AWS SDK for Ruby

Developer Guide

Version v1.0.0

AWS SDK for Ruby: Developer Guide

Table of Contents

What is the AWS SDK for Ruby?

The AWS SDK for Ruby provides a Ruby 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.

New AWS services are occasionally added to the AWS SDK for Ruby. For a complete list of the services that are supported by the SDK for Ruby, see Supported Services on the AWS SDK for Ruby home page.

Topics

What's in this Guide?

This is the AWS SDK for Ruby Developer Guide, which aims to provide you with information about how to install, set up, and use the SDK for Ruby to program applications in Ruby that can make full use of the services offered by Amazon Web Services.

Here is a guide to the contents:

Getting Started (p. 3)
: If you are just starting out with the SDK for Ruby, you should first read through the Getting Started (p. 3) section. It will guide you through downloading and installing the AWS SDK for Ruby, and how to set up your development environment.

Using AWS Services (p. 9)
: This chapter provides specific guidance about using the SDK for Ruby with various AWS services.

Additional Resources (p. 20)
: This chapter provides information about additional resources that you can use to learn about the SDK for Ruby.

Document History (p. 25)
: This chapter provides details about major changes to the documentation. New sections and topics as well as significantly revised topics are listed here.

Viewing the Revision History for the SDK for Ruby

The AWS SDK for Ruby is regularly updated to support new services and new service features. To see what changed with a given release, you can check the release notes history.

Each release of the AWS SDK for Ruby is also 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 the plus sign next to that commit.

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, see Test-Driving AWS in the Free Usage Tier. To obtain an AWS account, open the AWS home page and then click Sign Up.

Getting Started with the AWS SDK for Ruby

This section provides information about how to install, set up, and use the AWS SDK for Ruby. If you have never used the SDK for Ruby before, you should start here.

Topics

Get an AWS Account and Your AWS Credentials

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

To sign up for an AWS account

1. Open http://aws.amazon.com, and then click **Sign Up**.
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**.

To get your access key ID and secret access key

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS Management Console.

Note
To create access keys, you must have permissions to perform the required IAM actions. For more information, see Granting IAM User Permission to Manage Password Policy and Credentials in *Using IAM*.

1. Open the IAM console.
2. From the navigation menu, click **Users**.
3. Select your IAM user name.
4. Click **User Actions**, and then click **Manage Access Keys**.
5. Click **Create Access Key**.

   Your keys will look something like this:

   - Access key ID example: AKIAIOSFODNN7EXAMPLE
   - Secret access key example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

6. Click **Download Credentials**, and store the keys in a secure location.

   Your secret key will no longer be available through the AWS Management Console; you will have the only copy. Keep it confidential in order to protect your account, 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.

Related topics

- What Is IAM? in *Using IAM*
- AWS Security Credentials in *AWS General Reference*

Installing the SDK for Ruby

The SDK for Ruby is packaged as a gem. To install the SDK, enter the following command:

```
gem install aws-sdk
```

Tip
The AWS Ruby SDK runs on Ruby 1.8.7 and later versions. If you have an older version of Ruby, Ruby enVironment Manager (RVM) is a great way to try the latest version. RVM is a command-line tool that manages multiple versions of Ruby on a single computer.

Setting up AWS Credentials for Use with the SDK for Ruby

To use the SDK for Ruby to access AWS resources, you must provide a set of AWS credentials, which consist of an access key ID and a secret access key. We recommend that you do not use your account's root credentials to access AWS. Instead, create one or more IAM users and provide those credentials to the SDK for Ruby. In addition to providing a better way to manage credentials, each IAM user has one or more attached policies that specify which resources the user can access, and which actions they can perform on those resources. For more information, see Best Practices for Managing AWS Access Keys.

The recommended approach for managing credentials is to store them in one of the following ways and then load them into your application. This avoids putting explicit keys in your code, where they might be inadvertently exposed. For more information on how to load credentials, see Specifying Credentials (p. 5).

Credentials file

Create one or more profiles in your local system's AWS credentials file, which is located at `~/.aws/credentials` (Linux, Unix, and OS X systems) or `C:\Users\User_Name\.aws\credentials` (Windows systems). Each profile consists of a name and a set of credentials in the following format:

```
[profile_name]
aws_access_key_id = access_key_id
aws_secret_access_key = secret_access_key
```

Substitute a set of IAM credentials for the *access_key_id* and *secret_access_key*. You can use any name you prefer for the profile name except for the default profile, which must be named `default`. For example, if you have multiple IAM users with different policies, you can create a profile for each user, named with the user name. You can specify the appropriate profile by name for each application.

Environment variables

Specify default credentials by assigning an access key ID and a secret access key to the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

For Linux, OS X, or Unix systems, use **export**:

```
export AWS_ACCESS_KEY_ID=access_key_id
export AWS_SECRET_ACCESS_KEY=secret_access_key
```

For Windows, use **set**:

```
set AWS_ACCESS_KEY_ID=access_key_id
set AWS_SECRET_ACCESS_KEY=secret_access_key
```

IAM role

For applications running on an EC2 instance, the recommended approach is to create an IAM role with appropriate permissions and assign it to the instance. The application will then run with those permissions, as shown in Using IAM Roles for Amazon EC2 Instances with the AWS SDK for Ruby (p. 9).

The AWS SDKs and CLIs use a *provider chain* to look for default AWS credentials, or you can specify the appropriate credentials explicitly. For more information, see Specifying Credentials (p. 5).

Specifying AWS Credentials for SDK for Ruby Applications

For an SDK for Ruby application to access Amazon Web Services, you must provide the application with a set AWS credentials that have appropriate permissions. You specify which credentials your application will use to access a service when you initialize a new service client. This topic describes how to specify AWS credentials for SDK for Ruby applications.

Topics

Using Default Credentials

One option is to create the client without any arguments, as shown in the following example.

```
# Create a new S3 object
s3 = AWS::S3.new
```

In this case, the SDK for Ruby attempts to use your default AWS credentials to create the client object. It locates them by using the *default credential provider chain*, which is implemented by the DefaultProvider class. It looks for default credentials in the following order and loads the first set that it finds:

1. Credentials passed to the AWS.config method with the `:access_key_id` and `:secret_access_key_id` options.
2. **Environment Variables** – `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

   The SDK for Ruby uses the ENVProvider class to load these credentials.
3. **The credentials file's default profile** – For more information about the credentials file, see Setting up AWS Credentials (p. 4).

   The SDK for Ruby uses the SharedCredentialFileProvider to load profiles.
4. **Instance profile credentials** – these credentials can be assigned to Amazon EC2 instances, and are delivered through the Amazon EC2 metadata service.

   The SDK for Ruby uses EC2Provider to load these credentials.

Note
You must have specified default credentials in at least one of these ways, or the attempt will fail. For more information, see Setting up AWS Credentials (p. 4).

Specifying a Credentials Provider

If you don't want to use the default credentials provider, you can specify your preferred credentials provider. For example, you might want to use a nondefault profile from the credentials file, which is handled by `SharedCredentialFileProvider`. You can specify a credentials provider as follows:

- To specify a credentials provider for a particular service, pass a provider object to the service's client constructor, which must take a class with the Provider interface as input. A provider typically has options that you can use to specify a particular set of credentials.

  The following example directs the Amazon S3 client constructor to get its credentials from the `SharedCredentialFileProvider` object, which loads credentials from the credentials file.

```
# Create a new S3 object using a specific provider
s3 = AWS::S3.new(
  :credential_provider => AWS::Core::CredentialProviders::SharedCredentialFile
Provider.new)
```

Creating the provider object with the default constructor specifies the credentials file's default profile. You can also use the constructor's `profile_name` option to specify a profile by name or its `path` option to specify a nondefault credentials file location.

- You can specify a credential provider that will be used by all service clients by passing a provider object to the AWS.config method. The following example specifies the default profile from the credentials file for all services.

```
AWS.config(
  :credential_provider => AWS::Core::CredentialProviders::SharedCredentialFile
Provider.new)
# Create a new S3 object
s3 = AWS::S3.new
```

  You can then use the Amazon S3 client object's default constructor, which automatically loads the appropriate profile.

For the full list of SDK for Ruby credential provider classes, see AWS::Core::CredentialProviders. In the SDK for Ruby, all provider classes mix in the Provider interface.

Tip
You can also use this approach to supply your own credentials provider. Just implement a credentials provider class that mixes in the **Provider** interface. You can then pass an instance of the class to a service's client class constructor or to `AWS.config`, as described above.

Explicitly Specifying Credentials

You can specify credentials explicitly as follows.

- Specify credentials for a particular service by using the client class constructor's `:access_key_id` and `:secret_access_key` options.

  The following example specifies an explicit set of credentials for the Amazon S3 client object.

```
s3 = AWS::S3.new(
  :access_key_id => 'AKIAIOSFODNN7EXAMPLE',
  :secret_access_key => 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY')
```

- Specify credentials for every service by using the `AWS.config` method's `:access_key_id` and `:secret_access_key` options.

  The following example specifies an explicit set of credentials for every service, so you can use the default constructor to create the Amazon S3 client object. It will automatically use the credentials that you passed to `AWS.config`.

```
AWS.config(
  :access_key_id => 'AKIAIOSFODNN7EXAMPLE',
  :secret_access_key => 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY')
```

```
# S3.new will now use the credentials specified in AWS.config
s3 = AWS::S3.new
```

Warning
If you explicitly specify credentials, make sure that you do not accidentally expose your code on a publicly accessible site, such as a public GitHub repository. Doing so could compromise your account's integrity.

Where to Go from Here

The SDK reference documentation provides information about both the AWS Ruby gem and AWS Rails integration gem.

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

The SDK for Ruby is packaged with a number of code samples, which you can browse on your machine or view on GitHub. For more information about the samples that are provided, see AWS SDK for Ruby Samples (p. 20).

Using Amazon Web Services with the AWS SDK for Ruby

This section provides information about how to program various Amazon Web Services using the SDK for Ruby.

Topics

Using IAM Roles for Amazon EC2 Instances with the AWS SDK for Ruby

Note
For in-depth information about using IAM roles for EC2 instances, see Roles in *Using IAM*.

Securely managing authentication credentials is one of the first challenges that developers will 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 (Amazon EC2) instances, developers must store these credentials in a way that keeps them secure but also makes them accessible to the software, which needs them in order to make requests.

IAM roles for EC2 instances provides you with an effective way to manage credentials for AWS software running on EC2 instances. With IAM roles, you can develop software and deploy it to an EC2 instance without having to otherwise manage the credentials that the software is using.

Topics

Using IAM Roles for EC2 Instances to Manage Your Credentials

You can use the AWS Management Console to create an IAM role and configure it with the permissions that your software requires. Permissions for IAM roles are specified in a way similar to permissions for IAM users. For more information, see IAM Users and Groups in *Using IAM*.

EC2 can access credentials using EC2's Instance Metadata Service (IMDS), which can securely provide credentials using the IAM role you create. The instance metadata service is part of the default credential provider chain, so you don't need to change your code to use it if you're already using the default provider chain for your application.

Note
The default credential provider chain is explained in detail in the topic: Specifying Credentials (p. 5).

You can also specify the EC2 provider explicitly, by passing an instance of EC2Provider to either AWS.config or to your service object during initialization. For example:

```
AWS.config(:credential_provider => AWS::Core::CredentialProviders::EC2Pro
vider.new)
```

The **EC2Provider** object uses the IMDS to retrieve temporary credentials that have the same permissions as those associated with the IAM role. Although the credentials are temporary and eventually expire, the SDK periodically refreshes them so that they continue to enable access. This refresh is completely transparent to your code—you don't need to initiate it yourself.

Note
AWS CloudFormation does not support calling its API with an IAM role. You must call the AWS CloudFormation API as a regular IAM user.

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

In this walkthrough, we'll begin with a program that retrieves an object from Amazon S3 using regular account credentials loaded from the environment. Then, we'll use the IMDS in conjunction with an IAM role to get credentials.

Important
This tutorial assumes that you have installed the AWS SDK for Ruby and a compatible Ruby interpreter. If you have not done so, install the SDK (p. 3) before proceeding.

Create the sample program

Here's the program:

```
require 'rubygems'
require 'aws-sdk'

s3 = AWS::S3.new

bucket_name = 'text-content'
obj_name = 'text-object.txt'
```

```
document = s3.buckets[bucket_name].objects[obj_name]

File.open(obj_name, "w") do |f|
  f.write(document.read)
end

puts "'#{obj_name}' copied from S3."
```

You can either type or copy this code into a file on your machine. Name it whatever you like. For the purposes of the tutorial, we'll call it `get_object.rb`.

This code initializes an `AWS::S3` object without specifying any credentials, which initiates a search for credentials using the *default credential provider chain*. The default credential provider chain looks for credentials specified in the environment, and failing that, will look for credentials from EC2's metadata service.

You can use this feature to test your code locally by specifying credentials in the environment. On an EC2 instance, your code will work with the IMDS without any modification.

To test the program locally

1. Specify *your AWS credentials* in the local environment varables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. For more information about how to do this, see Setting up AWS Credentials (p. 4).
2. Substitute the names of an *Amazon S3 bucket* and *text object* associated with your AWS account for the values of `bucket_name` and `obj_name`, respectively.

   For instructions about how to create an Amazon S3 bucket and upload an object, see the Amazon Simple Storage Service Getting Started Guide.
3. Open a terminal and change to the directory where you saved your application code. For example, if you placed it in a directory called `aws_ruby_test` in your home directory, you would type:

   ```
   cd ~/aws_ruby_test
   ```

4. Run the program with the Ruby interpreter:

   ```
   ruby get_object.rb
   ```

Run the Program on EC2 using IAM Roles

Next, we'll run the program on an EC2 instance using IAM roles. To do this, we'll take the following actions:

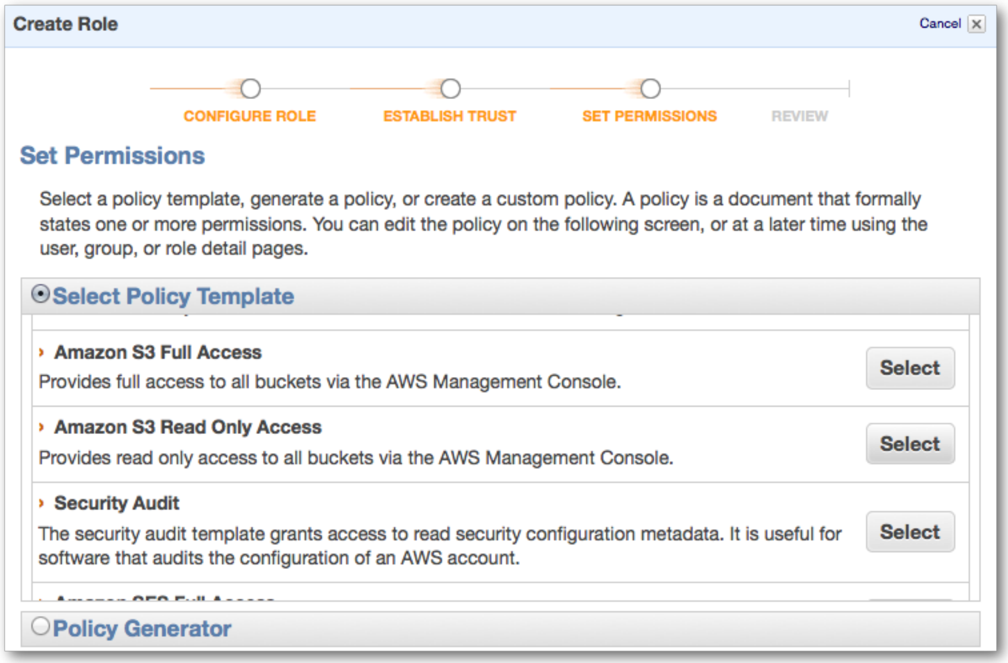
1. Create an IAM role using the AWS Management Console (p. 12)
2. Launch an EC2 instance with the corresponding instance profile (p. 12)
3. Transfer the source to your EC2 instance (p. 14)
4. Run the program within the EC2 instance (p. 15)

We'll now examine each of these steps in detail.

Create an IAM role using the AWS Management Console

The first step is to create an IAM role that has the appropriate permissions. To create the IAM role, follow the procedure Creating a IAM Role in *Using IAM*.

When creating the role, select **Amazon EC2** as the role type, and then select **Amazon S3 Read Only Access** as the permission type:



Set Permissions

Select a policy template, generate a policy, or create a custom policy. A policy is a document that formally states one or more permissions. You can edit the policy on the following screen, or at a later time using the user, group, or role detail pages.

Select Policy Template

› Amazon S3 Full Access
Provides full access to all buckets via the AWS Management Console.
Select

› Amazon S3 Read Only Access
Provides read only access to all buckets via the AWS Management Console.
Select

› Security Audit
The security audit template grants access to read security configuration metadata. It is useful for software that audits the configuration of an AWS account.
Select

Policy Generator

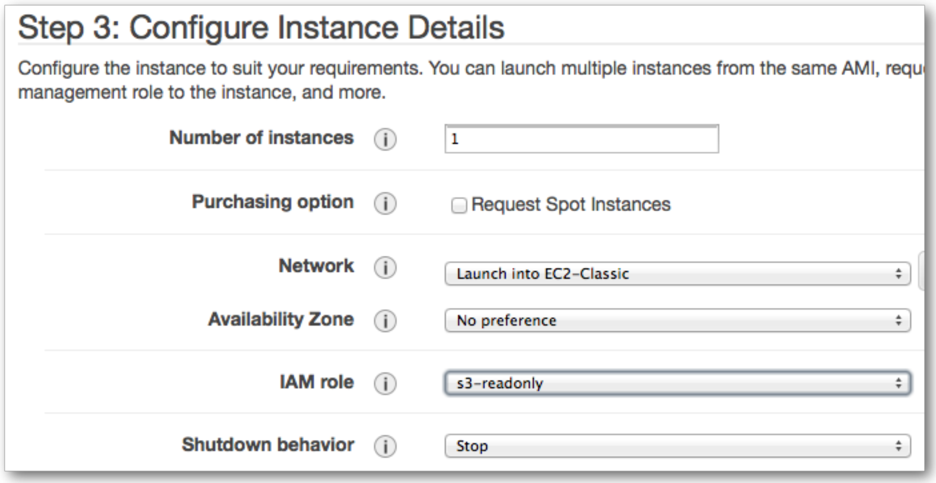
Policies can also be represented in JSON format. The following JSON block describes the policy for Amazon S3 read-only access.

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

Note 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 for Linux Instances*. 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.



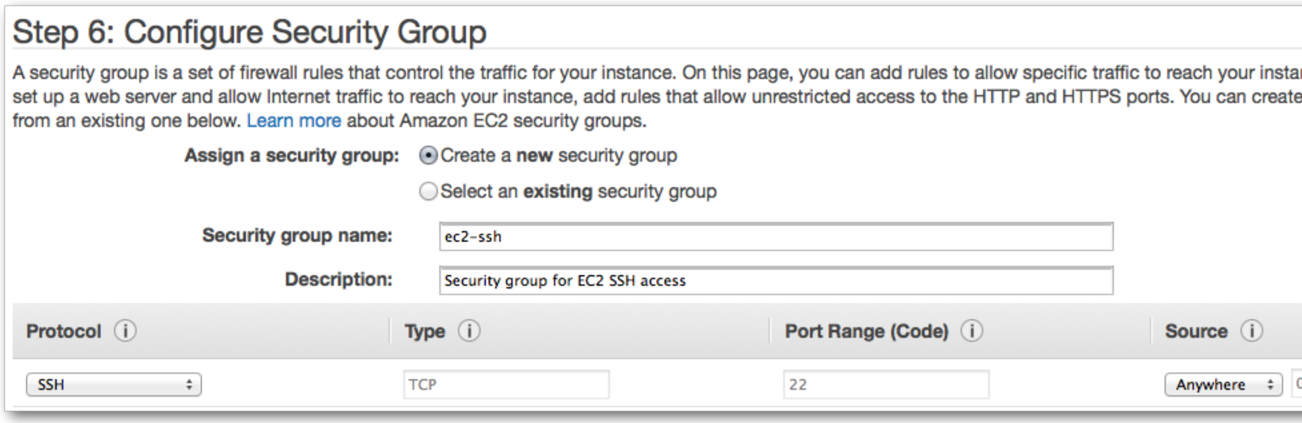
Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, requ
management role to the instance, and more.

Number of instances	1
Purchasing option	☐ Request Spot Instances
Network	Launch into EC2-Classic
Availability Zone	No preference
IAM role	s3-readonly
Shutdown behavior	Stop

When you create your EC2 instance, you must specify a *key pair* and a *security group* for access. You can create and configure these while setting up your EC2 instance.

When setting up your security group, make sure that it has SSH access enabled:



Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your insta
set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create
from an existing one below. Learn more about Amazon EC2 security groups.

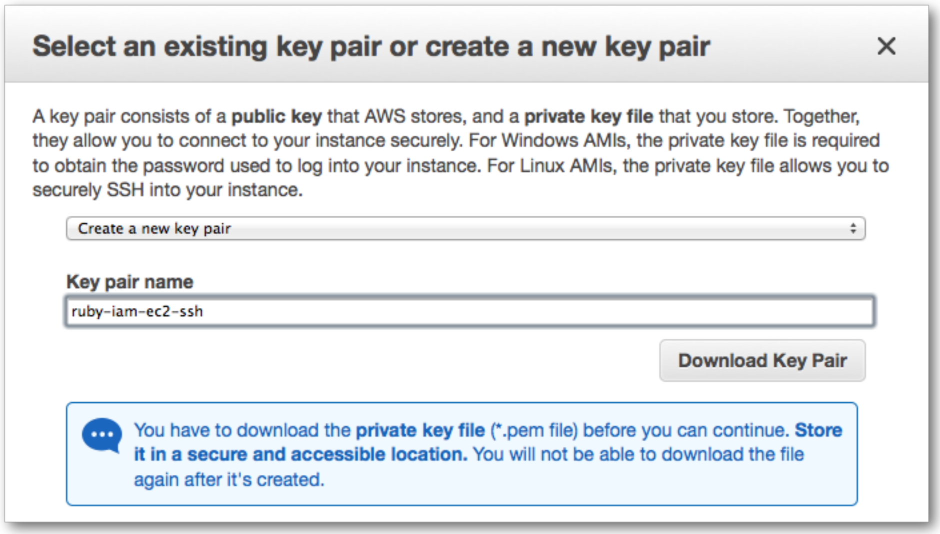
Assign a security group: [● Create a **new** security group] ○ Select an **existing** security group

Security group name: ec2-ssh

Description: Security group for EC2 SSH access

Protocol	Type	Port Range (Code)	Source
SSH	TCP	22	Anywhere

Be sure to download the `.pem` file for the keypair you created:



Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

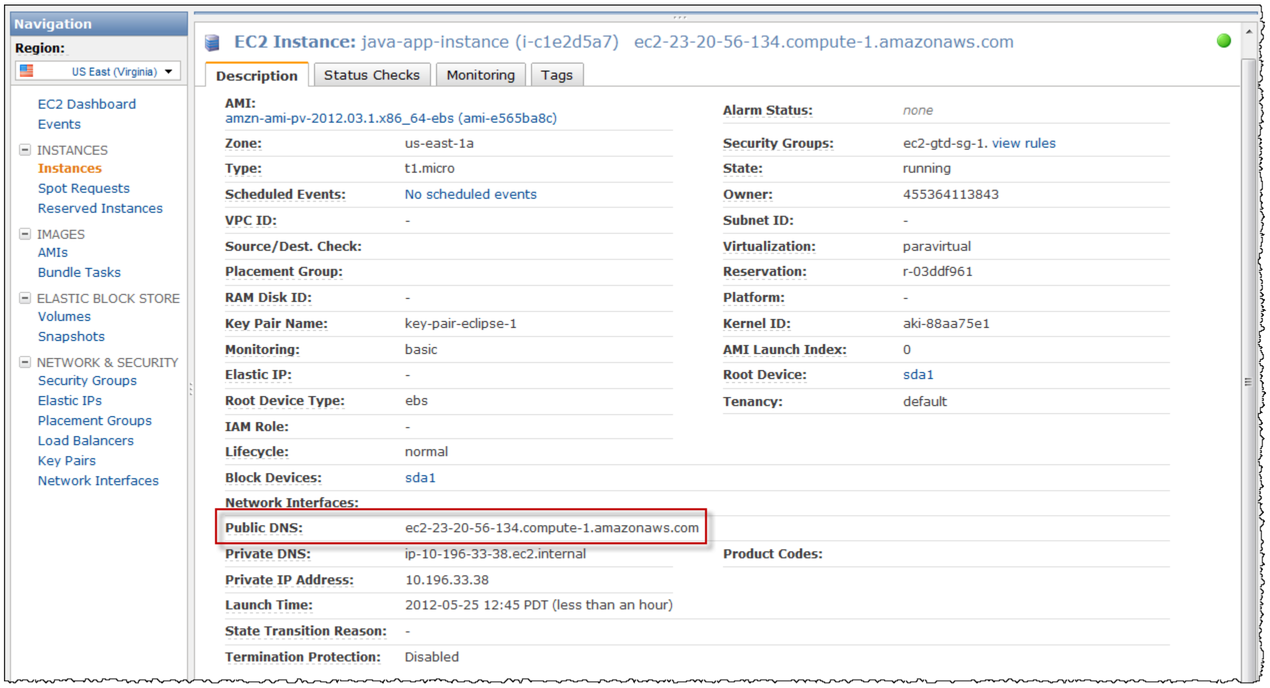
Create a new key pair

Key pair name: ruby-iam-ec2-ssh

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

When you are finished, click **Launch Instance** to launch your EC2 instance.

Go to the **EC2 Instances** area of the AWS Management Console and view the launch status of your instance. Once your instance is **running**, copy its public DNS name. You will use this DNS name to connect to the instance.



EC2 Instance: java-app-instance (i-c1e2d5a7) ec2-23-20-56-134.compute-1.amazonaws.com

AMI: amzn-ami-pv-2012.03.1.x86_64-ebs (ami-e565ba8c)

Zone:	us-east-1a	Alarm Status:	none
Type:	t1.micro	Security Groups:	ec2-gtd-sg-1. view rules
Scheduled Events:	No scheduled events	State:	running
VPC ID:	-	Owner:	455364113843
Source/Dest. Check:		Subnet ID:	-
Placement Group:		Virtualization:	paravirtual
RAM Disk ID:	-	Reservation:	r-03ddf961
Key Pair Name:	key-pair-eclipse-1	Platform:	-
Monitoring:	basic	Kernel ID:	aki-88aa75e1
Elastic IP:	-	AMI Launch Index:	0
Root Device Type:	ebs	Root Device:	sda1
IAM Role:	-	Tenancy:	default
Lifecycle:	normal		
Block Devices:	sda1		
Network Interfaces:			
Public DNS:	ec2-23-20-56-134.compute-1.amazonaws.com		
Private DNS:	ip-10-196-33-38.ec2.internal	Product Codes:	
Private IP Address:	10.196.33.38		
Launch Time:	2012-05-25 12:45 PDT (less than an hour)		
State Transition Reason:	-		
Termination Protection:	Disabled		

Transfer the source to your EC2 instance

Transfer the modified source file to your EC2 instance using **scp**. Be sure to specify the `.pem` file you created earlier, and use the public DNS name of the instance to connect with. The command will look something like this:

```
scp -i ruby-iam-ec2-ssh.pem get_object.rb \
ec2-user@ec2-23-20-56-134.compute-1.amazonaws.com:
```

Note
If you launched an EC2 machine image 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**. Additionally, the steps for configuring and running the program in the next section may differ somewhat.

Run the program within the EC2 instance

To run the program

1. Connect to your EC2 instance with **ssh**. Use the same public DNS name and `.pem` file you used to copy the source code in the preceding section—for example:

   ```
   ssh -i ruby-iam-ec2-ssh.pem ec2-user@ec2-23-20-56-134.compute-1.amazonaws.com
   ```

2. The Amazon Linux AMI has Ruby 1.8.7 installed by default. However, we recommend using Ruby 1.9 with the AWS SDK for Ruby. To install Ruby 1.9 on the Amazon Linux AMI, use the following command:

   ```
   sudo yum install ruby19 rubygems19
   ```

3. Set your AMI to use Ruby 1.9 by default by executing the following command:

   ```
   sudo alternatives --set ruby /usr/bin/ruby1.9
   ```

4. Install additional development packages needed by the AWS SDK for Ruby:

   ```
   sudo yum install -y gcc ruby-devel19 libxml2 libxml2-devel libxslt libxslt-
   devel make
   ```

5. Install the AWS SDK for Ruby:

   ```
   sudo gem install aws-sdk --no-ri --no-rdoc
   ```

   The `--no-ri` and `--no-rdoc` options tell gem to not compile the Ruby documentation for the `aws-sdk` gem. This will make the installation considerably faster.
6. Run the program:

   ```
   ruby get_object.rb
   ```

If everything is set up correctly, the file should be copied to your EC2 instance just as it was to your local machine when you were using credentials. This time, however, the credentials you used were not stored in your program or on your EC2 instance. Instead, IAM managed the credentials for you!

Start an Amazon EC2 Instance

This section demonstrates how to use the AWS SDK for Ruby 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 run Amazon EC2 instances. Before configuring your client, you must create a YAML file to store your AWS Access Key and your Secret Key. This YAML file must be placed in the same directory as your Ruby program.

The file looks like this:

```
access_key_id: YOUR_ACCESS_KEY
secret_access_key: YOUR_SECRET_KEY
```

Specify your AWS credentials as values for the `access_key_id` and `secret_access_key` entries. 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. Pass your configuration file into the AWS.config method, as follows:

```
config_file = File.join(File.dirname(__FILE__), "config.yml")

AWS.config(YAML.load(File.read(config_file)))
```

2. Create a new EC2 instance, specifying the service endpoint as follows:

```
ec2 = AWS::EC2.new(region: 'us-west-1')
```

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

Before running an Amazon EC2 instance, you will need to create an 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 an Amazon EC2 Security Group (p. 17).

For information about authorizing security group ingress, see Authorize Amazon EC2 Security Group Ingress (p. 17).

For information about creating a key pair, see Create a Key Pair (p. 18).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 18).

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. 17).

To create a security group, use the SecurityGroupCollection.create method and pass the name of a security group you created. The method returns a SecurityGroup object, as follows:

```
security_group = ec2.security_groups.create('YOUR_SECURITY_GROUP_NAME')
```

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, the method returns an error.

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. You can use the returned `SecurityGroup` object to authorize or revoke security group ingress and egress. You must also create a key pair to allow you to log into your instance.

For information about authorizing security group ingress, see Authorize Amazon EC2 Security Group Ingress (p. 17).

For information about creating a key pair, see Create a Key Pair (p. 18).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 18).

Authorize Security Group Ingress

By default, a new security group does not allow any inbound traffic. 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 protocol, and for TCP/UDP ports.

To authorize ingress for your security group, use the SecurityGroup.authorize_ingress method.

The following code demonstrates one way to authorize security group ingress for a range of IP addresses.

```
ip_addresses = ['111.111.111.111/0', '150.150.150.150/0']

security_group.authorize_ingress :tcp, 22, *ip_addresses
```

Specify the IP address using CIDR notation. If you specify the protocol as TCP/UDP, you must provide a source port or a range of ports. You can authorize ports only if you specify TCP or UDP.

If you authorize ingress for IP addresses that have already been authorized, the method returns an error.

Whenever you use `authorize_ingress` or SecurityGroup.authorize_egress, 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.

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 KeyPairCollection.create method and specify the key pair name. The method returns a KeyPair object, as follows:

   ```
   key_pair = ec2.key_pairs.create('YOUR_KEY_PAIR_NAME')
   ```

   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 occurs.
2. Use the returned object's fingerprint property to obtain an SHA-1 digest of the DER-encoded private key, as follows:

   ```
   private_key = key_pair.private_key;
   ```

   Calling `create` 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 about how to run an Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 18).

For information about how to use your key pair to connect to your Amazon EC2 instance, see Connect to Your Amazon EC2 Instance (p. 19).

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. 18). For information about creating a security group, see Create an Amazon EC2 Security Group (p. 17).

Use the InstanceCollection.create method to run an Amazon EC2 instance. Specify the Amazon Machine Image (AMI), the instance type, the maximum number of instance to run, the names of a security group and key pair you created, as follows:

```
instance = ec2.instances.create(
:image_id => 'ami-11d68a54',
:instance_type => 'm1.small',
:count => 1,
:security_groups => 'YOUR_SECURITY_GROUP_NAME',
:key_pair => ec2.key_pairs['YOUR_KEY_PAIR_NAME'])
```

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 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, this is 20. If fewer instances are 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. 19).

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 about how to obtain the private key, see Create a Key Pair (p. 18).

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 about authorizing security group ingress, see Authorize Security Group Ingress (p. 17).

For information about 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 Ruby.

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

Additional Resources

Topics

Home Page for AWS SDK for Ruby

For more information about the AWS SDK for Ruby, go to the homepage for the SDK at http://aws.amazon.com/sdkforruby.

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.aws.amazon.com/sdkforruby/latest/apidocs/Index.html.

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. There is a forum specifically for AWS development in Ruby 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.

To visit the AWS forums, visit aws.amazon.com/forums

AWS SDK for Ruby Samples

The AWS SDK for Ruby is packaged with a number of samples that demonstrate basic usage of the SDK for Ruby with AWS services such as Amazon S3 Amazon EC2 and more. By studying and running these samples, you can quickly gain understanding of how to use the SDK for Ruby and implement typical AWS programming patterns in Ruby.

Topics

Installing the samples

When you install the SDK for Ruby using `gem install`, you get the libraries that you can use to start building AWS applications in Ruby. However, to get the sample code, you'll need to download the SDK source itself.

The SDK for Ruby is an open-source project hosted on GitHub. However, you don't need an account on GitHub just to download the source code. In fact, you don't even need the git source control manager, though if you do, updating the source to keep up with new versions of the SDK is easy.

To get the SDK for Ruby source, follow one of the following procedures.

To download the source with git

1. Open a terminal window and change to the directory where you want to clone the SDK for Ruby source code. For example:

```
cd ~/source
```

2. Clone the AWS SDK for Ruby project with git:

```
git clone https://github.com/aws/aws-sdk-ruby.git
```

Note
If you have an account on GitHub and your have installed an SSH key, you can use the the SSH git URL instead:

```
git clone git@github.com:aws/aws-sdk-ruby.git
```

To download the source without git

1. Download the latest SDK for Ruby source code archive using the following URL (either with your browser, **curl**, **wget** or similar):

   - https://github.com/aws/aws-sdk-ruby/archive/master.zip

2. Unzip it into a local directory.
3. Open a terminal window and change to the directory where you unzipped the source archive.

The samples are in the `samples` directory in the SDK for Ruby source. You can view the samples by listing this directory:

```
ls samples
```

> **Note**
> On Windows, use the **dir** command instead.

Set your AWS Credentials

Because these are AWS SDK for Ruby samples, you will need to provide AWS credentials so that they can communicate with AWS.

The samples generally use the default credential provider chain (p. 5) to load credentials. The easiest way to set your credentials so that they can be loaded by the default provider chain is to set the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

For more information about how to set credentials for use with the SDK, including alternate ways to set credentials, see Setting up AWS Credentials (p. 4).

Run the Samples

Once you have downloaded the samples and have your credentials set, you can run the samples to see how they interact with AWS. We'll demonstrate using the `upload_file.rb` sample, located in the `aws-sdk-ruby/samples/s3` directory.

To run the upload_file sample

1. Open a terminal window and change to the `samples/s3` directory containing the `upload_file.rb` sample. For example, if you put the SDK for Ruby source in `~/source/aws-sdk-ruby`, then type:

   ```
   cd ~/source/aws-sdk-ruby/samples/s3
   ```

2. The `upload_file.rb` script requires a file to upload to S3. Copy any test file you like to the current directory. For example, if you have a test file called `test.png` in your `Pictures` directory, you would type:

   ```
   cp ~/Pictures/test.png .
   ```

3. Run the `upload_file.rb` script with Ruby, providing it with an S3 bucket name (the bucket will be created if it doesn't already exist) and the name of the file you want to upload. For example:

   ```
   ruby upload_file.rb my_test_bucket test.png
   ```

If successful, the sample will output the S3 URL of the file that you just upoaded, with an option to delete it:

```

Uploaded test.png to:
https://s3.amazonaws.com/my_test_bucket/test.png

Use this URL to download the file:
```

```
https://s3.amazonaws.com/my_test_bucket/test.png
(press any key to delete the object)
```

To use the AWS Object Relational Manager (ORM) in a Rails 3 application

1. Install the gem:

   ```
   $ gem install aws-sdk
   ```

2. Start a new Rails project:

   ```

   $ gem install rails
   $ rails new myapp
   $ cd myapp/
   ```

3. Add the following line to your Gemfile:

   ```
   gem 'aws-sdk'
   ```

4. Install dependencies:

   ```
   bundle install
   ```

5. Configure AWS with your access credentials.

   You can use a config initializer script (e.g., `config/initializers/aws.rb`) and use Ruby to configure your AWS credentials:

   ```

   AWS.config({
   :access_key_id => 'REPLACE_WITH_ACCESS_KEY_ID',
   :secret_access_key => 'REPLACE_WITH_SECRET_ACCESS_KEY',
   })
   ```

   Or you can create a `config/aws.yml` file that will also be automatically loaded with Rails:

   ```

   # Just like config/database.yml, this file requires an entry for each envir
   onment
   # http://aws.amazon.com/security-credentials
   development:
   access_key_id: REPLACE_WITH_ACCESS_KEY_ID
   secret_access_key: REPLACE_WITH_SECRET_ACCESS_KEY

   test:
     <<: *development

     production:
       <<: *development
   ```

6. Create `app/models/my_record.rb` as follows:

```
class MyRecord < AWS::Record::Base
    string_attr :name
  end
```

7. Create the SimpleDB domain:

```
$ rails console
> MyRecord.create_domain
```

8. Now, you can play around with the model by creating some records and querying them:

```
> MyRecord.find(:all).to_a
=> []

> MyRecord.new(:name => "The first one").save
=> true

> MyRecord.new(:name => "The second one").save
=> true

> MyRecord.where('name like ?', "%first%").count
=> 1
```

   Exit the rails console before continuing to the next step:

```
> exit
```

To generate a scaffold controller for your model

1. Type the following command:

```
$ rails generate scaffold_controller MyRecord name:string
rails server
```

2. Add a route to your scaffold controller in **`config/routes.rb`**:

```
Myapp::Application.routes.draw do
  # add this line:
  resources :my_records
end
```

   Now, you can create records in the browser at localhost:3000/my_records. Note that this link is valid only if you have completed the above procedure.

Document History

The following table describes the important changes since the last release of the *AWS SDK for Ruby Developer Guide*.

Last documentation update: September 9, 2013

Change	Description	Release Date
Updated Documentation	The documentation has been restructured and the Getting Started (p. 3) chapter has been revised with new guidance. The topic, AWS SDK for Ruby Samples (p. 20), has been updated and moved into Additional Resources (p. 20).	May 17, 2014
New topic	The new topic, Specifying Credentials (p. 5), discusses how to load AWS credentials with the AWS SDK for Ruby.	May 17, 2014
New topic	This topic tracks recent changes to the *AWS SDK for Ruby Developer Guide*. It is intended as a companion to the release notes history.	September 9, 2013