Amazon Machine Image (AMI), the minimum and maximum number of instances to start, the instance type, the key pair name, and the security group. The method returns a CFResponse object, as follows:

```
$response = $ec2 -> run_instances(
 'ami-38fe7308', 1, 1,
 array(
  'InstanceType' => 'm1.small',
  'KeyName' => 'my-php-key-pair',
  'SecurityGroup' => 'my-php-security-group'));
```

You must specify a public or privately-provided AMI. A large selection of Amazon-provided public AMIs is available for you to use. For a list of public AMIs provided by Amazon, go to Amazon Machine Images. Ensure that the specified image ID exists in the region in which your client was created.

The instance type must match the AMI you want to run. For 64-bit architecture, you cannot specify an instance type of `m1.small`. For more information on instance types, go to Instance Families and Types.

You must specify a minimum number and a maximum number of instances to launch. If the specified number of instances is greater than the number of instances you are authorized to launch, no instances are launched. The specified number of maximum instances must be no greater than the maximum number allowed for your account; by default, the maximum number of instances is 20. If there are fewer instances available than the maximum number specified, the largest possible number of images are launched.

Ensure that the specified key name and security group exists for the region in which your client was created.

After you have created your Amazon EC2 instance, you can log onto the AWS Management Console to check the status of the instance.

Once your Amazon EC2 instance is running, you can remotely connect to it using your key pair. For information about connecting to your instance, see Connect to Your Amazon EC2 Instance (p. 35).

Connect to Your Amazon EC2 Instance

Before connecting to your Amazon EC2 instance, you must ensure that the instance's SSH/RDP port is open to traffic. You must also install an SSH/RDP client on the computer you are accessing your instance from. You will need your Amazon EC2 instance ID and the private key from the key pair you created. For information on how to obtain the private key, see Create a Key Pair (p. 34).

If you did not authorize ingress for the security group that your instance belongs to, you will not be able to connect to your instance. By default, Amazon EC2 instances do not permit inbound traffic. For more information on authorizing security group ingress, see Authorize Security Group Ingress (p. 33).

For information on how to connect to your Amazon EC2 instance, go to Connecting to Instances in the Amazon EC2 User Guide.

Related Resources

The following table lists related resources that you'll find useful when using Amazon EC2 with the AWS SDK for PHP.

Resource	Description
PHP Developer Center	Provides sample code, documentation, tools, and additional resources to help you build applications on Amazon Web Services.
AWS SDK for PHP Documentation	Provides documentation for the AWS SDK for PHP.
Amazon Elastic Compute Cloud (EC2) Documentation	Provides documentation for the Amazon Elastic Compute Cloud (EC2) service.

Further Resources

Home Page for AWS SDK for PHP

For more information about the AWS SDK for PHP, including a complete list of supported AWS products, go to http://aws.amazon.com/sdkforphp.

SDK Reference Documentation

The SDK reference documentation includes the ability to browse and search across all code included with the SDK. It provides thorough documentation, usage examples, and even the ability to browse method source. You can find it at http://docs.amazonwebservices.com/AWSSDKforPHP/latest.

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. There is a forum specifically for **AWS development in PHP** as well as forums for individual services such as **Amazon S3**. AWS engineers monitor the forums and respond to questions, feedback, and issues. You can also subscribe RSS feeds for any of the forums.

Send Email from PHP with Amazon Simple Email Service

The developer guide for Amazon Simple Email Service (SES) shows a code example in which a PHP program connects to a MySQL database and iterates through a table containing email addresses. For each address, the program sends an email message using the SMTP email capability of PHP.

Migrating to the AWS SDK for PHP

How to migrate your application from the CloudFusion SDK to the AWS SDK for PHP.

The SDK reference documentation includes the ability to browse and search across all code included with the SDK. It provides thorough documentation, usage examples, and even the ability to browse method source. You can find it at http://docs.amazonwebservices.com/AWSSDKforPHP/latest.

Videos by Ryan Parman, AWS SDK Team

- AWS SDK for PHP Basics [13:42]
- Fundamentals of Amazon S3 [15:09]
- Fundamentals of Amazon SimpleDB [12:56]
- Fundamentals of Amazon SQS [11:46]
- Fundamentals of Amazon CloudFront [18:21]