AWS SDK for Ruby

Developer Guide

Version v1.0.0

AWS SDK for Ruby: Developer Guide

Copyright © 2014 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

AWS SDK for Ruby Developer Guide

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.

Getting Started

If you are just starting with the AWS SDK for Ruby, 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.

Supported Services

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

- **Compute**
  - Amazon EC2
  - Auto Scaling
  - Amazon Elastic MapReduce
- **Content Delivery**
  - CloudFront
- **Database**
  - Amazon DynamoDB
  - Amazon SimpleDB
  - Amazon RDS
  - Amazon Elasticache
  - Amazon RedShift
- **Deployment & Management**
  - AWS Elastic Beanstalk

  - AWS CloudFormation
  - AWS Data Pipeline
  - AWS OpsWorks
- **Application Services**
  - Amazon CloudSearch
  - Amazon Elastic Transcoder
  - Amazon SWF
  - Amazon SNS
  - Amazon SQS
  - Amazon SES
- **Monitoring**
  - CloudWatch
- **Networking**
  - Amazon Route 53
  - Amazon VPC
  - AWS Direct Connect
  - Elastic Load Balancing
- **Security**
  - AWS Identity and Access Management
  - AWS Security Token Service
- **Storage**
  - Amazon S3
  - Amazon Glacier
  - Import/Export
  - Amazon Storage Gateway

Revision History for the AWS SDK for Ruby

We regularly release updates to the AWS SDK for Ruby 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 SDK for Ruby 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 the plus sign next to that commit.

Additional Resources

The Additional 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 the AWS home page and click the **Sign Up Now** button.

Getting Started

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

- AWS account and credentials
- Ruby environment
- AWS SDK for Ruby

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://amazonaws.cn, 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 email 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://amazonaws.cn 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. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

> **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. Go to 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*

Set Up Your Ruby Environment

The AWS Ruby gem runs on Ruby 1.8.7 and later. 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.

Install the AWS SDK for Ruby

To install the AWS Ruby gem, enter the following command:

```
gem install aws-sdk
```

Install and Run the Samples Included in the SDK

To install the SDK samples, use Git to clone the AWS SDK for Ruby project from GitHub.

```
$ git clone git://github.com/aws/aws-sdk-ruby
$ cd aws-sdk-ruby/samples/
```

The subdirectories of the samples directory contain several code samples that you can run. These samples demonstrate basic usage of the SDK features with services such as Amazon S3.

To run the Amazon S3 Sample

1. Create a YAML file named `config.yml` in the samples directory as follows:

```
# Fill in your AWS Access Key ID and Secret Access Key
# http://amazonaws.cn/security-credentials
access_key_id: <REPLACE_WITH_ACCESS_KEY_ID>
secret_access_key: <REPLACE_WITH_SECRET_ACCESS_KEY>
```

2. Run a sample script with the Ruby interpreter. For example, to run the `s3/upload_file.rb` sample:

```
$ echo "Hello, World!" > helloworld.txt
$ ruby s3/upload_file.rb unique-bucket-name helloworld.txt
```

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://amazonaws.cn/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

Type the following command:

1.
```
$ 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.

Where Do I 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.

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.

Using *IAM roles for EC2 instances* provides an effective way to manage credentials for AWS software running on EC2 instances. Other common strategies for managing credentials on EC2 instances are:

- First launch an Amazon EC2 instance, and then securely transfer the credentials to the instance using a utility such as SCP (secure copy). 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 autoscaling groups.
- Embed the credentials as literal strings in the software itself. This means that anyone who comes into possession of the software can scan through the code and retrieve the credentials.
- Create a custom AMI (Amazon Machine Image) with the credentials, perhaps stored in a file on the AMI. With this approach, anyone with access to the AMI automatically has access to the credentials—which again creates an unnecessary security risk.

With each of the preceding strategies, it is cumbersome to rotate (update) the credentials. New credentials either have to be re-copied to the EC2 instance, compiled into a new build of the software, or incorporated into the creation of a new AMI.

Using IAM roles for EC2 instances is the recommended solution for securely accessing AWS services with an EC2 instance. With IAM roles, a developer 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*.

Amazon EC2 instances support the concept of an *instance profile*, which is a logical container for the IAM role. When you launch an EC2 instance, you can associate the instance with an instance profile that 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.

> **Note**
> If you're 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, such as Amazon Simple Storage Service (Amazon S3), using an overload of the constructor that does not take any parameters. When this parameterless constructor executes, it searches the "credentials provider chain." The credentials provider chain is the set of places where the constructor will attempt to find credentials if they are not specified explicitly as parameters. For Ruby, the credentials provider chain is:

- Static credentials provided to the AWS.config method. For example:

```
AWS.config(:access_key_id => '...', :secret_access_key => '...')
```

- Environment variables with an 'AWS' prefix: ENV['AWS_ACCESS_KEY'] and ENV['AWS_SECRET_ACCESS_KEY']
- Environment variables with an 'AMAZON' prefix: ENV['AMAZON_ACCESS_KEY'] and ENV['AMAZON_SECRET_ACCESS_KEY']
- Instance Metadata Service, which provides the credentials associated with the IAM role for the EC2 instance

If the client constructor does not find credentials in `AWS.config`, or in the environment, it retrieves temporary credentials that have the same permissions as those associated with the IAM role. The credentials are retrieved from the Instance Metadata Service (IMDS).

The credentials are stored by the constructor on behalf of the application software and are 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.

You can have the SDK automatically use IAM role credentials from the IMDS by specifying the following line in your program:

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

If the client constructor is not able to obtain credentials from the IMDS, or any of the earlier stages in the credentials provider chain, then it throws an AmazonClientException.

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. Then, we'll modify it to use IAM roles for EC2 instances.

Sample Program with Credentials

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

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

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

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."
```

To test the program locally

1. Substitute *your AWS credentials* for the values of `:access_key_id` and `:secret_access_key`.
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. Run the program with the AWS SDK for Ruby and a Ruby interpreter. For information about setting up the SDK, see Getting Started (p. 4).

   For example, if you've saved the code in a file called `get-object.rb`, run it by using `cd` on the command-line (terminal) to change to the directory in which you saved the file, and then type **`ruby get-object.rb`**.

Updating the Sample Program to Use IAM Roles

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

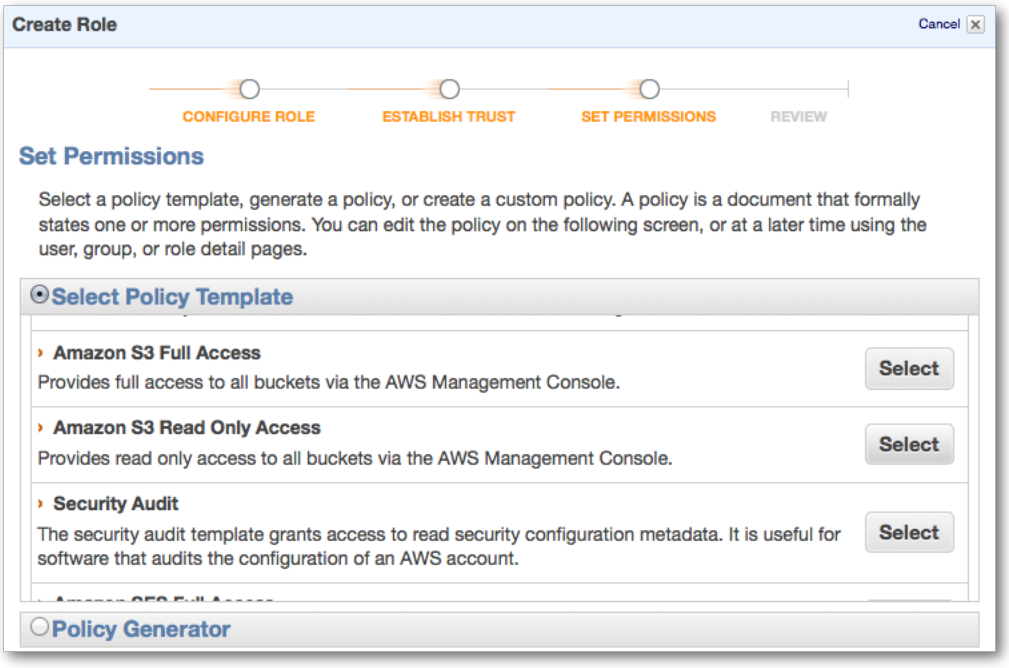
1. Create an IAM role using the AWS Management Console (p. 12)
2. Launch an EC2 instance with the corresponding instance profile (p. 13)
3. Remove the credentials from your source file (p. 14)
4. Transfer the modified source to your EC2 instance (p. 15)
5. Run the program within the EC2 instance (p. 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:



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

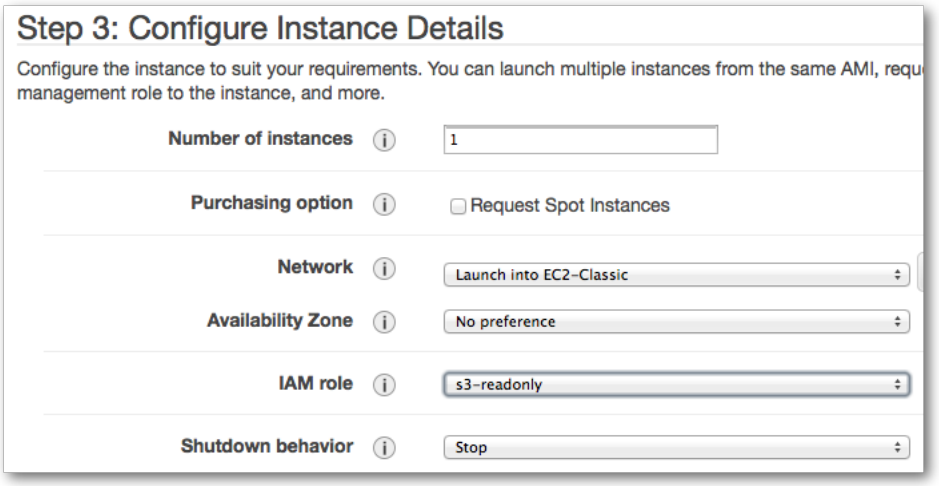
```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
```

```
        "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.

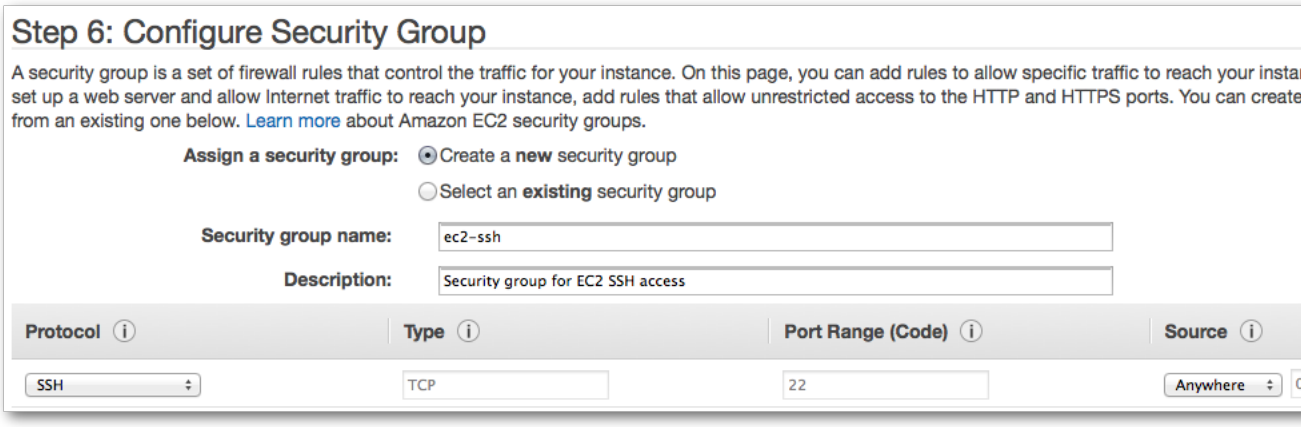
Launch an EC2 instance with the corresponding instance profile

To create an EC2 instance, follow the procedure Running an Instance in the *Amazon Elastic Compute Cloud 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.

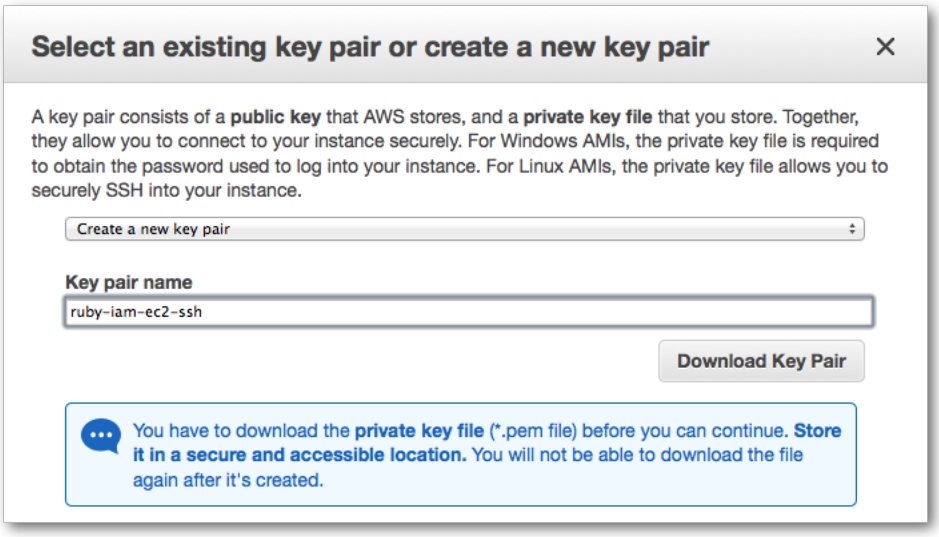


When you create your EC2 instance, you must specify a *key pair* and a *security group* for access. You can create and configure these while setting up your EC2 instance.

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

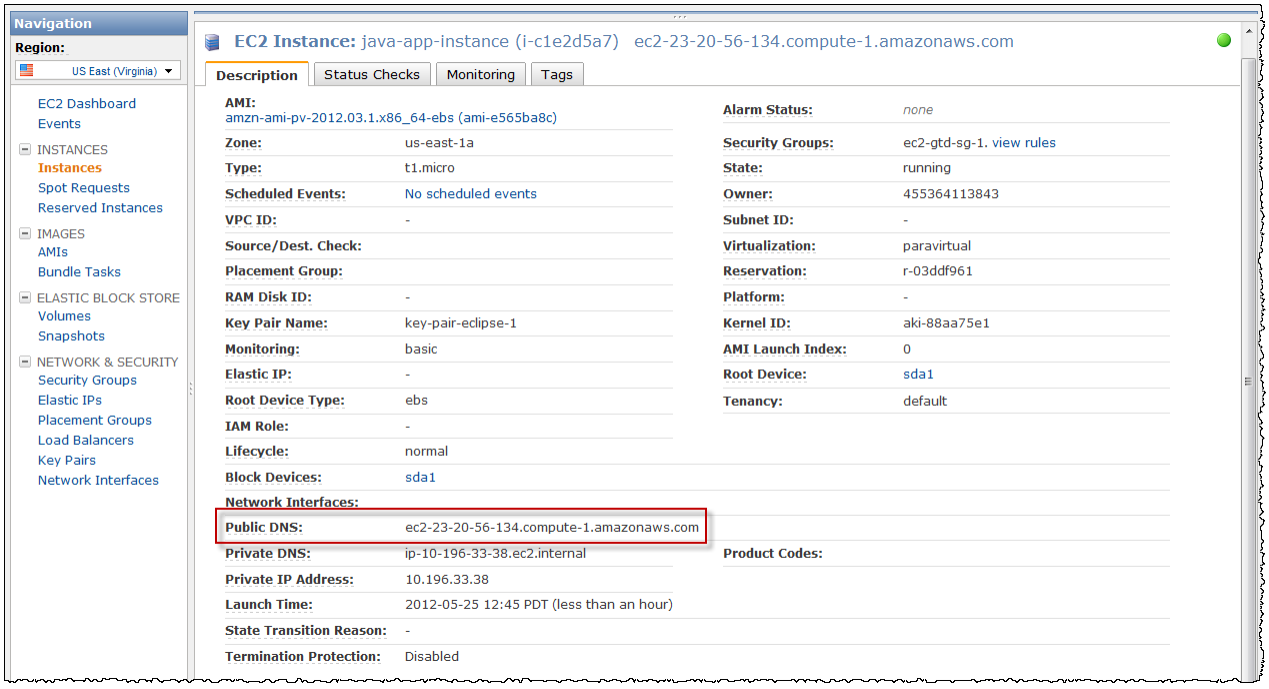


Be sure to download the `.pem` file for the keypair you created:



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.



Remove the credentials from your source file

Edit the source for the program so that it does *not* specify any credentials in the call that creates the Amazon S3 client; remove the parameters from the AWS::S3 constructor:

```
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."
```

Tip
Try running the modified program on your local system to verify that it does **not** work without credentials. If you haven't set your AWS credentials in your environment, you will get an AWS::Errors::MissingCredentialsError exception.

Transfer the modified 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(:ec2_endpoint => 'ec2.us-west-1.amazonaws.com')
```

   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 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. 18).

For information about authorizing security group ingress, see Authorize Amazon EC2 Security Group Ingress (p. 19).

For information about creating a key pair, see Create a Key Pair (p. 19).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 20).

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. 19).

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. 19).

For information about creating a key pair, see Create a Key Pair (p. 19).

For information about running your Amazon EC2 instance, see Run an Amazon EC2 Instance (p. 20).

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. 20).

For information about how to use your key pair to connect to your Amazon EC2 instance, see Connect to Your Amazon EC2 Instance (p. 20).

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

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. 20).

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. 19).

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. 19).

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

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://amazonaws.cn/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.amazonaws.cn/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 amazonaws.cn/forums

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
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