Amazon Simple Workflow Service

Developer Guide

API Version 2012-01-25

Amazon Web Services

Amazon Simple Workflow Service: Developer Guide

Amazon Web Services

What is Amazon Simple Workflow Service?

The Amazon Simple Workflow Service (Amazon SWF) makes it easy to build applications that coordinate work across distributed components. In Amazon SWF, a task represents a logical unit of work that is performed by a component of your application. Coordinating tasks across the application involves managing intertask dependencies, scheduling, and concurrency in accordance with the logical flow of the application. Amazon SWF gives you full control over implementing tasks and coordinating them without worrying about underlying complexities such as tracking their progress and maintaining their state.

When using Amazon SWF, you implement workers to perform tasks. These workers can run either on cloud infrastructure, such as Amazon Elastic Compute Cloud (Amazon EC2), or on your own premises. You can create tasks that are long-running, or that may fail, time out, or require restarts—or that may complete with varying throughput and latency. Amazon SWF stores tasks and assigns them to workers when they are ready, tracks their progress, and maintains their state, including details on their completion. To coordinate tasks, you write a program that gets the latest state of each task from Amazon SWF and uses it to initiate subsequent tasks. Amazon SWF maintains an application's execution state durably so that the application is resilient to failures in individual components. With Amazon SWF, you can implement, deploy, scale, and modify these application components independently.

Amazon SWF offers capabilities to support a variety of application requirements. It is suitable for a range of use cases that require coordination of tasks, including media processing, web application back-ends, business process workflows, and analytics pipelines.

Introduction to Amazon SWF

A growing number of applications are relying on asynchronous and distributed processing. The scalability of such applications is the primary motivation for using this approach. By designing autonomous distributed components, developers have the flexibility to deploy and scale out parts of the application independently if the load on the application increases. Another motivation is the availability of cloud services. As application developers start taking advantage of cloud computing, they have a need to combine their existing on-premises assets with additional cloud-based assets. Yet another motivation for the asynchronous and distributed approach is the inherent distributed nature of the process being modeled by the application; for example, the automation of an order fulfillment business process may span several systems and human tasks.

Developing such applications can be complicated. It requires that you coordinate the execution of multiple distributed components and deal with the increased latencies and unreliability inherent in remote communication. To accomplish this, you would typically need to write complicated infrastructure involving message queues and databases, along with the complex logic to synchronize them.

The Amazon Simple Workflow Service (Amazon SWF) makes it easier to develop asynchronous and distributed applications by providing a programming model and infrastructure for coordinating distributed components and maintaining their execution state in a reliable way. By relying on Amazon SWF, you are freed to focus on building the aspects of your application that differentiate it.

Simple Workflow Concepts

The basic concepts necessary for understanding Amazon SWF workflows are introduced below and are explained further in the subsequent sections of this guide. The following discussion is a high-level overview of the structure and components of a workflow.

The fundamental concept in Amazon SWF is the *workflow*. A workflow is a set of *activities* that carry out some objective, together with logic that coordinates the activities. For example, a workflow could receive a customer order and take whatever actions are necessary to fulfill it. Each workflow runs in an AWS resource called a *domain*, which controls the workflow's scope. An AWS account can have multiple domains, each of which can contain multiple workflows, but workflows in different domains cannot interact.

When designing an Amazon SWF workflow, you precisely define each of the required activities. You then register each activity with Amazon SWF as an activity type. When you register the activity, you provide information such as a name and version, and some timeout values based on how long you expect the activity to take. For example, a customer may have an expectation that an order will ship within 24 hours. Such expectations would inform the timeout values that you specify when registering your activities.

In the process of carrying out the workflow, some activities may need to be performed more than once, perhaps with varying inputs. For example, in a customer-order workflow, you might have an activity that handles purchased items. If the customer purchases multiple items, then this activity would have to run multiple times. Amazon SWF has the concept of an *activity task* that represents one invocation of an activity. In our example, the processing of each item would be represented by a single activity task.

An *activity worker* is a program that receives activity tasks, performs them, and provides results back. Note that the task itself might actually be performed by a person, in which case the person would use the activity worker software for the receipt and disposition of the task. An example might be a statistical analyst, who receives sets of data, analyzes them, and then sends back the analysis.

Activity tasks—and the activity workers that perform them—can run synchronously or asynchronously. They can be distributed across multiple computers, potentially in different geographic regions, or they can all run on the same computer. Different activity workers can be written in different programming languages and run on different operating systems. For example, one activity worker might be running on a desktop computer in Asia, whereas a different activity worker might be running on a hand-held computer device in North America.

The coordination logic in a workflow is contained in a software program called a *decider*. The decider schedules activity tasks, provides input data to the activity workers, processes events that arrive while the workflow is in progress, and ultimately ends (or closes) the workflow when the objective has been completed.

The role of the Amazon SWF service is to function as a reliable central hub through which data is exchanged between the decider, the activity workers, and other relevant entities such as the person administering the workflow. Amazon SWF also maintains the state of each workflow execution, which saves your application from having to store the state in a durable way.

The decider directs the workflow by receiving decision tasks from Amazon SWF and responding back to Amazon SWF with decisions. A decision represents an action or set of actions which are the next steps in the workflow. A typical decision would be to schedule an activity task. Decisions can also be used to set timers to delay the execution of an activity task, to request cancellation of activity tasks already in progress, and to complete or close the workflow.

The mechanism by which both the activity workers and the decider receive their tasks (activity tasks and decision tasks respectively) is by polling the Amazon SWF service.

Amazon SWF informs the decider of the state of the workflow by including with each decision task, a copy of the current workflow execution history. The workflow execution history is composed of events, where an event represents a significant change in the state of the workflow execution. Examples of events would be the completion of a task, notification that a task has timed out, or the expiration of a timer that was set earlier in the workflow execution. The history is a complete, consistent, and authoritative record of the workflow's progress.

A user must have authorized AWS access keys to run workflows in your account. However, access keys provide full access to all of the resources in your account and are difficult to revoke, so they are not appropriate for all applications. Amazon SWF access control uses AWS Identity and Access Management (IAM), which allows you to provide access to AWS resources in a controlled and limited way that does not expose your access keys. For example, you can allow a user to access your account, but only to run certain workflows in a particular domain.

Workflow Execution

Bringing together the ideas discussed in the preceding sections, here is an overview of the steps to develop and run a workflow in Amazon SWF:

1. Write activity workers that implement the processing steps in your workflow.

2. Write a decider to implement the coordination logic of your workflow.
3. Register your activities and workflow with Amazon SWF.

   You can do this step programmatically or by using the AWS Management Console.
4. Start your activity workers and decider.

   These actors can run on any computing device that can access an Amazon SWF endpoint. For example, you could use compute instances in the cloud, such as Amazon Elastic Compute Cloud (Amazon EC2); servers in your data center; or even a mobile device, to host a decider or activity worker. Once started, the decider and activity workers should start polling Amazon SWF for tasks.
5. Start one or more executions of your workflow.

   Executions can be initiated either programmatically or via the AWS Management Console.

   Each execution runs independently and you can provide each with its own set of input data. When an execution is started, Amazon SWF schedules the initial decision task. In response, your decider begins generating decisions which initiate activity tasks. Execution continues until your decider makes a decision to close the execution.
6. View workflow executions using the AWS Management Console.

   You can filter and view complete details of running as well as completed executions. For example, you can select an open execution to see which tasks have completed and what their results were.

Getting Set Up with Amazon SWF

Topics

This section discusses the prerequisites for developing with the Amazon Simple Workflow Service (Amazon SWF) and the development options that are available. The first step in using any AWS service is to sign up for an AWS account, discussed in detail in the following section. Once your account is set up, you have the option of developing for Amazon SWF in any of the programming languages supported by AWS. For Java and Ruby developers, the AWS Flow Framework is also available. AWS Identity and Access Management enables you to grant individuals other than the AWS account owner access to Amazon SWF resources.

AWS Account and Access Keys

To access Amazon SWF, 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**.
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.

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*

Development Options

You have a number of options for implementing your workflow solutions with the Amazon Simple Workflow Service.

Topics

HTTP Service API

Amazon SWF provides service operations that are accessible through HTTP requests. You can use these operations to communicate directly with Amazon SWF, and you can use them to develop your own libraries in any language that can communicate with Amazon SWF through HTTP.

You can develop deciders, activity workers, or workflow starters by using the service API. You can also access visibility operations through the API to develop your own monitoring and reporting tools.

For information about how to use the API, see Making HTTP Requests to Amazon SWF (p. 112) For detailed information about API operations, go to the *Amazon Simple Workflow Service API Reference*.

AWS SDKs

Amazon SWF is supported by the AWS SDKs for Java, .NET, Node.js, PHP, PHP2, Python and Ruby, providing a convenient way to use the Amazon SWF HTTP API in the programming language of your choice.

You can develop deciders, activities, or workflow starters using the API exposed by these libraries. Additionally, you can access visibility operations through these libraries so you can develop your own Amazon SWF monitoring and reporting tools.

To download any of the AWS SDKs, go to http://aws.amazon.com/code.

For detailed information about the Amazon SWF methods in each SDK, refer to the language-specific reference documentation for the SDK you are using.

Here is a list of the available AWS SDK documentation.

- AWS SDK for Java API Reference
- AWS SDK for .NET API Reference
- AWS SDK for JavaScript in Node.js API Reference
- AWS SDK for PHP API Reference
- AWS SDK for PHP API Reference
- AWS SDK for Python (Boto) API Reference
- AWS SDK for Ruby API Reference

AWS Flow Framework

The AWS Flow Framework is an enhanced SDK for writing distributed, asynchronous programs that can run as workflows on Amazon SWF. It is available for the Java and Ruby programming languages, and it provides classes that simplify writing complex distributed programs.

With the AWS Flow Framework, you use preconfigured types to map the definition of your workflow directly to methods in your program.

The AWS Flow Framework supports standard object-oriented concepts, such as exception-based error handling, which makes it easier to implement complex workflows. Programs written with the AWS Flow Framework can be created, executed, and debugged entirely within your preferred editor or IDE. For more information, see the AWS Flow Framework website.

Here are links to the AWS Flow Framework documentation:

- AWS Flow Framework for Java Developer Guide
- AWS Flow Framework for Java Reference
- AWS Flow Framework for Ruby Developer Guide
- AWS Flow Framework for Ruby API Reference

AWS Flow Framework Sample Code

In addition to the code snippets that appear in the AWS Flow Framework documentation, you can obtain full, downloadable code samples at the following locations:

- AWS Flow Framework Recipes
- AWS Flow Framework Samples for Amazon SWF

Development Environments

You will need to set up a development environment appropriate to the programming language that you will use. For example, if you intend to develop for Amazon SWF with Java, you will need to install a Java development environment, such as the AWS SDK for Java, on each of your development workstations. If you use the Eclipse IDE for Java development, you might consider also installing the AWS Toolkit for Eclipse. The Toolkit is an Eclipse plug-in that adds features that are helpful for AWS development.

If your programming language requires a run-time environment, you need to set up that environment on each computer on which these processes run.

Endpoints

To reduce latency and to store data in a location that meets your requirements, Amazon SWF provides endpoints in different regions.

Each endpoint in Amazon SWF is completely independent; any domains, workflows and activities you have registered in one region do not share any data or attributes with those in another. In other words, when you register an Amazon SWF domain, workflow or activity, it exists *only within the region you registered it in*. For example, you could register a domain named **`SWF-Flows-1`** in two different regions, but they will share no data or attributes with each other—each acts as a completely independent domain.

For a list of Amazon SWF endpoints, see Regions and Endpoints.

Tutorial: A Subscription Workflow with Amazon SWF and Amazon SNS

This section provides a tutorial that describes how to create an Amazon SWF workflow application that consists of a set of four activities that operate sequentially. It also covers:

- Setting *default* and *execution-time* workflow and activity options.
- Polling Amazon SWF for decision and activity tasks.
- Passing data between the activities and the workflow with Amazon SWF.
- Waiting for *human tasks* and reporting heartbeats to Amazon SWF from an activity task.
- Using Amazon SNS to create a topic, subscribe a user to it, and publish messages to subscribed endpoints.

You can use Amazon Simple Workflow Service (Amazon SWF) and Amazon Simple Notification Service (Amazon SNS) together to emulate a "human task" workflow—one in which a human worker is required to perform some action and then communicate with Amazon SWF to launch the next activity in the workflow.

Because Amazon SWF is a cloud-based web service, communication with Amazon SWF can originate from anywhere a connection to the Internet is available. In this case, we will use Amazon SNS to communicate with the user by either email, an SMS text message, or both.

This tutorial uses the AWS SDK for Ruby to access Amazon SWF and Amazon SNS, but there are many development options available, including the AWS Flow Framework for Ruby, which provides easier coordination and communication with Amazon SWF.

For a complete list of Amazon SWF development options, see Development Options (p. 6).

In this section:

About the Workflow

The workflow that we will be developing consists of four major steps:

1. Get a subscription address (email or SMS) from the user.
2. Create an SNS topic and subscribe the provided endpoints to the topic.
3. Wait for the user to confirm the subscription.
4. If the user confirms, publish a congratulatory message to the topic.

These steps include activities that are completely automated (steps 2 and 4), and others that require the workflow to wait for a human to provide some data to the activity before the workflow can progress (steps 1 and 3).

Because each step relies on data that is generated by the previous step (you must have an endpoint before subscribing it to a topic, and you must have a topic subscription before you can wait for confirmation, etc.) This tutorial will also cover how to provide activity results upon completion, and how to pass input to a task that is being scheduled. Amazon SWF handles coordination and delivery of information between the activities and the workflow, and vice-versa.

We're also using both keyboard input and Amazon SNS to handle communication between Amazon SWF and the human who is providing data to the workflow. In practice, you can use many different techniques to communicate with human users, but Amazon SNS provides a very easy way to use email or text messages to notify the user about events in the workflow.

Prerequisites

To follow along with this tutorial, you will need the following:

- Amazon Web Services (AWS) account
- Ruby interpreter
- AWS SDK for Ruby

If you already have these set up, you're ready to continue. If you don't want to run the example, you can still follow the tutorial—much of the content in this tutorial applies to using Amazon SWF and Amazon SNS regardless of what development options (p. 6) you are using.

Download the Source Code

You can download the complete source code for this tutorial from:
https://s3.amazonaws.com/codesamples/ruby/swf_sns_sample.zip

Tip
Even if you intend to type in (or cut and paste) the code from this tutorial directly into your own source files, having the downloaded source code available to compare your own code with can help identify and solve issues if you run into any along the way.

Tutorial Steps

This tutorial is divided into the following steps:

Subscription Workflow Tutorial Part 1: Using Amazon SWF with the AWS SDK for Ruby

Topics

Include the AWS SDK for Ruby

Begin by creating a file called `utils.rb`. The code in this file will obtain, or create if necessary, the Amazon SWF domain used by both the workflow and activities code and will provide a place to put code that is common to all of our classes.

First, we need to include the `aws-sdk` library in our code, so that we can use the features provided by the SDK for Ruby.

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

This gives us access to the AWS namespace, which provides the ability to set global session-related values, such as your AWS credentials and region, and also provides access to the AWS service APIs.

Configuring the AWS Session

We'll configure the AWS Session by setting our AWS credentials (which are needed for accessing AWS services) and the AWS region to use.

There are a number of ways to set AWS credentials in the AWS SDK for Ruby: by setting them in environment variables (AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY) or by setting them with AWS.config. We'll use the latter method, loading them from a YAML configuration file, called `aws-config.txt`, that looks like this.

```
---
:access_key_id: REPLACE_WITH_ACCESS_KEY_ID
:secret_access_key: REPLACE_WITH_SECRET_ACCESS_KEY
```

Create this file now, replacing the strings beginning with *REPLACE_WITH_* with your AWS access key ID and secret access key. For information about your AWS access keys, see How Do I Get Security Credentials? in the *Amazon Web Services General Reference*.

We also need to set the AWS region to use. Because we'll be using the Short Message Service (SMS) to send text messages to the user's phone with Amazon SNS, we need to make sure that we're using region **us-east-1**. *It is currently the only region that supports SMS*.

> **Note**
> If you don't have access to **us-east-1**, or don't care about running the demo with SMS messaging enabled, feel free to use any region you wish to. You can remove the SMS functionality from the sample and use email as the sole endpoint to subscribe to the Amazon SNS topic.
> For more information about sending SMS messages, see Sending and Receiving SMS Notifications Using Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

We'll now add some code to `utils.rb` to load the config file, get the user's credentials, then provide both the credentials and region to AWS.config.

```
# Load the user's credentials from a file, if it exists.
begin
  config_file = File.open('aws-config.txt') { |f| f.read }
rescue
  puts "No config file! Hope you set your AWS creds in the environment..."
end

if config_file.nil?
  options = { }
else
  options = YAML.load(config_file)
end

# SMS Messaging (which can be used by Amazon SNS) is available only in the
# `us-east-1` region.
$SMS_REGION = 'us-east-1'
options[:region] = $SMS_REGION

# Now, set the options
AWS.config(options)
```

Registering an Amazon SWF Domain

To use Amazon SWF, you need to set up a *domain*: a named entity that will hold your workflows and activities. You can have many Amazon SWF domains registered, but they must all have unique names within your AWS account, and workflows cannot interact across domains: All of the workflows and activities for your application must be in the same domain to interact with one another.

Because we'll be using the same domain throughout our application, we'll create a function in `utils.rb` called `init_domain`, that will retrieve the Amazon SWF domain named *SWFSampleDomain*.

Once you have registered a domain, you can reuse it for many workflow executions. However, *it is an error to try to register a domain that already exists*, so our code will first check to see if the domain exists, and will use the existing domain if it can be found. If the domain can't be found, we'll create it.

To work with Amazon SWF domains in the SDK for Ruby, use AWS::SimpleWorkflow.domains, which returns a DomainCollection that can be used to both enumerate and register domains:

- To check to see if a domain is already registered, you can look at the list provided by AWS::Simpleworkflow.domains.registered.
- To register a new domain, use AWS::Simpleworkflow.domains.register.

Here is the code for `init_domain` in `utils.rb`.

```
# Registers the domain that the workflow will run in.
def init_domain
  domain_name = 'SWFSampleDomain'
  domain = nil
  swf = AWS::SimpleWorkflow.new

  # First, check to see if the domain already exists and is registered.
  swf.domains.registered.each do | d |
    if(d.name == domain_name)
      domain = d
      break
    end
  end

  if domain.nil?
    # Register the domain for one day.
    domain = swf.domains.create(
      domain_name, 1, { :description => "#{domain_name} domain" })
  end

  return domain
end
```

Next Steps

That's it for `utils.rb`. Next, we'll create the workflow and starter code in Part 2: Implementing the Workflow (p. 13).

Subscription Workflow Tutorial Part 2: Implementing the Workflow

Up until now, our code has been pretty generic. This is the part where we begin to really define what our workflow does, and what activities we'll need to implement it.

Topics

Designing the Workflow

If you recall, the initial idea for this workflow consisted of the following steps:

1. Get a subscription address (email or SMS) from the user.

2. Create an SNS topic and subscribe the provided endpoints to the topic.
3. Wait for the user to confirm the subscription.
4. If the user confirms, publish a congratulatory message to the topic.

We can think of each step in our workflow as an *activity* that it must perform. Our *workflow* is responsible for scheduling each activity at the appropriate time, and coordinating data transfer between activities.

For this workflow, we'll create a separate activity for each of these steps, naming them descriptively:

1. get_contact_activity
2. subscribe_topic_activity
3. wait_for_confirmation_activity
4. send_result_activity

These activities will be executed in order, and data from each step will be used in the subsequent step.

We could design our application so that all of the code exists in one source file, but this runs contrary to the way that Amazon SWF was designed. It is designed for workflows that can span the entire Internet in scope, so let's at least break the application up into two separate executables:

- `swf_sns_workflow.rb` - Contains the workflow and workflow starter.
- `swf_sns_activities.rb` - Contains the activities and activities starter.

The workflow and activity implementations can be run in separate windows, separate computers, or even different parts of the world. Because Amazon SWF is keeping track of the details of your workflows and activities, your workflow can coordinate scheduling and data transfer of your activities no matter where they are running.

Setting up our Workflow Code

We'll begin by creating a file called `swf_sns_workflow.rb`. In this file, declare a class called **SampleWorkflow**. Here is the class declaration and its constructor, the `initialize` method.

```
require_relative 'utils.rb'

# SampleWorkflow - the main workflow for the SWF/SNS Sample
#
# See the file called `README.md` for a description of what this file does.
class SampleWorkflow

  attr_accessor :name

  def initialize(task_list)

    # the domain to look for decision tasks in.
    @domain = init_domain

    # the task list is used to poll for decision tasks.
    @task_list = task_list

    # The list of activities to run, in order. These name/version hashes can
be
    # passed directly to AWS::SimpleWorkflow::DecisionTask#schedule_activ
```

```
ity_task.
    @activity_list = [
      { :name => 'get_contact_activity', :version => 'v1' },
      { :name => 'subscribe_topic_activity', :version => 'v1' },
      { :name => 'wait_for_confirmation_activity', :version => 'v1' },
      { :name => 'send_result_activity', :version => 'v1' },
    ].reverse! # reverse the order... we're treating this like a stack.

    register_workflow
  end
```

As you can see, we are keeping the following class instance data:

- `domain` - The domain name retrieved from `init_domain` in `utils.rb`.
- `task_list` - The task list passed in to `initialize`.
- `activity_list` - The activity list, which has the names and versions of the activities we'll run.

The domain name, activity name, and activity version are enough for Amazon SWF to positively identify an activity type, so that is all of the data we need to keep about our activities in order to schedule them.

The task list will be used by the workflow's *decider* code to poll for decision tasks and schedule activities.

At the end of this function, we call a method we haven't yet defined: `register_workflow`. We'll define this method next.

Registering the Workflow

To use a workflow type, we must first register it. Like an activity type, a workflow type is identified by its domain, name, and version. Also, like both domains and activity types, you cannot re-register an existing workflow type. If you need to change anything about a workflow type, you must provide it with a new version, which essentially creates a new type.

Here is the code for `register_workflow`, which is used to either retrieve the existing workflow type we registered on a previous run or to register the workflow if it has not yet been registered.

```
  # Registers the workflow
  def register_workflow
    workflow_name = 'swf-sns-workflow'
    @workflow_type = nil

    # a default value...
    workflow_version = '1'

    # Check to see if this workflow type already exists. If so, use it.
    @domain.workflow_types.each do | a |
      if (a.name == workflow_name) && (a.version == workflow_version)
        @workflow_type = a
      end
    end

    if @workflow_type.nil?
      options =  {
        :default_child_policy => :terminate,
        :default_task_start_to_close_timeout => 3600,
```

```
        :default_execution_start_to_close_timeout => 24 * 3600 }

      puts "registering workflow: #{workflow_name}, #{workflow_version}, #{op
tions.inspect}"
      @workflow_type = @domain.workflow_types.register(workflow_name, work
flow_version, options)
    end

    puts "** registered workflow: #{workflow_name}"
  end
```

First, we check to see if the workflow name and version is already registered by iterating through the domain's workflow_types collection. If we find a match, we'll use the workflow type that was already registered.

If we don't find a match, then a new workflow type is registered (by calling register on the same `workflow_types` collection that we were searching for the workflow in) with the name 'swf-sns-workflow', version '1', and the following options.

```
      options =  {
        :default_task_start_to_close_timeout => 3600,
        :default_execution_start_to_close_timeout => 24 * 3600 }
```

Options passed in during registration are used to set *default behavior* for our workflow type, so we don't need to set these values every time we start a new workflow execution.

Here, we just set some timeout values: the maximum time it can take from the time a task starts to when it closes (one hour), and the maximum time it can take for the workflow execution to complete (24 hours). If either of these times are exceeded, the task or workflow will timeout.

For more information about timeout values, see Timeout Types (p. 118).

Polling for Decisions

At the heart of every workflow execution there is a *decider*. The decider's responsibility is for managing the execution of the workflow itself. The decider receives *decision tasks* and responds to them, either by scheduling new activities, cancelling and restarting activities, or by setting the state of the workflow execution as complete, cancelled, or failed.

The decider uses the workflow execution's *task list* name to receive decision tasks to respond to. To poll for decision tasks, call poll on the domain's decision_tasks collection to loop over available decision tasks. You can then check for new events in the decision task by iterating over its new_events collection.

The returned events are AWS::SimpleWorkflow::HistoryEvent objects, and you can get the type of the event by using the returned event's event_type member. For a list and description of history event types, see HistoryEvent in the *Amazon Simple Workflow Service API Reference*.

Here is the beginning of the decision task poller's logic. A new method in our workflow class called `poll_for_decisions`.

```
def poll_for_decisions
  # first, poll for decision tasks...
  @domain.decision_tasks.poll(@task_list) do | task |
```

```
      task.new_events.each do | event |
        case event.event_type
```

We'll now branch the execution of our decider based on the `event_type` that is received. The first one we are likely to receive is **WorkflowExecutionStarted**. When this event is received, it means that Amazon SWF is signaling to your decider that it should begin the workflow execution. We'll begin by scheduling the first activity by calling schedule_activity_task on the task we received while polling.

We'll pass it the first activity we declared in our activity list, which, because we reversed the list so we can use it like a stack, occupies the `last` position on the list. The "activities" we defined are just maps consisting of a name and version number, but this is all that Amazon SWF needs to identify the activity for scheduling, assuming that the activity has already been registered.

```
          when 'WorkflowExecutionStarted'
            # schedule the last activity on the (reversed, remember?) list to
            # begin the workflow.
            puts "** scheduling activity task: #{@activity_list.last[:name]}"

            task.schedule_activity_task( @activity_list.last,
              { :task_list => "#{@task_list}-activities" } )
```

When we schedule an activity, Amazon SWF sends an *activity task* to the activity task list that we pass in while scheduling it, signaling the task to begin. We'll deal with activity tasks in Part 3: Implementing the Activities (p. 20), but it is worth noting that we don't execute the task here. We only tell Amazon SWF that it should be *scheduled*.

The next activity that we'll need to address is the **ActivityTaskCompleted** event, which occurs when Amazon SWF has received an activity completed response from an activity task.

```
          when 'ActivityTaskCompleted'
            # we are running the activities in strict sequential order, and
            # using the results of the previous activity as input for the next

            # activity.
            last_activity = @activity_list.pop

            if(@activity_list.empty?)
              puts "!! All activities complete! Sending complete_workflow_exe
cution..."
              task.complete_workflow_execution
              return true;
            else
              # schedule the next activity, passing any results from the
              # previous activity. Results will be received in the activity
              # task.
             puts "** scheduling activity task: #{@activity_list.last[:name]}"

              if event.attributes.has_key?('result')
                task.schedule_activity_task(
                  @activity_list.last,
                  { :input => event.attributes[:result],
                    :task_list => "#{@task_list}-activities" } )
              else
                task.schedule_activity_task(
```

```
              @activity_list.last, { :task_list => "#{@task_list}-activities"
} )
            end
          end
```

Since we are executing our tasks in a linear fashion, and only one activity is executing at once, we'll take this opportunity to pop the completed task from the `activity_list` stack. If this results in an empty list, then we know that our workflow is complete. In this case, we signal to Amazon SWF that our workflow is complete by calling complete_workflow_execution on the task.

In the event that the list still has entries, we'll schedule the next activity on the list (again, in the last position). This time, however, we'll look to see if the previous activity returned any result data to Amazon SWF upon completion, which is provided to the workflow in the event's attributes, in the optional `result` key. If the activity generated a result, we'll pass it as the `input` option to the next scheduled activity, along with the activity task list.

By retrieving the `result` values of completed activities, and by setting the `input` values of scheduled activities, we can pass data from one activity to the next, or we can use data from an activity to change behavior in our decider based on the results from an activity.

For the purposes of this tutorial, these two event types are the most important in defining the behavior of our workflow. However, an activity can generate events other than **ActivityTaskCompleted**. We'll wrap up our decider code by providing demonstration handler code for the **ActivityTaskTimedOut** and **ActivityTaskFailed** events, and for the **WorkflowExecutionCompleted** event, which will be generated when Amazon SWF processes the `complete_workflow_execution` call that we make when we run out of activities to run.

```
        when 'ActivityTaskTimedOut'
          puts "!! Failing workflow execution! (timed out activity)"
          task.fail_workflow_execution
          return false

        when 'ActivityTaskFailed'
          puts "!! Failing workflow execution! (failed activity)"
          task.fail_workflow_execution
          return false

        when 'WorkflowExecutionCompleted'
          puts "## Yesss, workflow execution completed!"
          task.workflow_execution.terminate
          return false
      end
    end
  end
end
```

Starting the Workflow Execution

Before any decision tasks will be generated for the workflow to poll for, we need to start the workflow execution.

To start the workflow execution, call start_execution on your registered workflow type (AWS::SimpleWorkflow::WorkflowType). We'll define a small wrapper around this to make use of the `workflow_type` instance member that we retrieved in the class constructor.

```
  def start_execution
    workflow_execution = @workflow_type.start_execution( {
      :task_list => @task_list } )
    poll_for_decisions
  end
end
```

Once the workflow is executing, decision events will begin to appear on the workflow's task list, which is passed as a workflow execution option in start_execution.

Unlike options that are provided when the workflow type is registered, options that are passed to `start_execution` are not considered to be part of the workflow type. You are free to change these per workflow execution without changing the workflow's version.

Since we'd like the workflow to begin executing when we run the file, add some code that instantiates the class and then calls the `start_execution` method that we just defined.

```
if __FILE__ == $0
  require 'securerandom'

  # Use a different task list name every time we start a new workflow execution.

  #
  # This avoids issues if our pollers re-start before SWF considers them closed,

  # causing the pollers to get events from previously-run executions.
  task_list = SecureRandom.uuid

  # Let the user start the activity worker first...

  puts ""
  puts "Amazon SWF Example"
  puts "------------------"
  puts ""
  puts "Start the activity worker, preferably in a separate command-line window,
 with"
  puts "the following command:"
  puts ""
  puts "> ruby swf_sns_activities.rb #{task_list}-activities"
  puts ""
  puts "You can copy & paste it if you like, just don't copy the '>' character."

  puts ""
  puts "Press return when you're ready..."

  i = gets

  # Now, start the workflow.

  puts "Starting workflow execution."
  sample_workflow = SampleWorkflow.new(task_list)
  sample_workflow.start_execution
end
```

To avoid any task list naming conflicts, we'll use `SecureRandom.uuid` to generate a random UUID that we can use as the task list name, guaranteeing that a different task list name is used for each workflow execution.

> **Note**
> Task lists are used to record events about a workflow execution, so if you use the same task list for multiple executions of the same workflow type, you might get events that were generated during a previous execution, especially if you are running them in near succession to each other, which is often the case when trying out new code or running tests.

To avoid the issue of having to deal with artifacts from previous executions, we can use a new task list for each execution, specifying it when we begin the workflow execution.

There is also a bit of code here to provide instructions for the person running it (probably you), and to provide the "activity" version of the task list. The decider uses this task list name to schedule activities for the workflow, and the activities implementation will listen for activity events on this task list name to know when to begin the scheduled activities and to provide updates about the activity execution.

The code also waits for the user to start running the activities starter *before* it starts the workflow execution, so the activities starter will be ready to respond when activity tasks begin appearing on the provided task list.

Next Steps

We've completed the workflow implementation. Next, we'll define the activities and an activities starter, in Part 3: Implementing the Activities (p. 20).

Subscription Workflow Tutorial Part 3: Implementing the Activities

We'll now implement each of the activities in our workflow, beginning with a base class that provides some common features for the activity code.

Topics

Defining a Basic Activity Type

When designing the workflow, we identified the following activities:

- `get_contact_activity`
- `subscribe_topic_activity`
- `wait_for_confirmation_activity`
- `send_result_activity`

We'll implement each of these activities now. Since our activities will share some features, let's do a little groundwork and create some common code they can share. We'll call it **BasicActivity**, and define it in a new file called `basic_activity.rb`.

As with the other source files, we'll include `utils.rb` to access the `init_domain` function to set up the sample domain.

```
require_relative 'utils.rb'
```

Next, well declare the basic activity class and some common data that we'll be interested in for each activity. We'll save the activity's AWS::SimpleWorkflow::ActivityType instance, its *name*, and provide a special data member used to store the activity's *results*.

```
class BasicActivity

  attr_accessor :activity_type
  attr_accessor :name
  attr_accessor :results
```

These attributes access instance data that's defined in the class' `initialize` method, which takes an activity *name*, and an optional *version* and map of *options* to be used when registering the activity with Amazon SWF.

```
    def initialize(name, version = 'v1', options = nil)

      @activity_type = nil
      @name = name
      @results = nil

      # get the domain to use for activity tasks.
      @domain = init_domain

      # Check to see if this activity type already exists.
      @domain.activity_types.each do | a |
        if (a.name == @name) && (a.version == version)
          @activity_type = a
        end
      end

      if @activity_type.nil?
        # If no options were specified, use some reasonable defaults.
        if options.nil?
          options = {
            # All timeouts are in seconds.
            :default_task_heartbeat_timeout => 900,
            :default_task_schedule_to_start_timeout => 120,
            :default_task_schedule_to_close_timeout => 3800,
            :default_task_start_to_close_timeout => 3600 }
        end
        @activity_type = @domain.activity_types.register(@name, version, op
tions)
      end
    end
```

As with workflow type registration, if an activity type is already registered, we can retrieve it by looking at the domain's activity_types collection. If the activity can't be found, it will be registered.

Also, as with workflow types, you can set *default options* that are stored with your activity type when you register it.

The last thing our basic activity gets is a consistent way to run it. We'll define a `do_activity` method that takes an activity task. As shown, we can use the passed-in activity task to receive data via its `input` instance attribute.

```
  def do_activity(task)
    @results = task.input # may be nil
    return true
  end
end
```

That wraps up the **BasicActivity** class. Now we'll use it to make defining our activities simple and consistent.

Defining GetContactActivity

The first activity that is run during a workflow execution is `get_contact_activity`, which retrieves the user's Amazon SNS topic subscription information.

Create a new file called `get_contact_activity.rb`, and require both `yaml`, which we'll use to prepare a string for passing to Amazon SWF, and `basic_activity.rb`, which we'll use as the basis for this **GetContactActivity** class.

```
require 'yaml'
require_relative 'basic_activity.rb'

# **GetContactActivity** provides a prompt for the user to enter contact
# information. When the user successfully enters contact information, the
# activity is complete.
class GetContactActivity < BasicActivity
```

Since we put the activity registration code in **BasicActivity**, the `initialize` method for **GetContactActivity** is pretty simple. We simply call the base class constructor with the activity name, `get_contact_activity`. This is all that is required to register our activity.

```
  # initialize the activity
  def initialize
    super('get_contact_activity')
  end
```

We'll now define the `do_activity` method, which prompts for the user's email and/or phone number.

```
  def do_activity(task)
    puts ""
```

```
      puts "Please enter either an email address or SMS message (mobile phone)
 number to"
      puts "receive SNS notifications. You can also enter both to use both
address types."
      puts ""
      puts "If you enter a phone number, it must be able to receive SMS mes
sages, and must"
      puts "be 11 digits (such as 12065550101 to represent the number 1-206-
555-0101)."

      input_confirmed = false
      while !input_confirmed
        puts ""
        print "Email: "
        email = $stdin.gets.strip

        print "Phone: "
        phone = $stdin.gets.strip

        puts ""
        if (email == '') && (phone == '')
          print "You provided no subscription information. Quit? (y/n)"
           confirmation = $stdin.gets.strip.downcase
           if confirmation == 'y'
             return false
           end
        else
           puts "You entered:"
           puts "  email: #{email}"
           puts "  phone: #{phone}"
           print "\nIs this correct? (y/n): "
           confirmation = $stdin.gets.strip.downcase
           if confirmation == 'y'
             input_confirmed = true
           end
        end
      end

      # make sure that @results is a single string. YAML makes this easy.
      @results = { :email => email, :sms => phone }.to_yaml
      return true
    end
  end
```

At the end of `do_activity`, we take the email and phone number retrieved from the user, place it in a map and then use `to_yaml` to convert the entire map to a YAML string. There's an important reason for this: any results that you pass to Amazon SWF when you complete an activity must be *string data only*. Ruby's ability to easily convert objects to YAML strings and then back again into objects is, thankfully, well-suited for this purpose.

That's the end of the `get_contact_activity` implementation. This data will be used next in the `subscribe_topic_activity` implementation.

Defining SubscribeTopicActivity

We'll now delve into Amazon SNS and create an activity that uses the information generated by `get_contact_activity` to subscribe the user to an Amazon SNS topic.

Create a new file called `subscribe_topic_activity.rb`, add the same requirements that we used for `get_contact_activity`, declare your class, and provide its `initialize` method.

```
  require 'yaml'
  require_relative 'basic_activity.rb'

 # **SubscribeTopicActivity** sends an SMS / email message to the user, asking
for
 # confirmation.  When this action has been taken, the activity is complete.

  class SubscribeTopicActivity < BasicActivity

    def initialize
      super('subscribe_topic_activity')
    end
```

Now that we have the code in place to get the activity set up and registered, add some code to create an Amazon SNS topic. To do so, we'll use the AWS::SNS::Client object's create_topic method.

Add the `create_topic` method to your class, which takes a passed-in Amazon SNS client object.

```
    def create_topic(sns_client)
      topic_arn = sns_client.create_topic(:name => 'SWF_Sample_Topic')[:top
ic_arn]

      if topic_arn != nil
        # For an SMS notification, setting `DisplayName` is *required*. Note
that
        # only the *first 10 characters* of the DisplayName will be shown on
the
        # SMS message sent to the user, so choose your DisplayName wisely!
        sns_client.set_topic_attributes( {
          :topic_arn => topic_arn,
          :attribute_name => 'DisplayName',
          :attribute_value => 'SWFSample' } )
      else
        @results = {
          :reason => "Couldn't create SNS topic", :detail => "" }.to_yaml
        return nil
      end

      return topic_arn
    end
```

Once we have the topic's Amazon Resource Name (ARN), we can use it with the Amazon SNS client's set_topic_attributes method to set the topic's *DisplayName*, which is required for sending SMS messages with Amazon SNS.

Lastly, we'll define the `do_activity` method. We'll start by collecting any data that was passed via the `input` option when the activity was scheduled. As previously mentioned, this must be passed as a string, which we created using `to_yaml`. When retrieving it, we'll use `YAML.load` to turn the data into Ruby objects.

Here's the beginning of `do_activity`, in which we retrieve the input data.

```
    def do_activity(task)
      activity_data = {
        :topic_arn => nil,
        :email => { :endpoint => nil, :subscription_arn => nil },
        :sms => { :endpoint => nil, :subscription_arn => nil },
      }

      if task.input != nil
        input = YAML.load(task.input)
        activity_data[:email][:endpoint] = input[:email]
        activity_data[:sms][:endpoint] = input[:sms]
      else
        @results = { :reason => "Didn't receive any input!", :detail => ""
}.to_yaml
        puts("  #{@results.inspect}")
        return false
      end

      # Create an SNS client. This is used to interact with the service. Set
the
     # region to $SMS_REGION, which is a region that supports SMS notifications

      # (defined in the file `swf_sns_utils.rb`).
      sns_client = AWS::SNS::Client.new(
        :config => AWS.config.with(:region => $SMS_REGION))
```

If we didn't receive any input, there isn't much to do, so we'll just fail the activity.

Assuming that everything is fine, however, we'll continue filling in our `do_activity` method, get an Amazon SNS client with the AWS SDK for Ruby, and pass it to our `create_topic` method to create the Amazon SNS topic.

```
      # Create the topic and get the ARN
      activity_data[:topic_arn] = create_topic(sns_client)

      if activity_data[:topic_arn].nil?
        return false
      end
```

There are a couple of things worth noting here:

- We use AWS.config.with to set the region for our Amazon SNS client. Because we want to send SMS messages, we use the SMS-enabled region that we declared in `utils.rb`.
- We save the topic's ARN in our `activity_data` map. This is part of the data that will be passed to the *next* activity in our workflow.

Finally, this activity subscribes the user to the Amazon SNS topic, using the passed-in endpoints (email and SMS). We don't require the user to enter *both* endpoints, but we do need at least one.

```
      # Subscribe the user to the topic, using either or both endpoints.
      [:email, :sms].each do | x |
        ep = activity_data[x][:endpoint]
        # don't try to subscribe an empty endpoint
        if (ep != nil && ep != "")
          response = sns_client.subscribe( {
            :topic_arn => activity_data[:topic_arn],
            :protocol => x.to_s, :endpoint => ep } )
          activity_data[x][:subscription_arn] = response[:subscription_arn]
        end
      end
```

AWS::SNS::Client.subscribe takes the topic ARN, the *protocol* (which, cleverly, we disguised as the `activity_data` map key for the corresponding endpoint).

Finally, we re-package the information for the next activity in YAML format, so that we can send it back to Amazon SWF.

```
      # if at least one subscription arn is set, consider this a success.
      if (activity_data[:email][:subscription_arn] != nil) or (activ
ity_data[:sms][:subscription_arn] != nil)
        @results = activity_data.to_yaml
      else
        @results = { :reason => "Couldn't subscribe to SNS topic", :detail =>
 "" }.to_yaml
        puts("  #{@results.inspect}")
        return false
      end
      return true
    end
  end
```

That completes the implementation of the `subscribe_topic_activity`. Next, we'll define `wait_for_confirmation_activity`.

Defining WaitForConfirmationActivity

Once a user is subscribed to an Amazon SNS topic, he or she will still need to confirm the subscription request. In this case, we'll be waiting for the user to confirm by either email or an SMS message.

The activity that waits for the user to confirm the subscription is called `wait_for_confirmation_activity`, and we'll define it here. To begin, create a new file called `wait_for_confirmation_activity.rb` and set it up as we've set up the previous activities.

```
  require 'yaml'
  require_relative 'basic_activity.rb'

  # **WaitForConfirmationActivity** waits for the user to confirm the SNS
  # subscription.  When this action has been taken, the activity is complete.
```

```
It
  # might also time out...
  class WaitForConfirmationActivity < BasicActivity

    # Initialize the class
    def initialize
      super('wait_for_confirmation_activity')
    end
```

Next, we'll begin defining the `do_activity` method and retrieve any input data into a local variable called `subscription_data`.

```
    def do_activity(task)
      if task.input.nil?
        @results = { :reason => "Didn't receive any input!", :detail => ""
}.to_yaml
        return false
      end

      subscription_data = YAML.load(task.input)
```

Now that we have the topic ARN, we can retrieve the topic by creating a new instance of AWS::SNS::Topic and pass it the ARN.

```
      topic = AWS::SNS::Topic.new(subscription_data[:topic_arn])

      if topic.nil?
        @results = {
          :reason => "Couldn't get SWF topic ARN",
          :detail => "Topic ARN: #{topic.arn}" }.to_yaml
        return false
      end
```

Now, we'll check the topic to see if the user has confirmed the subscription using one of the endpoints. We'll only require that one endpoint has been confirmed to consider the activity a success.

An Amazon SNS topic maintains a list of the subscriptions for that topic, and we can check whether or not the user has confirmed a particular subscription by checking to see if the subscription's ARN is set to anything other than `PendingConfirmation`.

```
      # loop until we get some indication that a subscription was confirmed.
      subscription_confirmed = false
      while(!subscription_confirmed)
        topic.subscriptions.each do | sub |
         if subscription_data[sub.protocol.to_sym][:endpoint] == sub.endpoint

            # this is one of the endpoints we're interested in. Is it sub
scribed?
            if sub.arn != 'PendingConfirmation'
              subscription_data[sub.protocol.to_sym][:subscription_arn] =
sub.arn
              puts "Topic subscription confirmed for (#{sub.protocol}:
```

```
#{sub.endpoint})"
               @results = subscription_data.to_yaml
               return true
             else
               puts "Topic subscription still pending for (#{sub.protocol}:
#{sub.endpoint})"
             end
           end
         end
```

If we get an ARN for the subscription, we'll save it in the activity's result data, convert it to YAML, and return true from `do_activity`, which signals that the activity completed successfully.

Since waiting for a subscription to be confirmed might take a while, we'll occasionally call `record_heartbeat` on the activity task. This signals to Amazon SWF that the activity is still processing, and can also be used to provide updates about the progress of the activity (if you are doing something, like processing files, that you can report progress for).

```
         task.record_heartbeat!(
           { :details => "#{topic.num_subscriptions_confirmed} confirmed,
#{topic.num_subscriptions_pending} pending" })
         # sleep a bit.
         sleep(4.0)
       end
```

This ends our `while` loop. If we somehow get out of the while loop without success, we'll report failure and finish the `do_activity` method.

```
       if (subscription_confirmed == false)
         @results = {
           :reason => "No subscriptions could be confirmed",
           :detail => "#{topic.num_subscriptions_confirmed} confirmed, #{top
ic.num_subscriptions_pending} pending" }.to_yaml
         return false
       end
     end
   end
```

That ends the implementation of `wait_for_confirmation_activity`. We have only one more activity to define: `send_result_activity`.

Defining SendResultActivity

If the workflow has progressed this far, we've successfully subscribed the user to an Amazon SNS topic and the user has confirmed the subscription.

Our last activity, `send_result_activity`, sends the user a confirmation of the successful topic subscription, using the topic that the user subscribed to and the endpoint that the user confirmed the subscription with.

Create a new file called `send_result_activity.rb` and set it up as we've set up all the activities so far.

```
   require 'yaml'
   require_relative 'basic_activity.rb'

   # **SendResultActivity** sends the result of the activity to the screen,
and, if
   # the user successfully registered using SNS, to the user using the SNS
contact
   # information collected.
   class SendResultActivity < BasicActivity

     def initialize
       super('send_result_activity')
     end
```

Our `do_activity` method begins similarly, as well, getting the input data from the workflow, converting it from YAML, and then using the topic ARN to create an AWS::SNS::Topic instance.

```
     def do_activity(task)
       if task.input.nil?
         @results = { :reason => "Didn't receive any input!", :detail => "" }
         return false
       end

       input = YAML.load(task.input)

       # get the topic, so we publish a message to it.
       topic = AWS::SNS::Topic.new(input[:topic_arn])

       if topic.nil?
         @results = {
           :reason => "Couldn't get SWF topic",
           :detail => "Topic ARN: #{topic.arn}" }
         return false
       end
```

Once we have the topic, we'll publish a message to it (and echo it to the screen, as well).

```
      @results = "Thanks, you've successfully confirmed registration, and your
 workflow is complete!"

       # send the message via SNS, and also print it on the screen.
       topic.publish(@results)
       puts(@results)

       return true
     end
   end
```

Publishing to an Amazon SNS topic sends the message that you supply to *all* of the subscribed and confirmed endpoints that exist for that topic. So, if the user confirmed with *both* an email and an SMS number, he or she will receive two confirmation messages, one at each endpoint.

Next Steps

That completes the implementation of `send_result_activity`. Now, we'll tie all these activities together in an activities application that handles the activity tasks and can launch activities in response, as described in Part 4: Implementing the Activities Task Poller (p. 30).

Subscription Workflow Tutorial Part 4: Implementing the Activities Task Poller

In Amazon SWF, activity tasks for a running workflow execution appear on the *activity task list*, which is provided when you schedule an activity in the workflow.

We'll implement a basic activity poller to handle these tasks for our workflow, and use it to launch our activities when Amazon SWF places a task on the activity task list to start the activity.

To begin, create a new file called `swf_sns_activities.rb`. We'll use it to:

- Instantiate the activity classes that we created.
- Register each activity with Amazon SWF.
- Poll for activities and call `do_activity` for each activity when its name appears on the activity task list.

In `swf_sns_activities.rb`, add the following statements to require each of the activity classes we defined.

```
require_relative 'get_contact_activity.rb'
require_relative 'subscribe_topic_activity.rb'
require_relative 'wait_for_confirmation_activity.rb'
require_relative 'send_result_activity.rb'
```

Now, we'll create the class and provide some initialization code.

```
class ActivitiesPoller

  def initialize(domain, task_list)
    @domain = domain
    @task_list = task_list
    @activities = {}

    # These are the activities we'll run
    activity_list = [
      GetContactActivity,
      SubscribeTopicActivity,
      WaitForConfirmationActivity,
      SendResultActivity ]

    activity_list.each do | activity_class |
      activity_obj = activity_class.new
      puts "** initialized and registered activity: #{activity_obj.name}"
```

```
      # add it to the hash
      @activities[activity_obj.name.to_sym] = activity_obj
    end
  end
```

In addition to saving the passed in *domain* and *task list*, this code instantiates each of the activity classes we created. Because each class registers its associated activity (refer to `basic_activity.rb` if you need to review that code), this is enough to let Amazon SWF know about all of the activities we'll be running.

For each activity instantiated, we store it on a map using the activity name (such as `get_contact_activity`) as the key, so we can easily look these up in the activity poller code, which we'll define next.

Create a new method called `poll_for_activities` and call poll on the activity_tasks held by the domain to get activity tasks.

```
  def poll_for_activities
    @domain.activity_tasks.poll(@task_list) do | task |
      activity_name = task.activity_type.name
```

We can get the activity name from the task's activity_type member. Next, we'll use the activity name associated with this task to look up the class to run `do_activity` on, passing it the task (which includes any input data that should be transferred to the activity).

```
      # find the task on the activities list, and run it.
      if @activities.key?(activity_name.to_sym)
        activity = @activities[activity_name.to_sym]
        puts "** Starting activity task: #{activity_name}"
        if activity.do_activity(task)
          puts "++ Activity task completed: #{activity_name}"
          task.complete!({ :result => activity.results })
          # if this is the final activity, stop polling.
          if activity_name == 'send_result_activity'
             return true
          end
        else
          puts "-- Activity task failed: #{activity_name}"
          task.fail!(
            { :reason => activity.results[:reason],
              :details => activity.results[:detail] } )
        end
      else
        puts "couldn't find key in @activities list: #{activity_name}"
        puts "contents: #{@activities.keys}"
      end
    end
  end
end
```

The code just waits for `do_activity` to complete, and then calls either complete! or fail! on the task based on the return code.

Note
This code exits from the poller once the final activity has been launched, since our poller has completed its mission and has launched all of the activities. In your own Amazon SWF code, if your activities might be run again, you may want to keep the activity poller running indefinitely.

That's the end of the code for our **ActivitiesPoller** class, but we'll add a little more code at the end of the file to allow the user to run it from the command-line.

```
if __FILE__ == $0
  if ARGV.count < 1
    puts "You must supply a task-list name to use!"
    exit
  end
  poller = ActivitiesPoller.new(init_domain, ARGV[0])
  poller.poll_for_activities
  puts "All done!"
end
```

If the user runs the file from the command line (passing it an activity task list as the first argument), this code will instantiate the poller class and start it polling for activities. Once the poller completes (after it has launched the final activity), we just print a message and exit.

That's it for the activities poller. All that's left for you to do is to run the code and see how it works, in Running the Workflow (p. 32).

Subscription Workflow Tutorial: Running the Workflow

Now that you've completed the implementation of your workflow, activities, and the workflow and activity pollers, you're ready to run the workflow.

If you haven't done so already, you'll need to provide your AWS access keys in the `aws-config.txt` file as described in Configuring the AWS Session (p. 11) in Part 1 of the tutorial.

Now, go to your command line and change to the directory where the tutorial source files are located. You should have the following files:

```
.
|-- basic_activity.rb
|-- get_contact_activity.rb
|-- send_result_activity.rb
|-- subscribe_topic_activity.rb
|-- swf_sns_activities.rb
|-- swf_sns_workflow.rb
|-- utils.rb
`-- wait_for_confirmation_activity.rb
```

Now, start the workflow with the following command.

```
ruby swf_sns_workflow.rb
```

This will begin the workflow, and should print out a message with a line that you can copy and paste into a separate command-line window (or even on another computer, if you've copied the tutorial source files onto it).

```
Amazon SWF Example
------------------

Start the activity worker, preferably in a separate command-line window, with
the following command:

> ruby swf_sns_activities.rb 87097e76-7c0c-41c7-817b-92527bb0ea85-activities

You can copy & paste it if you like, just don't copy the '>' character.

Press return when you're ready...
```

The workflow code will wait patiently for you to start the activity poller in a separate window.

Open a new command-line window, change to the directory where the source files are located again, and then use the command provided by the `swf_sns_workflow.rb` file to start the activity poller. For example, if you received the preceding output, you would type (or paste) the following.

```
ruby swf_sns_activities.rb 87097e76-7c0c-41c7-817b-92527bb0ea85-activities
```

Once you begin running your activity poller, it will start to output information about activities registration.

```
** initialized and registered activity: get_contact_activity
** initialized and registered activity: subscribe_topic_activity
** initialized and registered activity: wait_for_confirmation_activity
** initialized and registered activity: send_result_activity
```

You can now return to your original command-line window, and press return to start your workflow execution. It will register the workflow and schedule the first activity.

```
Starting workflow execution.
** registered workflow: swf-sns-workflow
** scheduling activity task: get_contact_activity
```

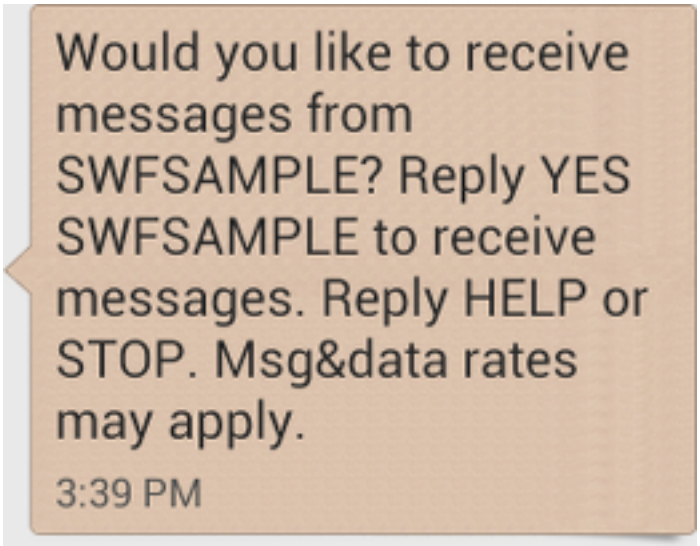
Go back to the other window, where your activity poller is running. You should see the result of the first activity running now, providing a prompt for you to enter your email and/or SMS phone number. Enter either, or both, of these pieces of data, and then confirm your text entry.

```
activity task received: <AWS::SimpleWorkflow::ActivityTask>
** Starting activity task: get_contact_activity

Please enter either an email address or SMS message (mobile phone) number to
receive Amazon SNS notifications. You can also enter both to use both address
types.
```

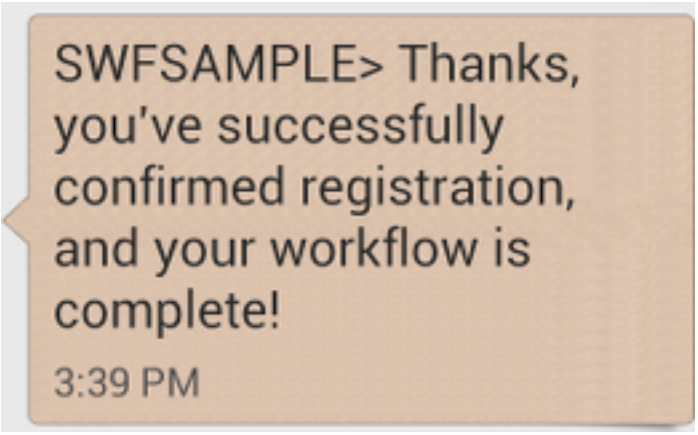
```
If you enter a phone number, it must be able to receive SMS messages, and must
be 11 digits (such as 12065550101 to represent the number 1-206-555-0101).

Email: me@example.com
Phone: 12065550101

You entered:
  email: me@example.com
  phone: 12065550101

Is this correct? (y/n): y
```

Note
The phone number provided is fictitious, and is used only for illustrative purposes. Use your own phone number and email address here!

Soon after entering this information, you should receive an email or text message from Amazon SNS, asking you to confirm your topic subscription. If you entered an SMS number, you will see something like the following appear on your phone.



If you reply to this message with YES, you'll get the response that we provided in `send_result_activity`.



While all of this was happening, did you see what was happening in your command-line window? Both the workflow and activity pollers have been hard at work.

Here's the output from the workflow poller.

```
** scheduling activity task: subscribe_topic_activity
** scheduling activity task: wait_for_confirmation_activity
** scheduling activity task: send_result_activity
!! All activities complete! Sending complete_workflow_execution...
```

Here's the output from the activity poller, which was happening at the same time in another command-line window.

```
++ Activity task completed: get_contact_activity
** Starting activity task: subscribe_topic_activity
++ Activity task completed: subscribe_topic_activity
** Starting activity task: wait_for_confirmation_activity
Topic subscription still pending for (email: me@example.com)
Topic subscription confirmed for (sms: 12065550101)
++ Activity task completed: wait_for_confirmation_activity
** Starting activity task: send_result_activity
Thanks, you've successfully confirmed registration, and your workflow is com
plete!
++ Activity task completed: send_result_activity
All done!
```

Congratulations, your workflow is complete, and so is this tutorial!

You may want to re-run the workflow again to see how timeouts work, or to enter different data. Just remember that once you subscribe to a topic, *you're still subscribed until you unsubscribe*. Re-running the workflow before unsubscribing to topics will probably result in automatic success, since the `wait_for_confirmation_activity` will see that your subscription is already confirmed.

To unsubscribe from the Amazon SNS topic

- Respond in the negative (send `STOP`) to the text message.
- Click the unsubscribe link that you received in your email.

You're now ready to re-subscribe to the topic again.

Where Do I Go from Here?

This tutorial has covered a lot of ground, but there's still much more you can learn about the AWS SDK for Ruby, Amazon SWF, or Amazon SNS. For more information and many more examples, see the official documentation for each:

- AWS SDK for Ruby Documentation
- Amazon Simple Notification Service Documentation
- Amazon Simple Workflow Service Documentation

Basic Concepts in Amazon SWF

The concepts in this chapter provide an overview of the Amazon Simple Workflow Service and describe its major features. While some examples of the use of Amazon SWF are provided in the topics within this chapter, refer to the section titled Using the API (p. 69) for more concrete examples of implementing the features described here.

Topics

Amazon SWF Workflows

Topics

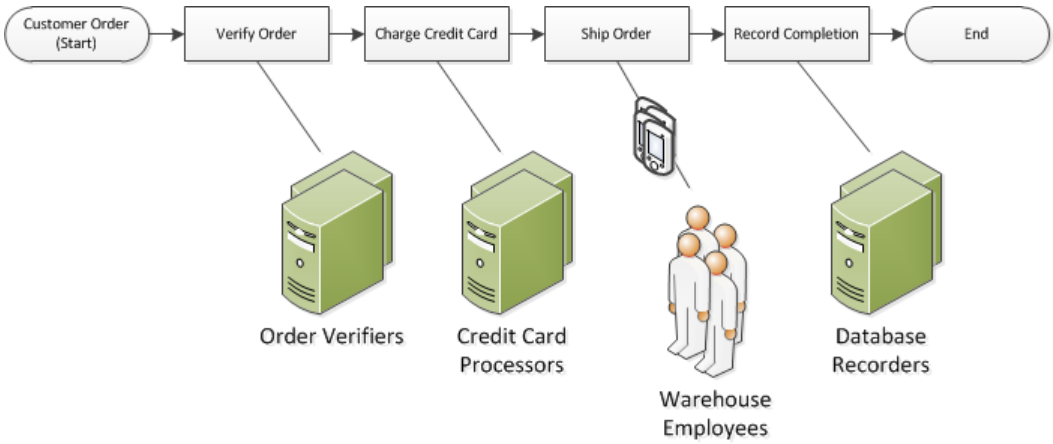
What is a Workflow?

Using the Amazon Simple Workflow Service (Amazon SWF), you can implement distributed, asynchronous applications as *workflows*. Workflows coordinate and manage the execution of activities that can be run asynchronously across multiple computing devices and that can feature both sequential and parallel processing.

When designing a workflow, you analyze your application to identify its component *tasks*. In Amazon SWF, these tasks are represented by *activities*. The order in which activities are performed is determined by the workflow's coordination logic.

A Simple Workflow Example: an E-Commerce Application

For example, the following figure shows a simple e-commerce order-processing workflow involving both people and automated processes.



This workflow starts when a customer places an order. It includes four *tasks*:

1. Verify the order.
2. If the order is valid, charge the customer.
3. If the payment is made, ship the order.
4. If the order is shipped, save the order details.

The tasks in this workflow are *sequential*: an order must be verified before a credit card can be charged; a credit card must be charged successfully before an order can be shipped; and an order must be shipped before it can be recorded. Even so, because Amazon SWF supports distributed processes, these tasks can be carried out in different locations. If the tasks are programmatic in nature, they can also be written in different programming languages or using different tools.

In addition to sequential processing of tasks, Amazon SWF also supports workflows with parallel processing of tasks. Parallel tasks are performed at the same time, and may be carried out independently by different applications or human workers. Your workflow makes decisions about how to proceed once one or more of the parallel tasks have been completed.

Workflow Registration and Execution

After the coordination logic and the activities have been designed, you register these components as workflow and activity types with Amazon SWF. During registration, you specify for each type a name, a version, and some default configuration values.

Only registered workflow and activity types can be used with Amazon SWF. In the e-commerce example, you would register the CustomerOrder workflow type and the VerifyOrder, ChargeCreditCard, ShipOrder, and RecordCompletion activity types.

After registering your workflow type, you can run it as often you like. A *workflow execution* is a running instance of a workflow. In the e-commerce example, a new workflow execution is started with each customer order.

A workflow execution can be started by any process or application, even another workflow execution. In the e-commerce example, what type of application initiates the workflow depends on how the customer places the order. The workflow could be initiated by a web site or mobile application or by a customer service representative using an internal company application.

With Amazon SWF, you can associate an identifier—called a `workflowId`—with your workflow executions, so you can integrate your existing business identifiers into your workflow. In the e-commerce example, each workflow execution might be identified using the customer invoice number.

In addition to the identifier that you provide, Amazon SWF associates a unique system-generated identifier—a `runId`—with each workflow execution. Amazon SWF allows only one workflow execution with this identifier to run at any given time; although you can have multiple workflows executions of the same workflow type, each workflow execution has a distinct `runId`.

See Also

- Workflow History (p. 38)

Amazon SWF Workflow History

The progress of every workflow execution is recorded in its workflow history, which Amazon SWF maintains. The workflow history is a detailed, complete, and consistent record of every event that occurred since the workflow execution started. An event represents a discrete change in your workflow execution's state, such as a new activity being scheduled or a running activity being completed. The workflow history contains every event that causes the execution state of the workflow execution to change, such as scheduled and completed activities, task timeouts, and signals.

Operations that do not change the state of the workflow execution do not typically appear in the workflow history. For example, the workflow history does not show poll attempts or the use of visibility operations.

The workflow history has several key benefits:

- It enables applications to be stateless, because all information about a workflow execution is stored in its workflow history.
- For each workflow execution, the history provides a record of which activities were scheduled, their current status, and their results. The workflow execution uses this information to determine next steps.
- The history provides a detailed audit trail that you can use to monitor running workflow executions and verify completed workflow executions.

The following is a conceptual view of the e-commerce workflow history:

```
Invoice0001

Start Workflow Execution

Schedule Verify Order
Start Verify Order Activity
Complete Verify Order Activity
```

```
Schedule Charge Credit Card
Start Charge Credit Card Activity
Complete Charge Credit Card Activity

Schedule Ship Order
Start Ship Order Activity
```

In the preceding example, the order is waiting to ship. In the following example, the order is complete. Because the workflow history is cumulative, the newer events are appended:

```
Invoice0001

Start Workflow Execution

Schedule Verify Order
Start Verify Order Activity
Complete Verify Order Activity

Schedule Charge Credit Card
Start Charge Credit Card Activity
Complete Charge Credit Card Activity

Schedule Ship Order
Start Ship Order Activity

Complete Ship Order Activity

Schedule Record Order Completion
Start Record Order Completion Activity
Complete Record Order Completion Activity

Close Workflow
```

Programmatically, the events in the workflow execution history are represented as JavaScript Object Notation (JSON) objects. The history itself is a JSON array of these objects. Each event has the following:

- A type, such as WorkflowExecutionStarted or ActivityTaskCompleted
- A timestamp in Unix time format
- An ID that uniquely identifies the event

In addition, each type of event has a distinct set of descriptive attributes that are appropriate to that type. For example, the `ActivityTaskCompleted` event has attributes that contain the IDs for the events that correspond to the time that the activity task was scheduled and when it was started, as well as an attribute that holds result data.

You can obtain a copy of the current state of the workflow execution history by using the GetWorkflowExecutionHistory action. In addition, as part of the interaction between Amazon SWF and the decider for your workflow, the decider periodically receives copies of the history.

Below is a section of an example workflow execution history in JSON format.

```
[  {
      "eventId": 11,
```

```
    "eventTimestamp": 1326671603.102,
    "eventType": "WorkflowExecutionTimedOut",
    "workflowExecutionTimedOutEventAttributes": {
       "childPolicy": "TERMINATE",
       "timeoutType": "START_TO_CLOSE"
    }
 }, {
    "decisionTaskScheduledEventAttributes": {
       "startToCloseTimeout": "600",
       "taskList": {
          "name": "specialTaskList"
       }
    },
    "eventId": 10,
    "eventTimestamp": 1326670566.124,
    "eventType": "DecisionTaskScheduled"
 }, {
    "activityTaskTimedOutEventAttributes": {
       "details": "Waiting for confirmation",
       "scheduledEventId": 8,
       "startedEventId": 0,
       "timeoutType": "SCHEDULE_TO_START"
    },
    "eventId": 9,
    "eventTimestamp": 1326670566.124,
    "eventType": "ActivityTaskTimedOut"
 }, {
    "activityTaskScheduledEventAttributes": {
       "activityId": "verification-27",
       "activityType": {
          "name": "activityVerify",
          "version": "1.0"
       },
       "control": "digital music",
       "decisionTaskCompletedEventId": 7,
       "heartbeatTimeout": "120",
       "input": "5634-0056-4367-0923,12/12,437",
       "scheduleToCloseTimeout": "900",
       "scheduleToStartTimeout": "300",
       "startToCloseTimeout": "600",
       "taskList": {
          "name": "specialTaskList"
       }
    },
    "eventId": 8,
    "eventTimestamp": 1326670266.115,
    "eventType": "ActivityTaskScheduled"
 }, {
    "decisionTaskCompletedEventAttributes": {
       "executionContext": "Black Friday",
       "scheduledEventId": 5,
       "startedEventId": 6
    },
    "eventId": 7,
    "eventTimestamp": 1326670266.103,
    "eventType": "DecisionTaskCompleted"
 }, {
    "decisionTaskStartedEventAttributes": {
```

```
         "identity": "Decider01",
         "scheduledEventId": 5
      },
      "eventId": 6,
      "eventTimestamp": 1326670161.497,
      "eventType": "DecisionTaskStarted"
   }, {
      "decisionTaskScheduledEventAttributes": {
         "startToCloseTimeout": "600",
         "taskList": {
            "name": "specialTaskList"
         }
      },
      "eventId": 5,
      "eventTimestamp": 1326668752.66,
      "eventType": "DecisionTaskScheduled"
   }, {
      "decisionTaskTimedOutEventAttributes": {
         "scheduledEventId": 2,
         "startedEventId": 3,
         "timeoutType": "START_TO_CLOSE"
      },
      "eventId": 4,
      "eventTimestamp": 1326668752.66,
      "eventType": "DecisionTaskTimedOut"
   }, {
      "decisionTaskStartedEventAttributes": {
         "identity": "Decider01",
         "scheduledEventId": 2
      },
      "eventId": 3,
      "eventTimestamp": 1326668152.648,
      "eventType": "DecisionTaskStarted"
   }, {
      "decisionTaskScheduledEventAttributes": {
         "startToCloseTimeout": "600",
         "taskList": {
            "name": "specialTaskList"
         }
      },
      "eventId": 2,
      "eventTimestamp": 1326668003.094,
      "eventType": "DecisionTaskScheduled"
   }
]
```

For a detailed list of the different types of events that can appear in the workflow execution history, see the HistoryEvent data type in the *Amazon Simple Workflow Service API Reference*.

Amazon SWF stores the complete history of all workflow executions for a configurable number of days after the execution closes. This period, which is known as the workflow history retention period, is specified when you register a *Domain* for your workflow. Domains are discussed in greater detail later in this section.

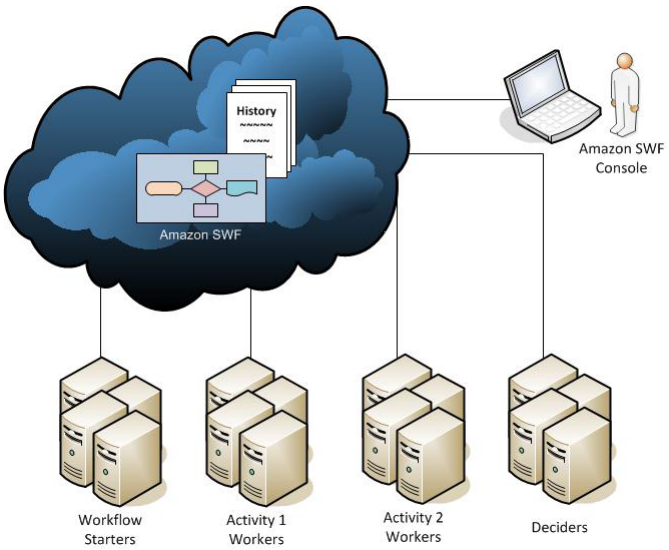
Amazon SWF Actors

Topics

What is an Actor in Amazon SWF?

In the course of its operations, Amazon SWF interacts with a number of different types of programmatic *actors*. Actors can be workflow starters (p. 42), deciders (p. 42), or activity workers (p. 43). These actors communicate with Amazon SWF through its API. You can develop these actors in any programming language.

The following diagram shows the Amazon SWF architecture, including Amazon SWF and its actors.



Workflow Starters

A workflow starter is any application that can initiate workflow executions. In the e-commerce example, one workflow starter could be the website at which the customer places an order. Another workflow starter could be a mobile application or system used by a customer service representative to place the order on behalf of the customer.

Deciders

A decider is an implementation of a workflow's coordination logic. Deciders control the flow of activity tasks in a workflow execution. Whenever a change occurs during a workflow execution, such as the completion of an activity task, Amazon SWF creates a decision task that contains the workflow history up to that point in time and assigns the task to a decider. When the decider receives the decision task from Amazon SWF, it analyzes the workflow execution history to determine the next appropriate steps in the workflow execution. The decider communicates these steps back to Amazon SWF using *decisions*. A decision is an Amazon SWF data type that can represent various next actions. For a list of the possible decisions, go to Decision in the Amazon Simple Workflow Service API Reference.

Here is an example of a decision in JSON format, the format in which it is transmitted to Amazon SWF. This decision schedules a new activity task.

```
{
   "decisionType" : "ScheduleActivityTask",
   "scheduleActivityTaskDecisionAttributes" : {
      "activityType" : {
         "name" : "activityVerify",
         "version" : "1.0"
      },
      "activityId" : "verification-27",
      "control" : "digital music",
      "input" : "5634-0056-4367-0923,12/12,437",
      "scheduleToCloseTimeout" : "900",
      "taskList" : {
         "name": "specialTaskList"
      },
      "scheduleToStartTimeout" : "300",
      "startToCloseTimeout" : "600",
      "heartbeatTimeout" : "120"
   }
}
```

A decider receives a decision task when the workflow execution starts and each time a state change occurs in the workflow execution. Deciders continue to move the workflow execution forward by receiving decision tasks and responding to Amazon SWF with more decisions until the decider determines that the workflow execution is complete. It then responds with a decision to close the workflow execution. After the workflow execution closes, Amazon SWF will not schedule additional tasks for that execution.

In the e-commerce example, the decider determines if each step was performed correctly, and then either schedules the next step or manages any error conditions.

A decider represents a single computer process or thread. Multiple deciders can process tasks for the same workflow type.

Activity Workers

An activity worker is a process or thread that performs the *activity tasks* that are part of your workflow. The activity task represents one of the tasks that you identified in your application.

To use an activity task in your workflow, you must register it using either the Amazon SWF console or the RegisterActivityType action.

Each activity worker polls Amazon SWF for new tasks that are appropriate for that activity worker to perform; certain tasks can be performed only by certain activity workers. After receiving a task, the activity worker processes the task to completion and then reports to Amazon SWF that the task was completed and provides the result. The activity worker then polls for a new task. The activity workers associated with a workflow execution continue in this way, processing tasks until the workflow execution itself is complete. In the e-commerce example, activity workers are independent processes and applications used by people, such as credit card processors and warehouse employees, that perform individual steps in the process.

An activity worker represents a single computer process (or thread). Multiple activity workers can process tasks of the same activity type.

Data Exchange Between Actors

Input data can be provided to a workflow execution when it is started. Similarly, input data can be provided to activity workers when they schedule activity tasks. When an activity task is complete, the activity worker can return results to Amazon SWF. Similarly, a decider can report the results of a workflow execution when the execution is complete. Each actor can send data to, and receive data from, Amazon SWF through strings, the form of which is user-defined. Depending on the size and sensitivity of the data, you can pass data directly or pass a pointer to data stored on another system or service (such as Amazon S3 or DynamoDB). Both the data passed directly and the pointers to other data stores are recorded in the workflow execution history; however, Amazon SWF does not copy or cache any of the data from external stores as part of the history.

Because Amazon SWF maintains the complete execution state of each workflow execution, including the inputs and the results of tasks, all actors can be stateless. As a result, workflow processing is highly scalable. As the load on your system grows, you can simply add more actors to increase capacity.

Amazon SWF Tasks

Amazon SWF interacts with activity workers and deciders by providing them with work assignments known as tasks. There are two types of tasks in Amazon SWF:

- **Activity task**. An activity task tells an activity worker to perform its function, such as to check inventory or charge a credit card. The activity task contains all the information that the activity worker needs to perform its function.
- **Decision task**. A decision task tells a decider that the state of the workflow execution has changed so that the decider can determine the next activity that needs to be performed. The decision task contains the current workflow history.

Amazon SWF schedules a decision task when the workflow starts and whenever the state of the workflow changes, such as when an activity task completes. Each decision task contains a paginated view of the entire workflow execution history. The decider analyzes the workflow execution history and responds back to Amazon SWF with a set of decisions that specify what should occur next in the workflow execution. Essentially, every decision task gives the decider an opportunity to assess the workflow and provide direction back to Amazon SWF.

To ensure that no conflicting decisions are processed, Amazon SWF assigns each decision task to exactly one decider and allows only one decision task at a time to be active in a workflow execution.

The following table shows the relationship between the different constructs related to workflows and deciders.

Logical Design	Registered As	Performed By	Receives & Performs	Generates
Workflow	Workflow Type	Decider	Decision Tasks	Decisions

When an activity worker has completed the activity task, it reports to Amazon SWF that the task was completed, and it includes any relevant results that were generated. Amazon SWF updates the workflow execution history with an event that indicates the task completed and then schedules a decision task to transmit the updated history to the decider.

Amazon SWF assigns each activity task to exactly one activity worker. Once the task is assigned, no other activity worker can claim or perform that task.

The following table shows the relationship between the different constructs related to activities.

Logical Design	Registered As	Performed By	Receives & Performs	Generates
Activity	Activity Type	Activity Worker	Activity Tasks	Results Data

Amazon SWF Domains

Domains provide a way of scoping Amazon SWF resources within your AWS account. All the components of a workflow, such as the workflow type and activity types, must be specified to be in a domain. It is possible to have more than one workflow in a domain; however, workflows in different domains cannot interact with each other.

When setting up a new workflow, before you set up any of the other workflow components you need to register a domain if you have not already done so.

When you register a domain, you specify a *workflow history retention period*. This period is the length of time that Amazon SWF will continue to retain information about the workflow execution after the workflow execution is complete.

Amazon SWF Object Identifiers

The following list describes how Amazon SWF objects, such as workflow executions, are uniquely identified.

- **Workflow Type:** A registered workflow type is identified by its domain, name, and version. Workflow types are specified in the call to `RegisterWorkflowType`.
- **Activity Type:** A registered activity type is identified by its domain, name, and version. Activity types are specified in the call to `RegisterActivityType`.
- **Decision Tasks and Activity Tasks:** Each decision task and activity task is identified by a unique task token. The task token is generated by Amazon SWF and is returned with other information about the task in the response from `PollForDecisionTask` or `PollForActivityTask`. Although the token is most commonly used by the process that received the task, that process could pass the token to another process, which could then report the completion or failure of the task.
- **Workflow Execution:** A single execution of a workflow is identified by the domain, workflow ID, and run ID. The first two are parameters that are passed to StartWorkflowExecution. The run ID is returned by `StartWorkflowExecution`.

Amazon SWF Task Lists

Task lists provide a way of organizing the various tasks associated with a workflow. You could think of task lists as similar to dynamic queues. When a task is scheduled in Amazon SWF, you can specify a queue (task list) to put it in. Similarly, when you poll Amazon SWF for a task you say which queue (task list) to get the task from.

Task lists provide a flexible mechanism to route tasks to workers as your use case necessitates. Task lists are dynamic in that you don't need to register a task list or explicitly create it through an action: simply scheduling a task creates the task list if it doesn't already exist.

There are separate lists for activity tasks and decision tasks. A task is always scheduled on only one task list; tasks are not shared across lists.

Topics

Decision Task Lists

Each workflow execution is associated with a specific decision task list. When a workflow type is registered (RegisterWorkflowType action), you can specify a default task list for executions of that workflow type. When the workflow starter initiates the workflow execution (`StartWorkflowExecution` action), it has the option of specifying a different task list for that workflow execution.

When a decider polls for a new decision task (`PollForDecisionTask` action), the decider specifies a decision task list to draw from. A single decider could serve multiple workflow executions by calling `PollForDecisionTask` multiple times, using a different task list in each call, where each task list is specific to a particular workflow execution. Alternatively, the decider could poll a single decision task list that provides decision tasks for multiple workflow executions. You could also have multiple deciders serving a single workflow execution by having all of them poll the task list for that workflow execution.

Activity Task Lists

A single activity task list can contain tasks of different activity types. Tasks are scheduled on the task list in order. Amazon SWF returns the tasks from the list in order on a best effort basis. Under some circumstances, the tasks may not come off the list in order.

When an activity type is registered (RegisterActivityType action), you can specify a default task list for that activity type. By default, activity tasks of this type will be scheduled on the specified task list; however, when the decider schedules an activity task (ScheduleActivityTask decision), the decider can optionally specify a different task list on which to schedule the task. If the decider does not specify a task list, the default task list is used. As a result, you can place activity tasks on specific task lists according to attributes of the task. For example, you could place all instances of an activity task for a given credit card type on a particular task list.

Task Routing

When an activity worker polls for a new task (PollForActivityTask action), it can specify an activity task list to draw from. If it does, the activity worker will accept tasks only from that list. In this way, you can ensure that certain tasks get assigned only to particular activity workers. For example, you might create a task list that holds tasks that require the use of a high-performance computer. Only activity workers running on the appropriate hardware would poll that task list. Another example would be to create a task list for a particular geographic region. You could then ensure that only workers deployed in that region would pick up those tasks. Or you could create a task list for high-priority orders and always check that list first.

Assigning particular tasks to particular activity workers in this way is called *task routing*. Task routing is optional; if you do not specify a task list when scheduling an activity task, the task is automatically placed on the default task list.

Amazon SWF Workflow Execution Closure

Once you start a workflow execution, it is open. An open workflow execution could be closed as completed, canceled, failed, or timed out. It could also be continued as a new execution, or it could be terminated.

A workflow execution could be closed by the decider, by the person administering the workflow, or by Amazon SWF.

If the decider determines that the activities of the workflow have finished, it should close the workflow execution as completed by using the RespondDecisionTaskCompleted action and pass the CompleteWorkflowExecution decision.

Alternatively, a decider might close the workflow execution as canceled or failed. In order to cancel the execution, the decider should use the `RespondDecisionTaskCompleted` action and pass the CancelWorkflowExecution decision.

A decider should fail the workflow execution if it enters a state outside the realm of normal completion. In order to fail the execution, the decider should use the `RespondDecisionTaskCompleted` action and pass the FailWorkflowExecution decision.

Amazon SWF monitors workflow executions to ensure that they do not exceed any user-specified timeout settings. If a workflow execution times out, Amazon SWF automatically closes it. For more information about timeout values, see the Timeout Types (p. 118) section.

A decider might also close the execution and logically continue it as a new execution using the `RespondDecisionTaskCompleted` action and passing the ContinueAsNewWorkflowExecution decision. This is a useful strategy for long-running workflow executions for which the history may grow too large over time.

Finally, you could terminate workflow executions directly from the Amazon SWF console or programmatically by using the TerminateWorkflowExecution API. Termination forces closure of the workflow execution. Cancellation is preferred over termination, because your deciders can manage closure of the workflow execution.

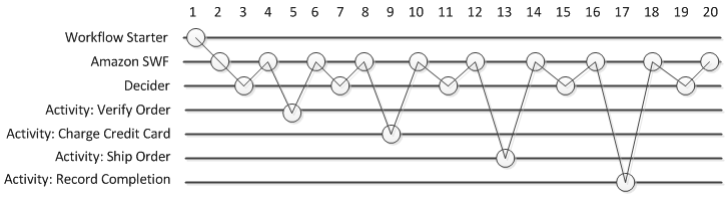
Amazon SWF would terminate a workflow execution if the execution exceeds certain service-defined limits. Amazon SWF would also terminate a child workflow if the parent workflow has terminated and the applicable child policy indicates that the child workflow should also be terminated.

Life Cycle of an Amazon SWF Workflow Execution

Life Cycle of an Amazon SWF Workflow Execution

From the start of a workflow execution to its completion, Amazon SWF interacts with actors by assigning them appropriate tasks, either activity tasks or decision tasks.

The following diagram shows the life cycle of an order-processing workflow execution from the perspective of components that act on it.



The following table explains each task in the preceding image.

Workflow Execution Life Cycle

Description	Action, Decision, or Event
1) The workflow starter calls the appropriate Amazon SWF action to start the workflow execution for an order, providing the order information.	StartWorkflowExecution action.
2) Amazon SWF receives the start workflow execution request and then schedules the first decision task.	WorkflowExecutionStarted event and DecisionTaskScheduled event.
3) The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic to determine that no previous activities occurred, makes a decision to schedule the Verify Order activity with the information the activity worker needs to process the task, and returns the decision to Amazon SWF.	PollForDecisionTask action. RespondDecisionTaskCompleted action with ScheduleActivityTask decision.
4) Amazon SWF receives the decision, schedules the Verify Order activity task, and waits for the activity task to complete or time out.	ActivityTaskScheduled event.
5) An activity worker that can perform the Verify Order activity receives the task, performs it, and returns the results to Amazon SWF.	`PollForActivityTask` action and `RespondActivityTaskCompleted` action.
6) Amazon SWF receives the results of the Verify Order activity, adds them to the workflow history, and schedules a decision task.	ActivityTaskCompleted event and DecisionTaskScheduled event.

Description	Action, Decision, or Event
7) The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a ChargeCreditCard activity task with the information the activity worker needs to process the task, and returns the decision to Amazon SWF.	PollForDecisionTask action. RespondDecisionTaskCompleted action with ScheduleActivityTask decision.
8) Amazon SWF receives the decision, schedules the ChargeCreditCard activity task, and waits for it to complete or time out.	DecisionTaskCompleted event and ActivityTaskScheduled event.
9) An activity worker that can perform the ChargeCreditCard activity receives the task, performs it, and returns the results to Amazon SWF.	`PollForActivityTask` and `RespondActivityTaskCompleted` action.
10) Amazon SWF receives the results of the ChargeCreditCard activity task, adds them to the workflow history, and schedules a decision task.	ActivityTaskCompleted event and DecisionTaskScheduled event.
11) The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a ShipOrder activity task with the information the activity worker needs to perform the task, and returns the decision to Amazon SWF.	PollForDecisionTask action. RespondDecisionTaskCompleted with ScheduleActivityTask decision.

Description	Action, Decision, or Event
12) Amazon SWF receives the decision, schedules a ShipOrder activity task, and waits for it to complete or time out.	DecisionTaskCompleted event and ActivityTaskScheduled event.
13) An activity worker that can perform the ShipOrder activity receives the task, performs it, and returns the results to Amazon SWF.	`PollForActivityTask` action and `RespondActivityTaskCompleted` action.
14) Amazon SWF receives the results of the ShipOrder activity task, adds them to the workflow history, and schedules a decision task.	ActivityTaskCompleted event and DecisionTaskScheduled event.
15) The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a RecordCompletion activity task with the information the activity worker needs to perform the task, and returns the decision to Amazon SWF.	PollForDecisionTask action. RespondDecisionTaskCompleted action with ScheduleActivityTask decision.
16) Amazon SWF receives the decision, schedules a RecordCompletion activity task, and waits for it to complete or time out.	DecisionTaskCompleted event and ActivityTaskScheduled event.
17) An activity worker that can perform the RecordCompletion activity receives the task, performs it, and returns the results to Amazon SWF.	`PollForActivityTask` action and `RespondActivityTaskCompleted` action.

Description	Action, Decision, or Event
18) Amazon SWF receives the results of the RecordCompletion activity task, adds them to the workflow history, and schedules a decision task.	ActivityTaskCompleted event and DecisionTaskScheduled event.
19) The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to close the workflow execution and returns the decision along with any results to Amazon SWF.	PollForDecisionTask action. RespondDecisionTaskCompleted action with CompleteWorkflowExecution decision
20) Amazon SWF closes the workflow execution and archives the history for future reference.	WorkflowExecutionCompleted event.

Polling for Tasks in Amazon SWF

Deciders and activity workers communicate with Amazon SWF using *long polling*. The decider or activity worker periodically initiates communication with Amazon SWF, notifying Amazon SWF of its availability to accept a task, and then specifies a task list to get tasks from.

If a task is available on the specified task list, Amazon SWF returns it immediately in the response. If no task is available, Amazon SWF holds the TCP connection open for up to 60 seconds so that, if a task becomes available during that time, it can be returned in the same connection. If no task becomes available within 60 seconds, it returns an empty response and closes the connection. (An empty response is a Task structure in which the value of taskToken is an empty string.) If this happens, the decider or activity worker should poll again.

Long polling works well for high-volume task processing. Deciders and activity workers can manage their own capacity, and is easy to use when the deciders and activity workers are behind a firewall.

For more information, see Polling for Decision Tasks (p. 80) and Polling for Activity Tasks (p. 76).

Using IAM to Manage Access to Amazon SWF Resources

Every actor that accesses an Amazon SWF resource—deciders, activity workers, workflow administrators—must have authorized AWS access keys. An actor can access resources by using the account's access keys. However, access keys provide unrestricted access to all of the account's resources and are difficult to revoke, so they are not appropriate for all applications.

Amazon SWF uses AWS Identity and Access Management (IAM) to provide controlled access to resources. IAM provides a flexible way to manage access to an account's AWS resources without exposing the access keys. With IAM, you create one or more *users* that are associated with the AWS account. Each user has a separate set of IAM access keys that provide access to the account's resources. You then attach an *IAM policy* to the user—or a group that includes the user—to specify which resources the user can access. The policy can be much more granular than simply specifying whether to allow or deny account access. You could, for example, create a policy that allows a user to access an account, but only for a specified set of domains.

A further advantage of IAM is that you can revoke IAM access without affecting your access keys. In fact, periodically *rotating access keys*—revoking users' IAM access keys and issuing new ones—is a security best practice.

This topic discusses the details of how to use IAM to provide controlled access to Amazon SWF resources. It assumes that you are generally familiar with IAM, which is described in detail in the following documents.

- AWS Identity and Access Management (IAM)
- Using Identity and Access Management

Basic Principles

Amazon SWF access control is based primarily on two types of permissions:

- Resource permissions: Which Amazon SWF resources a user can access.

  You can express resource permissions only for domains.
- API permissions: Which Amazon SWF actions a user can call.

The simplest approach is to grant full account access—call any Amazon SWF action in any domain—or deny access entirely. However, IAM supports a more granular approach to access control that is often more useful. For example, you could:

- Allow a user to call any Amazon SWF action without restrictions, but only in a specified domain. You could use such a policy to allow workflow applications that are under development to use any action, but only a "sandbox" domain.
- Allow a user to access any domain, but constrain how they use the API. You could use such a policy to allow an "auditor" application to call the API in any domain, but allow only read access.
- Allow a user to call only a limited set of actions in certain domains. You could use such a policy to allow a workflow starter to call only the `StartWorkflowExecution` action in a specified domain.

Amazon SWF access control is based on the following principles:

- Access control decisions are based only on IAM policies; all policy auditing and manipulation is done through IAM.
- The access control model uses a deny-by-default policy; any access that is not explicitly allowed is denied.
- You control access to Amazon SWF resources by attaching appropriate IAM policies to the workflow's actors.
- Resource permissions can be expressed only for domains.
- You can further constrain the usage of some actions by applying conditions to one or more parameters.
- If you grant permission to use RespondDecisionTaskCompleted, you can express permissions for the list of decisions included in that action.

  Each of the decisions has one or more parameters, much like a regular API call. To allow for policies to be as readable as possible, you can express permissions on decisions as if they were actual API calls, including applying conditions to some parameters. These types of permissions are called *pseudo API* permissions.

For a summary of which regular and pseudo API parameters can be constrained by using conditions, see API Summary (p. 59).

Amazon SWF IAM Policies

An IAM policy contains one or more **Statement** elements, each of which contains a set of elements that define the policy. For a complete list of elements and a general discussion of how to construct policies, see The Access Policy Language. Amazon SWF access control is based on the following elements:

Effect
: [Required] The effect of the statement: **deny** or **allow**.

  Note
  You must explicitly allow access; IAM denies access by default.

Resource
: [Required] The resource—an entity in an AWS service that a user can interact with—that the statement applies to.

  You can express resource permissions only for domains. For example, a policy can allow access to only certain domains in your account. To express permissions for a domain, set **Resource** to the domain's Amazon Resource Name (ARN), which has the format "arn:aws:swf:*Region*:*AccountID*:/domain/*DomainName*". *Region* is the AWS region, *AccountID* is the account ID with no dashes, and *DomainName* is the domain name.

Action

[Required] The action that the statement applies to, which you refer to by using the following format: *`serviceId`*:*`action`*. For Amazon SWF, set *`serviceID`* to `swf`. For example, `swf:StartWorkflowExecution` refers to the StartWorkflowExecution action, and is used to control which users are allowed to start workflows.

If you grant permission to use RespondDecisionTaskCompleted, you can also control access to the included list of decisions by using **Action** to express permissions for the pseudo API. Because IAM denies access by default, a decider's decision must be explicitly allowed or it will not be accepted. You can use a '*' value to allow all decisions.

Condition

[Optional] Expresses a constraint on one or more of an action's parameter's, which restricts the allowed values.

Amazon SWF actions often have a wide scope, which you can reduce by using IAM conditions. For example, to limit which task lists the PollForActivityTask action is allowed to access, you include a **Condition** and use the `swf:taskList.name` key to specify the allowable lists.

You can express constraints for the following entities.

- The workflow type. The name and version have separate keys.
- The activity type. The name and version have separate keys.
- Task lists.
- Tags. You can specify multiple tags for some actions. In that case, each tag has a separate key.

Note

For Amazon SWF, the values are all strings so you constrain a parameter by using a string operator such as `StringEquals`, which restricts the parameter to a specified string. However, the regular string comparison operators such as `StringEquals` require all requests to include the parameter. If you do not include the parameter explicitly, and there is no default value such as the default task list provided during type registration, access will be denied.
It is often useful to treat conditions as optional, so that you can call an action without necessarily including the associated parameter. For example, you might want to allow a decider to specify a set of RespondDecisionTaskCompleted decisions, but also allow it to specify only one of them for any particular call. In that case, you constrain the appropriate parameters by using a `StringEqualsIfExists` operator, which allows access if the parameter satisfies the condition, but does not deny access if the parameter is absent.

For a complete list of constrainable parameters and the associated keys, see API Summary (p. 59).

The following section provides examples of how to construct Amazon SWF policies. For details, see String Conditions.

Amazon SWF Policy Examples

A workflow consists of multiple actors—activities, deciders, and so on. You can control access for each actor by attaching an appropriate IAM policy. This section provides some examples. The following shows the simplest case:

```
{
   "Version": "2012-10-17",
   "Statement" : [ {
      "Effect" : "Allow",
      "Action" : "swf:*",
```

```
      "Resource" : "arn:aws:swf:*:123456789012:/domain/*"
   } ]
}
```

If you attach this policy to an actor, it has full account access across all regions. You can use wildcards to have a single value represent multiple resources, actions, or regions.

- The first '*' wildcard in the **Resource** value (...:swf:*:123...) indicates that the resource permissions apply to all regions. To restrict permissions to a single region, replace the '*' with the appropriate region string, such as us-east-1.
- The second '*' wildcard in the **Resource** value (/domain/*) allows the actor to access any of the account's domains in the specified regions.
- The '*' wildcard in the **Action** value allows the actor to call any Amazon SWF action.

For details on how to use wildcards, see Element Descriptions

The following sections show examples of policies that grant permissions in a more granular way.

Domain Permissions

If you want to restrict a department's workflows to a particular domain, you can use something like:

```
{
   "Version": "2012-10-17",
   "Statement": [ {
      "Effect" : "Allow",
      "Action" : "swf:*",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/department1"
   } ]
}
```

If you attach this policy to an actor, it can call any action, but only for the department1 domain.

If you want an actor to have access to more than one domain, you can express permission for each domain separately, as follows:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect" : "Allow",
         "Action" : "swf:*",
         "Resource" : "arn:aws:swf:*:123456789012:/domain/department1"
      }, {
         "Effect" : "Allow",
         "Action" : "swf:*",
         "Resource" : "arn:aws:swf:*:123456789012:/domain/department2"
      }
   ]
}
```

If you attach this policy to an actor, it can use any Amazon SWF action in the "department1" and "department2" domains. You can also sometimes use wildcards to represent multiple domains.

API Permissions and Constraints

You control which actions an actor can use with the **Action** element. Optionally, you can constrain the action's allowable parameter values by using a **Condition** element.

If you want to restrict an actor to only certain actions, you can use something like the following:

```
{
   "Version": "2012-10-17",
   "Statement": [ {
      "Effect" : "Allow",
      "Action" : "swf:StartWorkflowExecution",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/department2"
   } ]
}
```

If you attach this policy to an actor, it can call `StartWorkflowExecution` to start workflows in the "department2" domain. It cannot use any other actions or start workflows in any other domains.

You can further restrict which workflows an actor can start by constraining one or more of the `StartWorkflowExecution` parameter values, as follows:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect" : "Allow",
         "Action" : "swf:StartWorkflowExecution",
         "Resource" : "arn:aws:swf:*:123456789012:/domain/department1",
         "Condition" : {
            "StringEquals" : { "swf:workflowType.name" : "workflow1" },
            "StringEquals" : { "swf:workflowType.version" : "version2" }
         }
      }
   ]
}
```

This policy constrains the `StartWorkflowExecution` action's **name** and **version** parameters. If you attach the policy to an actor, it can run only "version2" of "workflow1" in the "department1" domain and both parameters must be included in the request.

You can constrain a parameter without requiring it to be included in a request by using a `StringEqualsIfExists` operator, as follows:

```
{
   "Version": "2012-10-17",
   "Statement" : [ {
      "Effect" : "Allow",
      "Action" : "swf:StartWorkflowExecution",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/some_domain",
      "Condition" : {
         "StringEqualsIfExists" : { "swf:taskList.name" : "task_list_name" }
      }
```

```
   } ]
}
```

This policy allows an actor to optionally specify a task list when starting a workflow execution.

You can constrain a list of tags for some actions. In that case, each tag has a separate key, so you use `swf:tagList.member.0` to constrain the first tag in the list, `swf:tagList.member.1` to constrain the second tag in the list, and so on, up to a maximum of 5. However, you must be careful how you constrain tag lists. For instance, here is an example of a policy that is *not* recommended:

```
{
   "Version": "2012-10-17",
   "Statement" : [ {
      "Effect" : "Allow",
      "Action" : "swf:StartWorkflowExecution",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/some_domain",
      "Condition" : {
         "StringEqualsIfExists" : {
            "swf:tagList.member.0" : "some_ok_tag", "another_ok_tag"
         }
      }
   } ]
}
```

This policy allows you to optionally specify either "some_ok_tag" or "another_ok_tag". However, this policy constrains only the first element of the tag list. The list could have additional elements with arbitrary values that would all be allowed because this policy does not apply any conditions to `swf:tagList.member.1`, `swf:tagList.member.2`, and so on .

One way to address this issue is to disallow the use of tag lists. The following policy ensures that only "some_ok_tag" or "another_ok_tag" are allowed by requiring the list to have only one element.

```
{
   "Version": "2012-10-17",
   "Statement" : [ {
      "Effect" : "Allow",
      "Action" : "swf:StartWorkflowExecution",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/some_domain",
      "Condition" : {
         "StringEqualsIfExists" : {
            "swf:tagList.member.0" : "some_ok_tag", "another_ok_tag"
         },
         "Null" : { "swf:tagList.member.1" : "true" }
      }
   } ]
}
```

Pseudo API Permissions and Constraints

If you want to restrict the decisions available to `RespondDecisionTaskCompleted`, you must first allow the actor to call `RespondDecisionTaskCompleted`. You can then express permissions for the appropriate pseudo API members by using the same syntax as for the regular API, as follows:

```
{
   "Version": "2012-10-17",
   "Statement" : [
      {
         "Resource" : "arn:aws:swf:*:123456789012:/domain/*",
         "Action" : "swf:RespondDecisionTaskCompleted",
         "Effect" : "Allow"
      }, {
         "Resource" : "*",
         "Action" : "swf:ScheduleActivityTask",
         "Effect" : "Allow",
         "Condition" : {
            "StringEquals" : { "swf:activityType.name" : "SomeActivityType" }
         }
      }
   ]
}
```

If you attach this policy to an actor, the first `Statement` element allows the actor to call `RespondDecisionTaskCompleted`. The second element allows the actor to use the `ScheduleActivityTask` decision to direct Amazon SWF to schedule an activity task. To allow all decisions, replace "swf:ScheduleActivityTask" with "swf:*".

You can use Condition operators to constrain parameters just as with the regular API. The `StringEquals` operator in this Condition allows `RespondDecisionTaskCompleted` to schedule an activity task for the "SomeActivityType" activity, and it must schedule that task. If you want to allow `RespondDecisionTaskCompleted` to use a parameter value but not require it to do so, you can instead use the `StringEqualsIfExists` operator.

Service Model Limitations on IAM Policies

You must consider service model constraints when creating IAM policies. It is possible to create a syntactically valid IAM policy that represents an invalid Amazon SWF request; a request that is allowed in terms of access control can still fail because it is an invalid request.

For instance, the following policy for ListOpenWorkflowExecutions is *not* recommended:

```
{
   "Version": "2012-10-17",
   "Statement" : [ {
      "Effect" : "Allow",
      "Action" : "swf:ListOpenWorkflowExecutions",
      "Resource" : "arn:aws:swf:*:123456789012:/domain/domain_name",
      "Condition" : {
         "StringEquals" : { "swf:typeFilter.name" : "workflow_name" },
         "StringEquals" : { "swf:typeFilter.version" : "workflow_version" },
         "StringEquals" : { "swf:tagFilter.tag" : "some_tag" }
      }
   } ]
}
```

The Amazon SWF service model does not allow the `typeFilter` and `tagFilter` parameters to be used in the same `ListOpenWorkflowExecutions` request. The policy therefore allows calls that the service will reject—by throwing `ValidationException`—as an invalid request.

API Summary

This section briefly describes how you can use IAM policies to control how an actor can use each API and pseudo API to access Amazon SWF resources.

- For all actions except `RegisterDomain` and `ListDomains`, you can allow or deny access to any or all of an account's domains by expressing permissions for the domain resource.
- You can allow or deny permission for any member of the regular API and, if you grant permission to call RespondDecisionTaskCompleted, any member of the pseudo API.
- You can use a Condition to constrain some parameters' allowable values.

The following sections list the parameters that can be constrained for each member of the regular and pseudo API and provide the associated key, and note any limitations on how you can control domain access.

Regular API

This section lists the regular API members, and briefly describes the parameters that can be constrained and the associated keys. It also notes any limitations on how you can control domain access.

CountClosedWorkflowExecutions

- **tagFilter.tag**: String constraint. The key is `swf:tagFilter.tag`
- **typeFilter.name**: String constraint. The key is `swf:typeFilter.name`.
- **typeFilter.version**: String constraint. The key is `swf:typeFilter.version`.

> **Note**
> `CountClosedWorkflowExecutions` requires **typeFilter** and **tagFilter** to be mutually exclusive.

CountOpenWorkflowExecutions

- **tagFilter.tag**: String constraint. The key is `swf:tagFilter.tag`
- **typeFilter.name**: String constraint. The key is `swf:typeFilter.name`.
- **typeFilter.version**: String constraint. The key is `swf:typeFilter.version`.

> **Note**
> `CountOpenWorkflowExecutions` requires **typeFilter** and **tagFilter** to be mutually exclusive.

CountPendingActivityTasks

- **taskList.name**: String constraint. The key is `swf:taskList.name`.

CountPendingDecisionTasks

- **taskList.name**: String constraint. The key is `swf:taskList.name`.

DeprecateActivityType

- **activityType.name**: string constraint. The key is `swf:activityType.name`.
- **activityType.version**: String constraint. The key is `swf:activityType.version`.

DeprecateDomain

- You cannot constrain this action's parameters.

DeprecateWorkflowType

- **workflowType.name**: String constraint. The key is `swf:workflowType.name`.
- **workflowType.version**: String constraint. The key is `swf:workflowType.version`.

DescribeActivityType

- **activityType.name**: string constraint. The key is `swf:activityType.name`.
- **activityType.version**: String constraint. The key is `swf:activityType.version`.

DescribeDomain

- You cannot constrain this action's parameters.

DescribeWorkflowExecution

- You cannot constrain this action's parameters.

DescribeWorkflowType

- **workflowType.name**: String constraint. The key is `swf:workflowType.name`.
- **workflowType.version**: String constraint. The key is `swf:workflowType.version`.

GetWorkflowExecutionHistory

- You cannot constrain this action's parameters.

ListActivityTypes

- You cannot constrain this action's parameters.

ListClosedWorkflowExecutions

- **tagFilter.tag**: String constraint. The key is `swf:tagFilter.tag`
- **typeFilter.name**: String constraint. The key is `swf:typeFilter.name`.
- **typeFilter.version**: String constraint. The key is `swf:typeFilter.version`.

> **Note**
> `ListClosedWorkflowExecutions` requires **typeFilter** and **tagFilter** to be mutually exclusive.

ListDomains

- You cannot constrain this action's parameters.

ListOpenWorkflowExecutions

- **tagFilter.tag**: String constraint. The key is `swf:tagFilter.tag`

- **typeFilter.name**: String constraint. The key is `swf:typeFilter.name`.
- **typeFilter.version**: String constraint. The key is `swf:typeFilter.version`.

> **Note**
> `ListOpenWorkflowExecutions` requires **typeFilter** and **tagFilter** to be mutually exclusive.

ListWorkflowTypes

- You cannot constrain this action's parameters.

PollForActivityTask

- **taskList.name**: String constraint. The key is `swf:taskList.name`.

PollForDecisionTask

- **taskList.name**: String constraint. The key is `swf:taskList.name`.

RecordActivityTaskHeartbeat

- You cannot constrain this action's parameters.

RegisterActivityType

- **defaultTaskList.name**: String constraint. The key is `swf:defaultTaskList.name`.
- **name**: String constraint. The key is `swf:name`.
- **version**: String constraint. The key is `swf:version`.

RegisterDomain

- **name**: The name of the domain being registered is available as the resource of this action.

RegisterWorkflowType

- **defaultTaskList.name**: String constraint. The key is `swf:defaultTaskList.name`.
- **name**: String constraint. The key is `swf:name`.
- **version**: String constraint. The key is `swf:version`.

RequestCancelWorkflowExecution

- You cannot constrain this action's parameters.

RespondActivityTaskCanceled

- You cannot constrain this action's parameters.

RespondActivityTaskCompleted

- You cannot constrain this action's parameters.

RespondActivityTaskFailed

- You cannot constrain this action's parameters.

RespondDecisionTaskCompleted

- **decisions.member.N**: Restricted indirectly through pseudo API permissions. For details, see Pseudo API (p. 62).

SignalWorkflowExecution

- You cannot constrain this action's parameters.

StartWorkflowExecution

- **tagList.member.0**: String constraint. The key is `swf:tagList.member.0`
- **tagList.member.1**: String constraint. The key is `swf:tagList.member.1`
- **tagList.member.2**: String constraint. The key is `swf:tagList.member.2`
- **tagList.member.3**: String constraint. The key is `swf:tagList.member.3`
- **tagList.member.4**: String constraint. The key is `swf:tagList.member.4`
- **taskList.name**: String constraint. The key is `swf:taskList.name`.
- **workflowType.name**: String constraint. The key is `swf:workflowType.name`.
- **workflowType.version**: String constraint. The key is `swf:workflowType.version`.

> **Note**
> You cannot constrain more than five tags.

TerminateWorkflowExecution

- You cannot constrain this action's parameters.

Pseudo API

This section lists the members of the pseudo API, which represent the decisions included in RespondDecisionTaskCompleted. If you have granted permission to use `RespondDecisionTaskCompleted`, your policy can express permissions for the members of this API in the same way as the regular API. You can further restrict some members of the pseudo-API by setting conditions on one or more parameters. This section lists the pseudo API members, and briefly describes the parameters that can be constrained and the associated keys.

> **Note**
> The `aws:SourceIP`, `aws:UserAgent`, and `aws:SecureTransport` keys are not available for the pseudo API. If your intended security policy requires these keys to control access to the pseudo API, you can use them with the `RespondDecisionTaskCompleted` action.

`CancelTimer`

- You cannot constrain this action's parameters.

`CancelWorkflowExecution`

- You cannot constrain this action's parameters.

`CompleteWorkflowExecution`

- You cannot constrain this action's parameters.

`ContinueAsNewWorkflowExecution`

- **tagList.member.0**: String constraint. The key is `swf:tagList.member.0`
- **tagList.member.1**: String constraint. The key is `swf:tagList.member.1`
- **tagList.member.2**: String constraint. The key is `swf:tagList.member.2`
- **tagList.member.3**: String constraint. The key is `swf:tagList.member.3`
- **tagList.member.4**: String constraint. The key is `swf:tagList.member.4`
- **taskList.name**: String constraint. The key is `swf:taskList.name`.
- **workflowTypeVersion**: String constraint. The key is `swf:workflowTypeVersion`.

Note
You cannot constrain more than five tags.

`FailWorkflowExecution`

- You cannot constrain this action's parameters.

`RecordMarker`

- You cannot constrain this action's parameters.

`RequestCancelActivityTask`

- You cannot constrain this action's parameters.

`RequestCancelExternalWorkflowExecution`

- You cannot constrain this action's parameters.

`ScheduleActivityTask`

- **activityType.name**: String constraint. The key is `swf:activityType.name`.
- **activityType.version**: String constraint. The key is `swf:activityType.version`.
- **taskList.name**: String constraint. The key is `swf:taskList.name`.

`SignalExternalWorkflowExecution`

- You cannot constrain this action's parameters.

`StartChildWorkflowExecution`

- **tagList.member.0**: String constraint. The key is `swf:tagList.member.0`
- **tagList.member.1**: String constraint. The key is `swf:tagList.member.1`
- **tagList.member.2**: String constraint. The key is `swf:tagList.member.2`
- **tagList.member.3**: String constraint. The key is `swf:tagList.member.3`
- **tagList.member.4**: String constraint. The key is `swf:tagList.member.4`
- **taskList.name**:String constraint. The key is `swf:taskList.name`.
- **workflowType.name**: String constraint. The key is `swf:workflowType.name`.

- **workflowType.version**: String constraint. The key is `swf:workflowType.version`.

  Note
  You cannot constrain more than five tags.

`StartTimer`

- You cannot constrain this action's parameters.

Advanced Concepts in Amazon SWF

Topics

The e-commerce example in the Basic Concepts (p. 36) section represents a simplified workflow scenario. In reality, you are likely to want your workflow to do concurrent tasks (send an order confirmation email while authorizing a credit card), record major events (all items are packed), update the order with changes (add or remove an item), and make other more advanced decisions as part of your workflow execution. This section describes advanced workflow features that you can use to construct robust and sophisticated workflows.

Versioning

Business needs often require you to have different implementations or variations of the same workflow or activity running simultaneously. For example, you might want to test a new implementation of a workflow while another one is in production. You might also want to run two different implementations with two different feature sets, such as a basic and premium implementation. Versioning enables you to run multiple implementations of workflows and activities concurrently, for any purpose that meets your requirements.

Workflow and activity types have a version associated with them which is specified at registration time. Version is a free-form string and you can choose your own versioning scheme. In order to create a new version of a registered type, you should register it with the same name and a different version. Task Lists (p. 45), described earlier, can further help you to implement versioning. Consider a situation in which you have long-running workflow executions of a given type already in progress, and circumstances require that you revise the workflow, such as to add a new feature. You could implement the new feature by creating new versions of activity types and workers, and a new decider. Then you could launch executions of the new workflow version using a different set of task lists. This way, you could have executions of workflows of different versions running simultaneously without affecting each other.

Signals

Signals enable you to inject information into a running workflow execution. In some scenarios, you might want to add information to a running workflow execution to let it know that something has changed or to inform it of an external event. Any process can send a signal to an open workflow execution. For example, one workflow execution might signal another.

To use signals, define the signal name and data to be passed to the signal—if any. Then, program the decider to recognize the signal event (WorkflowExecutionSignaled) in the history and process it appropriately. When a process wants to signal a workflow execution, it makes a call to Amazon SWF (using the SignalWorkflowExecution action or, in the case of a decider, using the SignalExternalWorkflowExecution decision) that specifies the identifier for the target workflow execution, the signal name, and the signal data. Amazon SWF then receives the signal, records it in the history of the target workflow execution, and schedules a decision task for it. When the decider receives the decision task, it also receives the signal inside the workflow execution history. The decider can then take appropriate actions based on the signal and its data.

Some applications for signals include the following:

- Pausing workflow executions from progressing until a signal is received (e.g., waiting for an inventory shipment).
- Providing information to a workflow execution that might affect the logic of how deciders make decisions. This is useful for workflows affected by external events (e.g., trying to finish the sale of a stock after the market closes).
- Updating a workflow execution when you anticipate that changes might occur (e.g., changing order quantities after an order is placed and before it ships).

For cases in which a workflow should be canceled—for example, the order itself was canceled by the customer—the `RequestCancelWorkflowExecution` action should be used rather than sending a signal to the workflow.

Child Workflows

Complicated workflows can be broken into smaller, more manageable, and potentially reusable components by using child workflows. A child workflow is a workflow execution that is initiated by another (parent) workflow execution. To initiate a child workflow, the decider for the parent workflow uses the `StartChildWorkflowExecution` decision. Input data specified with this decision is made available to the child workflow through its history.

The attributes for the `StartChildWorkflowExecution` decision also specify the *child policy*, that is, how Amazon SWF should handle the situation in which the parent workflow execution terminates before the child workflow execution. There are three possible values:

- TERMINATE: Amazon SWF will terminate the child executions.
- REQUEST_CANCEL: Amazon SWF will attempt to cancel the child execution by placing a `WorkflowExecutionCancelRequested` event in the child's workflow execution history.
- ABANDON: Amazon SWF will take no action; the child executions will continue to run.

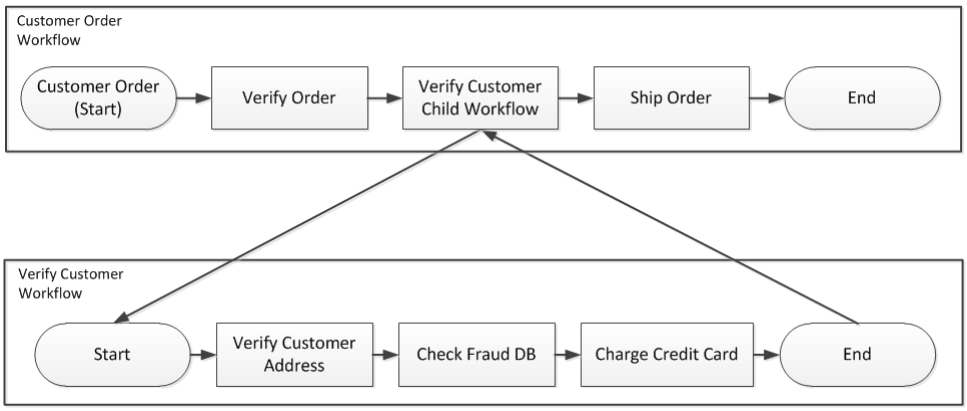
After the child workflow execution starts, it runs like a regular execution. When it completes, Amazon SWF records the completion, along with its results, in the parent workflow execution's workflow history. Examples of child workflows include the following:

- Credit card processing child workflow used by workflows in different websites

- Email child workflow that verifies the customer email address, checks the opt-out list, sends the email, and verifies that it didn't bounce or fail.
- Database storage and retrieval child workflow that combines connection, setup, transaction, and verification.
- Source code compilation child workflow that combines building, packaging, and verification.

In the e-commerce example, you might want to make the Charge Credit Card activity a child workflow. To do this, you could register a new Verify Customer workflow, register the Verify Customer Address and Check Fraud DB activities, and define coordination logic for the tasks. Then, a decider in the Customer Order workflow can initiate a Verify Customer child workflow by scheduling the `StartChildWorkflowExecution` decision that specifies this workflow type.

The following figure shows a customer order workflow that includes a new Verify Customer child workflow, which checks the customer address, checks the fraud database, and charges the credit card.



Multiple workflows could create child workflow executions using the same workflow type. For example, the Verify Customer child workflow could also be used in other parts of an organization. The events for a child workflow are contained in its own workflow history and are not included in the parent's workflow history.

Because child workflows are simply workflow executions that are initiated by a decider, they could also be started as normal stand-alone workflows executions.

Markers

At times, you might want to record information in the workflow history of a workflow execution that is specific to your use case. Markers enable you to record information in the workflow execution history that you can use for any custom or scenario-specific purpose.

To use markers, a decider uses the RecordMarker decision, names the marker, attaches desired data to the decision, and notifies Amazon SWF using the `RespondDecisionTaskCompleted` action. Amazon SWF receives the request, records the marker in the workflow history, and enacts any other decisions in the request. From that point on, deciders can see the marker in the workflow history and use it in any way that you program.

Examples of markers include the following:

- A counter that counts the number of loops in a recursive workflow.
- Progress of the workflow execution based on the results of activities.
- Information summarized from earlier workflow history events.

In the e-commerce example, you might add an activity that checks the inventory every day and increments the count in a marker each time. Then, you could add decision logic that emails the customer or notifies a manager when the count exceeds five, without having to review the entire history.

Tags

Amazon SWF enables you to associate tags with workflow executions and later query for workflow executions based on these tags. Tagging enables you to filter the listing of the executions when you use the visibility operations. By carefully selecting the tags you assign to an execution, you can use them to help provide meaningful listings.

For example, suppose you run several fulfillment centers. Proper tagging could enable you to list the processes occurring in a specific fulfillment center. Or, to take another example, if a customer is converting different types of media files, tagging could enable you to show the processing differences used for converting video, audio, and image files.

Amazon SWF supports tagging a workflow execution with up to five tags. Each tag is a free-form string and may be up to 256 characters in length. If you want to use tags, you must assign them when you start a workflow execution. You cannot add tags to a workflow execution after it has been started, nor may you edit or remove tags that have been assigned to a workflow execution.

Using the Amazon SWF API

This section describes how to develop workflows in Amazon Simple Workflow Service (Amazon SWF) using the service API. We recommend that you also refer to the *Amazon Simple Workflow Service API Reference* while reading this section. Examples of API calls in this section specify parameters using the JavaScript Object Notation (JSON) format.

Topics

List of Amazon SWF Actions by Category

This section lists the reference topics for Amazon SWF actions in the Amazon SWF application programming interface (API). These are listed by *functional category*.

For an *alphabetic* list of actions, see the Amazon Simple Workflow Service API Reference.

Topics

Actions Related to Activities

Activity workers use **PollForActivityTask** to get new activity tasks. After a worker receives an activity task from Amazon SWF, it performs the task and responds using **RespondActivityTaskCompleted** if successful or **RespondActivityTaskFailed** if unsuccessful.

The following are actions that are performed by activity workers.

- PollForActivityTask
- RespondActivityTaskCompleted
- RespondActivityTaskFailed
- RespondActivityTaskCanceled
- RecordActivityTaskHeartbeat

Actions Related to Deciders

Deciders use **PollForDecisionTask** to get decision tasks. After a decider receives a decision task from Amazon SWF, it examines its workflow execution history and decides what to do next. It calls **RespondDecisionTaskCompleted** to complete the decision task and provides zero or more next decisions.

The following are actions that are performed by deciders.

- PollForDecisionTask
- RespondDecisionTaskCompleted

Actions Related to Workflow Executions

The following actions operate on a workflow execution.

- RequestCancelWorkflowExecution
- StartWorkflowExecution
- SignalWorkflowExecution
- TerminateWorkflowExecution

Actions Related to Administration

Although you can perform administrative tasks from the Amazon SWF console, you can use the actions in this section to automate functions or build your own administrative tools.

Activity Management

- RegisterActivityType
- DeprecateActivityType

Workflow Management

- RegisterWorkflowType
- DeprecateWorkflowType

Domain Management

These actions allow you to register and deprecate Amazon SWF domains.

- RegisterDomain
- DeprecateDomain

For more information and examples of these domain management actions, see Registering a Domain (p. 72).

Workflow Execution Management

- RequestCancelWorkflowExecution
- TerminateWorkflowExecution

Visibility Actions

Although you can perform visibility actions from the Amazon SWF console, you can use the actions in this section to build your own console or administrative tools.

Activity Visibility

- ListActivityTypes
- DescribeActivityType

Workflow Visibility

- ListWorkflowTypes
- DescribeWorkflowType

Workflow Execution Visibility

- DescribeWorkflowExecution
- ListOpenWorkflowExecutions
- ListClosedWorkflowExecutions
- CountOpenWorkflowExecutions
- CountClosedWorkflowExecutions
- GetWorkflowExecutionHistory

Domain Visibility

- ListDomains
- DescribeDomain

Task List Visibility

- CountPendingActivityTasks

- CountPendingDecisionTasks

Creating a Basic Workflow in Amazon SWF

Creating a basic sequential workflow involves the following stages.

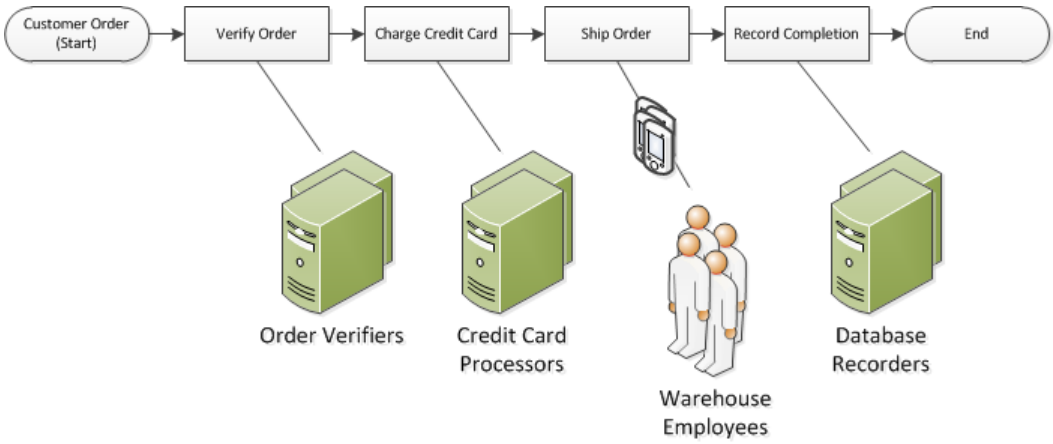
- Modeling a workflow, registering its type, and registering its activity types
- Developing and launching activity workers that perform activity tasks
- Developing and launching deciders that use the workflow history to determine what to do next
- Developing and launching workflow starters, that is, applications that start workflow executions

Modeling Your Workflow and Its Activities

To use Amazon SWF, model the logical steps in your application as activities. An activity represents a single logical step or task in your workflow. For example, authorizing a credit card is an activity that involves providing a credit card number and other information, and receiving an approval code or a message that the card was declined.

In addition to defining activities, you also need to define the coordination logic that handles decision points. For example, the coordination logic might schedule a different follow-up activity depending on whether the credit card was approved or declined.

The following figure shows an example of a sequential customer order workflow with four activities (Verify Order, Charge Credit Card, Ship Order, and Record Completion).



Registering a Domain with Amazon SWF

Your workflow and activity types and the workflow execution itself are all scoped to a *domain*. Domains isolate a set of types, executions, and task lists from others within the same account.

You can register a domain by using the AWS Management Console or by using the `RegisterDomain` action in the Amazon SWF API. The following example uses the API.

```
https://swf.us-east-1.amazonaws.com
```

```
RegisterDomain
{
  "name" : "867530901",
  "description" : "music",
  "workflowExecutionRetentionPeriodInDays" : "60"
}
```

The parameters are specified in JavaScript Object Notation (JSON) format. Here, the retention period is set to 60 days. During the retention period, all information about the workflow execution is available through visibility operations using either the AWS Management Console or the Amazon SWF API.

After registering the domain, you should register the workflow type and the activity types used by the workflow. You need to register the domain first because a registered domain name is part of the required information for registering workflow and activity types.

See Also

- RegisterDomain in the *Amazon Simple Workflow Service API Reference*

Setting Timeout Values in Amazon SWF

Topics

Limits on Timeout Values

Timeout values are always declared in seconds, and can be set to any number of seconds up to a year (31536000 seconds)—the maximum execution limit for any workflow or activity. The special value "NONE" is used to set a timeout parameter to "no timeout", or infinite, but the maximum limit of a year still applies.

Workflow Execution and Decision Task Timeouts

You can set timeout values for your Workflow and Decision tasks when registering the workflow type. For example:

```
https://swf.us-east-1.amazonaws.com
RegisterWorkflowType
{
  "domain": "867530901",
  "name": "customerOrderWorkflow",
  "version": "1.0",
  "description": "Handle customer orders",
  "defaultTaskStartToCloseTimeout": "600",
  "defaultExecutionStartToCloseTimeout": "3600",
  "defaultTaskList": { "name": "mainTaskList" },
```

```
  "defaultChildPolicy": "TERMINATE"
}
```

This workflow type registration sets the defaultTaskStartToCLoseTimeout to 600 seconds (10 minutes), and defaultExecutionStartToCloseTimeout to 3600 seconds (1 hour).

For more information about workflow type registration, see Registering a Workflow Type (p. 74), and RegisterWorkflowType in the *Amazon Simple Workflow Service API Reference*.

You can override the value set for `defaultExecutionStartToCloseTimeout` by specifying executionStartToCloseTimeout in StartWorkflowExecution.

Activity Task Timeouts

You can set timeout values for your activity tasks when registering the activity type. For example:

```
https://swf.us-east-1.amazonaws.com
RegisterActivityType
{
  "domain": "867530901",
  "name": "activityVerify",
  "version": "1.0",
  "description": "Verify the customer credit",
  "defaultTaskStartToCloseTimeout": "600",
  "defaultTaskHeartbeatTimeout": "120",
  "defaultTaskList": { "name": "mainTaskList" },
  "defaultTaskScheduleToStartTimeout": "1800",
  "defaultTaskScheduleToCloseTimeout": "5400"
}
```

This activity type registration sets the defaultTaskStartToCloseTimeout to 600 seconds (10 minutes), the defaultTaskHeartbeatTimeout to 120 seconds (2 minutes), the defaultTaskScheduleToStartTimeout to 1800 seconds (30 minutes) and defaultTaskScheduleToCloseTimeout to 5400 seconds (1.5 hours).

For more information about activity type registration, see Registering an Activity Type (p. 75), and RegisterActivityType in the *Amazon Simple Workflow Service API Reference*.

You can override the value set for `defaultTaskStartToCloseTimeout` by specifying taskStartToCloseTimeout in StartWorkflowExecution.

See Also

- Timeout Types (p. 118)

Registering a Workflow Type with Amazon SWF

The example discussed in this section registers a workflow type using the Amazon SWF API. The name and version that you specify during registration form a unique identifier for the workflow type. The specified domain must have already been registered using the RegisterDomain API.

The timeout parameters in the following example are duration values specified in seconds. For the `defaultTaskStartToCloseTimeout` parameter, you can use the duration specifier "NONE" to indicate no timeout. However, you cannot specify a value of "NONE" for

`defaultExecutionStartToCloseTimeout`; there is a one-year maximum limit on the time that a workflow execution can run. Exceeding this limit always causes the workflow execution to time out. If you specify a value for `defaultExecutionStartToCloseTimeout` that is greater than one year, the registration will fail.

```
https://swf.us-east-1.amazonaws.com
RegisterWorkflowType
{
  "domain" : "867530901",
  "name" : "customerOrderWorkflow",
  "version" : "1.0",
  "description" : "Handle customer orders",
  "defaultTaskStartToCloseTimeout" : "600",
  "defaultExecutionStartToCloseTimeout" : "3600",
  "defaultTaskList" : { "name": "mainTaskList" },
  "defaultChildPolicy" : "TERMINATE"
}
```

See Also

- RegisterWorkflowType in the *Amazon Simple Workflow Service API Reference*

Registering an Activity Type with Amazon SWF

The following example registers an activity type by using the Amazon SWF API. The name and version that you specify during registration form a unique identifier for the activity type within the domain. The specified domain must have already been registered using the `RegisterDomain` action.

The timeout parameters in this example are duration values specified in seconds. You can use the duration specifier "NONE" to indicate no timeout.

```
https://swf.us-east-1.amazonaws.com
RegisterActivityType
{
  "domain" : "867530901",
  "name" : "activityVerify",
  "version" : "1.0",
  "description" : "Verify the customer credit",
  "defaultTaskStartToCloseTimeout" : "600",
  "defaultTaskHeartbeatTimeout" : "120",
  "defaultTaskList" : { "name" : "mainTaskList" },
  "defaultTaskScheduleToStartTimeout" : "1800",
  "defaultTaskScheduleToCloseTimeout" : "5400"
}
```

See Also

- RegisterActivityType in the *Amazon Simple Workflow Service API Reference*

Developing an Activity Worker in Amazon SWF

An activity worker provides the implementation of one or more activity types. An activity worker communicates with Amazon SWF to receive activity tasks and perform them. You can have a fleet of multiple activity workers performing activity tasks of the same activity type.

Amazon SWF makes an activity task available to activity workers when the decider schedules the activity task. When a decider schedules an activity task, it provides the data (which you determine) that the activity worker needs to perform the activity task. Amazon SWF inserts this data into the activity task before sending it to the activity worker.

Activity workers are managed by you. They can be written in any language. A worker can be run anywhere, as long as it can communicate with Amazon SWF through the API. Because Amazon SWF provides all the information needed to perform an activity task, all activity workers can be stateless. Statelessness enables your workflows to be highly scalable; to handle increased capacity requirements, simply add more activity workers.

This section explains how to implement an activity worker. The activity workers should repeatedly do the following.

1. Poll Amazon SWF for an activity task.
2. Begin performing the task.
3. Periodically report a heartbeat to Amazon SWF if the task is long-lived.
4. Report that the task completed or failed and return the results to Amazon SWF.

Topics

Polling for Activity Tasks

To perform activity tasks, each activity worker must poll Amazon SWF by periodically calling the `PollForActivityTask` action.

In the following example, the activity worker **`ChargeCreditCardWorker01`** polls for a task on the task list, **`ChargeCreditCard-v0.1`**. If no activity tasks are available, after 60 seconds, Amazon SWF sends back an empty response. An empty response is a `Task` structure in which the value of the `taskToken` is an empty string.

```
https://swf.us-east-1.amazonaws.com
PollForActivityTask
{
  "domain" : "867530901",
  "taskList" : { "name": "ChargeCreditCard-v0.1" },
  "identity" : "ChargeCreditCardWorker01"
}
```

If an activity task becomes available, Amazon SWF returns it to the activity worker. The task contains the data that the decider specifies when it schedules the activity.

After an activity worker receives an activity task, it is ready to perform the work. The next section provides information on performing an activity task.

Performing the Activity Task

After receiving an activity task, the activity worker is ready to perform it.

To perform an activity task

1. Program your activity worker to interpret the content in the input field of the task. This field contains the data specified by the decider when the task was scheduled.
2. Program the activity worker to begin processing the data and executing your logic.

The next section describes how to program your activity workers to provide status updates to Amazon SWF for long running activities.

Reporting Activity Task Heartbeats

If a heartbeat timeout was registered with the activity type, then the activity worker must record a heartbeat before the heartbeat timeout is exceeded. If an activity task does not provide a heartbeat within the timeout, the task times out, Amazon SWF closes it and schedules a new decision task to inform a decider of the timeout. The decider can then reschedule the activity task or take another action.

If, after timing out, the activity worker attempts to contact Amazon SWF, such as by calling `RespondActivityTaskCompleted`, Amazon SWF will return an `UnknownResource` fault.

This section describes how to provide an activity heartbeat.

To record an activity task heartbeat, program your activity worker to call the `RecordActivityTaskHeartbeat` action. This action also provides a string field that you can use to store free-form data to quantify progress in whatever way works for your application.

In this example, the activity worker reports heartbeat to Amazon SWF and uses the details field to report that the activity task is 40 percent complete. To report heartbeat, the activity worker must specify the task token of the activity task.

```
https://swf.us-east-1.amazonaws.com
RecordActivityTaskHeartbeat
{
  "taskToken" : "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "details" : "40"
}
```

This action does not in itself create an event in the workflow execution history; however, if the task times out, the workflow execution history will contain a `ActivityTaskTimedOut` event that contains the information from the last heartbeat generated by the activity worker.

Completing or Failing an Activity Task

After executing a task, the activity worker should report whether the activity task completed or failed.

Completing an Activity Task

To complete an activity task, program the activity worker to call the `RespondActivityTaskCompleted` action after it successfully completes an activity task, specifying the task token.

In this example, the activity worker indicates that the task completed successfully.

```
https://swf.us-east-1.amazonaws.com
RespondActivityTaskCompleted
{
  "taskToken": "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "results": "40"
}
```

When the activity completes, Amazon SWF schedules a new decision task for the workflow execution with which the activity is associated.

Program the activity worker to poll for another activity task after it has completed the task at hand. This creates a loop where the activity worker continuously polls for and completes tasks.

If the activity does not respond within the *StartToCloseTimeout* period, or if *ScheduleToCloseTimeout* has been exceeded, Amazon SWF times out the activity task and schedules a decision task. This enables a decider to take an appropriate action, such as rescheduling the task.

For example, if an Amazon EC2 instance is executing an activity task and the instance fails before the task is complete, the decider receives a timeout event in the workflow execution history. If the activity task is using a heartbeat, the decider receives the event when the task fails to deliver the next heartbeat after the Amazon EC2 instance fails. If not, the decider eventually receives the event when the activity task fails to complete before it hits one of its overall timeout values. It is then up to the decider to re-assign the task or take some other action.

Failing an Activity Task

If an activity worker cannot perform an activity task for some reason, but it can still communicate with Amazon SWF, you can program it to fail the task.

To program an activity worker to fail an activity task, program the activity worker to call the `RespondActivityTaskFailed` action that specifies the task token of the task.

```
https://swf.us-east-1.amazonaws.com
RespondActivityTaskFailed
{
  "taskToken" : "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "reason" : "CC-Invalid",
  "details" : "Credit Card Number Checksum Failed"
}
```

As the developer, you define the values that are stored in the reason and details fields. These are free-form strings; you can use any error code conventions that serve your application. Amazon SWF does not process these values. However, Amazon SWF may display these values in the console.

When an activity task is failed, Amazon SWF schedules a decision task for the workflow execution with which the activity task is associated to inform the decider of the failure. Program your decider to handle

failed activities, such as by rescheduling the activity or failing the workflow execution, depending on the nature of the failure.

Launching Activity Workers

To launch activity workers, package your logic into an executable that you can use on your activity worker platform. For example, you might package your activity code as a Java executable that you can run on both Linux and Windows servers.

Once launched, your workers start polling for tasks. Until the decider schedules activity tasks, though, these polls time out with no tasks and your workers just continue polling.

Because polls are outbound requests, activity worker can run on any network that has access to the Amazon SWF endpoint.

You can launch as many activity workers as you like. As the decider schedules activity tasks, Amazon SWF automatically distributes the activity tasks to the polling activity workers.

Developing Deciders in Amazon SWF

A decider is an implementation of the coordination logic of your workflow type that runs during the execution of your workflow. You can run multiple deciders for a single workflow type.

Because the execution state for a workflow execution is stored in its workflow history, deciders can be stateless. Amazon SWF maintains the workflow execution history and provides it to a decider with each decision task. This enables you to dynamically add and remove deciders as necessary, which makes the processing of your workflows highly scalable. As the load on your system grows, you simply add more deciders to handle the increased capacity. Note, however, that there can be only one decision task open at any time for a given workflow execution.

Every time a state change occurs for a workflow execution, Amazon SWF schedules a decision task. Each time a decider receives a decision task, it does the following:

- Interprets the workflow execution history provided with the decision task
- Applies the coordination logic based on the workflow execution history and makes decisions on what to do next. Each decision is represented by a Decision structure
- Completes the decision task and provides a list of decisions to Amazon SWF.

This section describes how to develop a decider, which involves:

- Programming your decider to poll for decision tasks
- Programming your decider to interpret the workflow execution history and make decisions
- Programming your decider to respond to a decision task.

The examples in this section show how you might program a decider for the e-commerce example workflow.

You can implement the decider in any language that you like and run it anywhere, as long as it can communicate with Amazon SWF through its service API.

Topics

Defining Coordination Logic

The first thing to do when developing a decider is to define the coordination logic. In the e-commerce example, coordination logic that schedules each activity after the previous activity completes might look similar to the following:

```
IF lastEvent = "StartWorkflowInstance"
 addToDecisions ScheduleVerifyOrderActivity

ELSIF lastEvent = "CompleteVerifyOrderActivity"
 addToDecisions ScheduleChargeCreditCardActivity

ELSIF lastEvent = "CompleteChargeCreditCardActivity"
 addToDecisions ScheduleCompleteShipOrderActivity

ELSIF lastEvent = "CompleteShipOrderActivity"
 addToDecisions ScheduleRecordOrderCompletion

ELSIF lastEvent = "CompleteRecordOrderCompletion"
 addToDecisions CloseWorkflow

ENDIF
```

The decider applies the coordination logic to the workflow execution history, and creates a list of decisions when completing the decision task using the `RespondDecisionTaskCompleted` action.

Polling for Decision Tasks

Each decider polls for decision tasks. The decision tasks contain the information that the decider uses to generate decisions such as scheduling activity tasks. To poll for decision tasks, the decider uses the `PollForDecisionTask` action.

In this example, the decider polls for a decision task, specifying the **`customerOrderWorkflow-0.1`** tasklist.

```
https://swf.us-east-1.amazonaws.com
PollForDecisionTask
{
  "domain": "867530901",
  "taskList": {"name": "customerOrderWorkflow-v0.1"},
  "identity": "Decider01",
  "maximumPageSize": 50,
  "reverseOrder": true
}
```

If a decision task is available from the task list specified, Amazon SWF returns it immediately. If no decision task is available, Amazon SWF holds the connection open for up to 60 seconds, and returns a task as

soon as it becomes available. If no task becomes available, Amazon SWF returns an empty response. An empty response is a `Task` structure in which the value of `taskToken` is an empty string. Make sure to program your decider to poll for another task if it receives an empty response.

If a decision task is available, Amazon SWF returns a response that contains the decision task as well as a paginated view of the workflow execution history.

In this example, the type of the most recent event indicates the workflow execution started and the input element contains the information needed to perform the first task.

```
{
  "events": [
    {
      "decisionTaskStartedEventAttributes": {
        "identity": "Decider01",
        "scheduledEventId": 2
      },
      "eventId": 3,
      "eventTimestamp": 1326593394.566,
      "eventType": "DecisionTaskStarted"
    }, {
      "decisionTaskScheduledEventAttributes": {
        "startToCloseTimeout": "600",
        "taskList": { "name": "specialTaskList" }
      },
      "eventId": 2,
      "eventTimestamp": 1326592619.474,
      "eventType": "DecisionTaskScheduled"
    }, {
      "eventId": 1,
      "eventTimestamp": 1326592619.474,
      "eventType": "WorkflowExecutionStarted",
      "workflowExecutionStartedEventAttributes": {
        "childPolicy" : "TERMINATE",
        "executionStartToCloseTimeout" : "3600",
        "input" : "data-used-decider-for-first-task",
        "parentInitiatedEventId": 0,
        "tagList" : ["music purchase", "digital", "ricoh-the-dog"],
        "taskList": { "name": "specialTaskList" },
        "taskStartToCloseTimeout": "600",
        "workflowType": {
          "name": "customerOrderWorkflow",
          "version": "1.0"
        }
      }
    }
  ],
  ...
}
```

After receiving the workflow execution history, the decider interprets history and makes decisions based on its coordination logic.

Because the number of workflow history events for a single workflow execution might be large, the result returned might be split up across a number of pages. To retrieve subsequent pages, make additional calls to `PollForDecisionTask` using the *nextPageToken* returned by the initial call. Note that you do

not call `GetWorkflowExecutionHistory` with this *nextPageToken*. Instead, call `PollForDecisionTask` again.

Applying the Coordination Logic

After the decider receives a decision task, program it to interpret the workflow execution history to determine what has happened so far. Based on this, it should generate a list of decisions.

In the e-commerce example, we are concerned only with the last event in the workflow history, so we define the following logic.

```
IF lastEvent = "StartWorkflowInstance"
 addToDecisions ScheduleVerifyOrderActivity

ELSIF lastEvent = "CompleteVerifyOrderActivity"
 addToDecisions ScheduleChargeCreditCardActivity

ELSIF lastEvent = "CompleteChargeCreditCardActivity"
 addToDecisions ScheduleCompleteShipOrderActivity

ELSIF lastEvent = "CompleteShipOrderActivity"
 addToDecisions ScheduleRecordOrderCompletion

ELSIF lastEvent = "CompleteRecordOrderCompletion"
 addToDecisions CloseWorkflow

ENDIF
```

If the lastEvent is `CompleteVerifyOrderActivity`, you would add the `ScheduleChargeCreditCardActivity` activity to the list of decisions.

After the decider determines the decision(s) to make, it can respond to Amazon SWF with appropriate decisions.

Responding with Decisions

After interpreting the workflow history and generating a list of decisions, the decider is ready to respond back to Amazon SWF with those decisions.

Program your decider to extract the data that it needs from the workflow execution history, then create decisions that specify the next appropriate actions for the workflow. The decider transmits these decision back to Amazon SWF using the `RespondDecisionTaskCompleted` action. See the *Amazon Simple Workflow Service API Reference* for a list of the available decision types.

In the e-commerce example, when the decider responds with the set of decisions that it generated, it also includes the credit card input from the workflow execution history. The activity worker then has the information it needs to perform the activity task.

When all activities in the workflow execution are complete, the decider closes the workflow execution.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "taskToken" : "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
```

```
  "decisions" : [
    {
      "decisionType" :"ScheduleActivityTask",
      "scheduleActivityTaskDecisionAttributes" : {
        "control" :"OPTIONAL_DATA_FOR_DECIDER",
        "activityType" : {
          "name" :"ScheduleChargeCreditCardActivity",
          "version" :"1.1"
        },
        "activityId" :"3e2e6e55-e7c4-beef-feed-aa815722b7be",
        "scheduleToCloseTimeout" :"360",
        "taskList" : { "name" :"CC_TASKS" },
        "scheduleToStartTimeout" :"60",
        "startToCloseTimeout" :"300",
        "heartbeatTimeout" :"60",
        "input" : "4321-0001-0002-1234: 0212 : 234"
      }
    }
  ]
}
```

Closing a Workflow Execution

When the decider determines that the business process is complete, that is, that there are no more activities to perform, the decider generates a decision to close the workflow execution.

To close a workflow execution, program your decider to interpret the events in the workflow history to determine what has happened in the execution so far and see if the workflow execution should be closed.

If the workflow has completed successfully, then close the workflow execution by calling `RespondDecisionTaskCompleted` with the `CompleteWorkflowExecution` decision. Alternatively, you can fail an erroneous execution using the `FailWorkflowExecution` decision.

In the e-commerce example, the decider reviews the history and based on the coordination logic adds a decision to close the workflow execution to its list of decisions, and initiates a `RespondDecisionTaskCompleted` action with a close workflow decision.

> **Note**
> There are some cases where closing a workflow execution fails. For example, if a signal is received while the decider is closing the workflow execution, the close decision will fail. To handle this possibility, ensure that the decider continues polling for decision tasks. Also, ensure that the decider that receives the next decision task responds to the event—in this case, a signal—that prevented the execution from closing.

You might also support cancellation of workflow executions. This could be especially useful for long running workflows. To support cancellation, your decider should handle the `WorkflowExecutionCancelRequested` event in the history. This event indicates that cancellation of the execution has been requested. Your decider should perform appropriate clean-up actions, such as canceling ongoing activity tasks, and closing the workflow calling the `RespondDecisionTaskCompleted` action with the `CancelWorkflowExecution` decision.

The following example calls `RespondDecisionTaskCompleted` to specify that the current workflow execution is canceled.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
```

```
{
  "taskToken" : "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "decisions" : [
    {
      "decisionType":"CancelWorkflowExecution",
      "CancelWorkflowExecutionAttributes":{
        "Details": "Customer canceled order"
      }
    }
  ]
}
```

Amazon SWF checks to ensure that the decision to close or cancel the workflow execution is the last decision sent by the decider. That is, it isn't valid to have a set of decisions in which there are decisions after the one that closes the workflow.

Launching Deciders

After completing decider development, you are ready to launch one or more deciders.

To launch deciders, package your coordination logic into an executable that you can use on your decider platform. For example, you might package your decider code as a Java executable that you can run on both Linux and Windows computers.

Once launched, your deciders should start polling Amazon SWF for tasks. Until you start workflow executions and Amazon SWF schedules decision tasks, these polls will time out and get empty responses. An empty response is a `Task` structure in which the value of `taskToken` is an empty string. Your deciders should simply continue to poll.

Amazon SWF ensures that only one decision task can be active for a workflow execution at any time. This prevents issues such as conflicting decisions. Additionally, Amazon SWF ensures that a single decision task is assigned to a single decider, regardless of the number of deciders that are running.

If something occurs that generates a decision task while a decider is processing another decision task, Amazon SWF queues the new task until the current task completes. After the current task completes, Amazon SWF makes the new decision task available. Also, decision tasks are batched in the sense that, if multiple activities complete while a decider is processing a decision task, Amazon SWF will create only a single new decision task to account for the multiple task completions. However, each task completion will receive an individual event in the workflow execution history.

Because polls are outbound requests, deciders can run on any network that has access to the Amazon SWF endpoint.

In order for workflow executions to progress, one or more deciders must be running. You can launch as many deciders as you like. Amazon SWF supports multiple deciders polling on the same task list.

Starting Workflow Executions with Amazon SWF

You can start a workflow execution of a registered workflow type from any application using the `StartWorkflowExecution` action. When you start the execution you associate an identifier, called the `workflowId`, with it. The `workflowId` can be any string that is appropriate for your application, such as the order number in an order processing application. You cannot use the same `workflowId` for multiple open workflow executions within the same domain. For example, if you start two workflow executions with the `workflowId` **`Customer Order 01`**, the second workflow execution will not start

and the request will fail. You can, however, reuse the `workflowId` of a closed execution. Amazon SWF also associates a unique system generated identifier, called the `runId`, with each workflow execution.

After the workflow and activity types are registered, start the workflow by calling the `StartWorkflowExecution` action. The value of the `input` parameter can be any string specified by the application that is starting the workflow. The `executionStartToCloseTimeout` is the length of time in seconds that the workflow execution can consume from start to close. Exceeding this limit causes the workflow execution to time out. Unlike some of the other timeout parameters in Amazon SWF, you cannot specify a value of "NONE" for this timeout; there is a one-year maximum limit on the time that a workflow execution can run. Similarly, the *taskStartToCloseTimeout* is the length of time in seconds that a decision task associated with this workflow execution can take before timing out.

```
https://swf.us-east-1.amazonaws.com
StartWorkflowExecution
{
  "domain" : "867530901",
  "workflowId" : "20110927-T-1",
  "workflowType" : {
    "name" : "customerOrderWorkflow", "version" : "1.1"
  },
  "taskList" : { "name" : "specialTaskList" },
  "input" : "arbitrary-string-that-is-meaningful-to-the-workflow",
  "executionStartToCloseTimeout" : "1800",
  "tagList" : [ "music purchase", "digital", "ricoh-the-dog" ],
  "taskStartToCloseTimeout" : "1800",
  "childPolicy" : "TERMINATE"
}
```

If the `StartWorkflowExecution` action is successful, Amazon SWF returns the `runId` for the workflow execution. The `runId` uniquely identifies this workflow execution from any other workflow executions currently running under Amazon SWF. Save the `runId` in case you later need to specify this workflow execution in a call to Amazon SWF. For example, you would use the `runId` if you later needed to send a signal to the workflow execution.

```
{"runId": "9ba33198-4b18-4792-9c15-7181fb3a8852"}
```

Handling Errors in Amazon SWF

There are a number of different types of errors that can occur during the course of a workflow execution.

Topics

Validation Errors

Validation errors occur when a request to Amazon SWF fails because it is not properly formed or it contains invalid data. In this context, a request could be an action such as `DescribeDomain` or it could be a decision such as `StartTimer`. If the request is an action, Amazon SWF returns an error code in the response. Check this error code as it may provide information about what aspect of the request caused the failure. For example, one or more of the arguments passed with the request might be invalid. For a list of common error codes, go to the topic for the action in the *Amazon Simple Workflow Service API Reference*.

If the request that failed is a decision, an appropriate event will be listed in the workflow execution history. For example, if the `StartTimer` decision failed, you would see a `StartTimerFailed` event in the history. The decider should check for these events when it receives the history in response to `PollForDecisionTask` or `GetWorkflowExecutionHistory`. Below is a list of possible decision failure events that can occur when the decision is not correctly formed or contains invalid data.

Errors in Enacting Actions or Decisions

Even if the request is properly formed, errors may occur when Amazon SWF attempts to carry out the request. In these cases, one of the following events in the history will indicate that an error occurred. Look at the `reason` field of the event to determine the cause of failure.

- CancelTimerFailed
- RequestCancelActivityTaskFailed
- RequestCancelExternalWorkflowExecutionFailed
- ScheduleActivityTaskFailed
- SignalExternalWorkflowExecutionFailed
- StartChildWorkflowExecutionFailed
- StartTimerFailed

Timeouts

Deciders, activity workers, and workflow executions all operate within the constraints of timeout periods. In this type of error, a task or a child workflow times out. An event will appear in the history that describes the timeout. The decider should handle this event by, for example, rescheduling the task or restarting the child workflow. For more information about timeouts, see Timeout Types (p. 118)

- ActivityTaskTimedOut
- ChildWorkflowExecutionTimedOut
- DecisionTaskTimedOut
- WorkflowExecutionTimedOut

Errors raised by user code

Examples of this type of error condition are activity task failures and child-workflow failures. As with timeout errors, Amazon SWF adds an appropriate event to the workflow execution history. The decider should handle this event, possibly by rescheduling the task or restarting the child workflow.

- ActivityTaskFailed
- ChildWorkflowExecutionFailed

Errors related to closing a workflow execution

Deciders may also see the following events if they attempt to close a workflow that has a pending decision task.

- FailWorkflowExecutionFailed
- CompleteWorkFlowExecutionFailed
- ContinueAsNewWorkflowExecutionFailed
- CancelWorkflowExecutionFailed

For more information about any of the events listed above, see History Event in the Amazon SWF API Reference.

Using the Amazon SWF Console

The Amazon Simple Workflow Service (Amazon SWF) console provides an alternative way to configure, initiate, and manage workflow executions.

With the Amazon SWF console, you can:

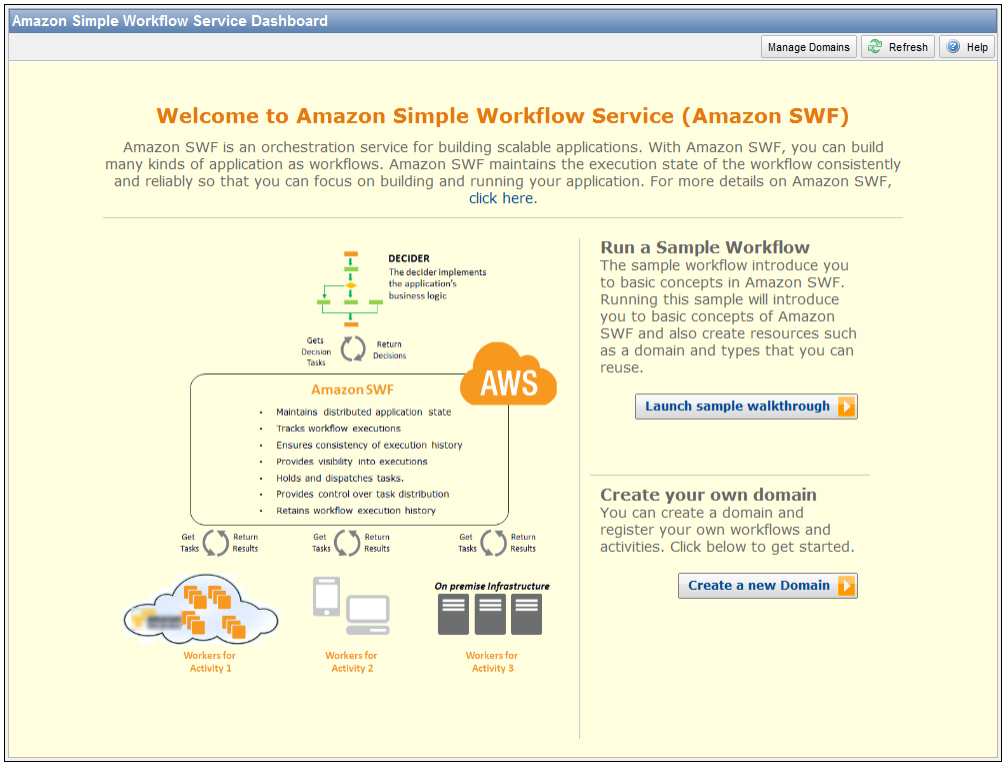
- Register workflow domains.
- Register workflow types.
- Register activity types.
- Initiate workflow executions.
- View information about pending tasks.
- View running workflow executions.
- Cancel, terminate, and send signals to running workflow executions.
- Restart closed workflow executions.

The Amazon SWF console is part of the larger AWS Console experience, which you can access by signing in at http://aws.amazon.com. The sign-in link is located in the upper-right corner of the page.

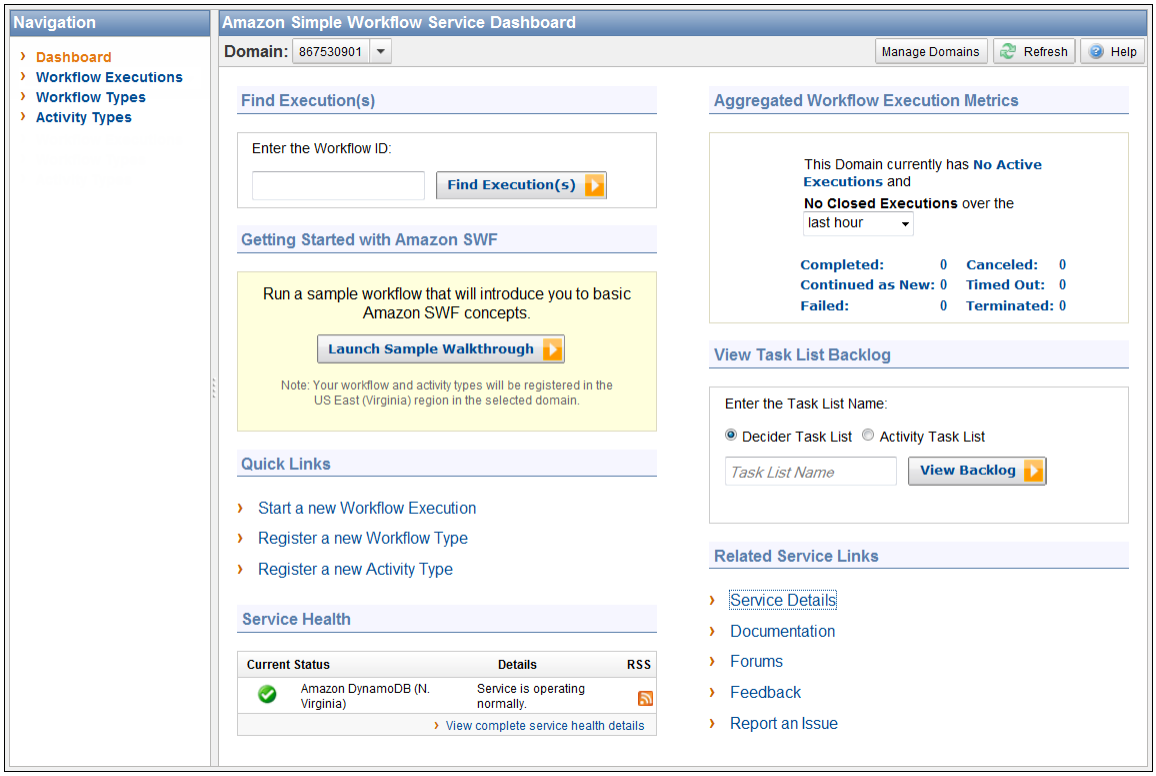
Topics

Amazon Simple Workflow Service Dashboard

The following image shows the **Amazon Simple Workflow Service Dashboard** area of the Amazon SWF console as it looks when no domains are registered.



When at least one domain is registered, the **Amazon Simple Workflow Service Dashboard** displays full functionality.



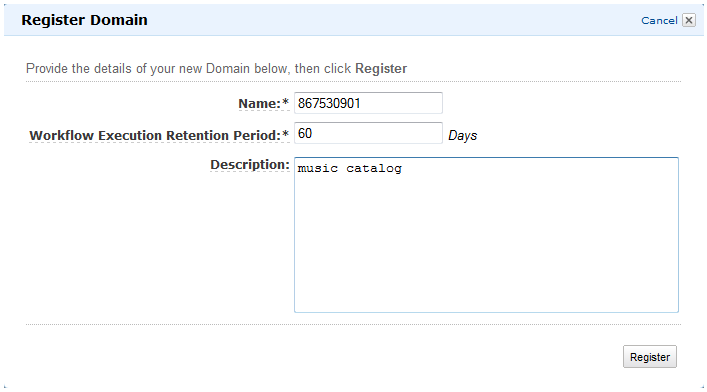
Registering an Amazon SWF Domain

Until at least one domain is registered, domain registration is the only functionality available from the console.

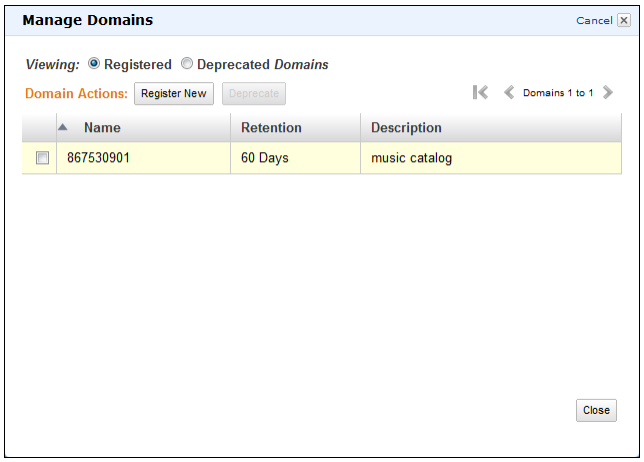
To register an Amazon SWF domain using the console

1. If no domains have been registered, in the center of the main pane, click **Register a New Domain**.

   If at least one domain is registered, in the dashboard view, click the **Manage Domains** button, and then in the **Manage Domains** dialog box, click **Register New**.
2. In the **Register Domain** dialog box, enter a Name, Retention Period, and Description. These values correspond to the similarly-named parameters in the **RegisterDomain** action.



3. Click **Register**.
4. After the domain is registered, the console displays the **Manage Domains** dialog box.



Registering a Workflow Type

You can register workflow types using the Amazon Simple Workflow console. You are not able to register a workflow type until at least one domain is registered.

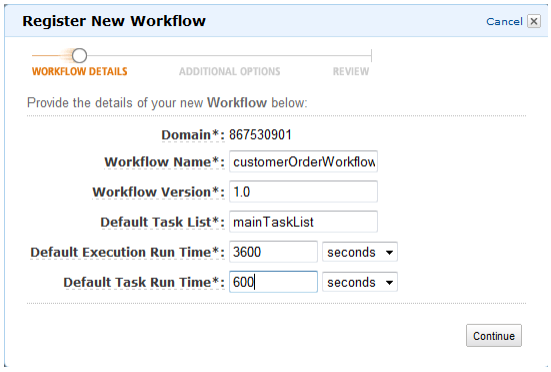
To register an Amazon SWF workflow type using the console

1. In the **Amazon Simple Workflow Service Dashboard**, under **Quick Links**, click **Register New Workflow Type**.

   In the **Workflow Details** dialog box, enter the following information.

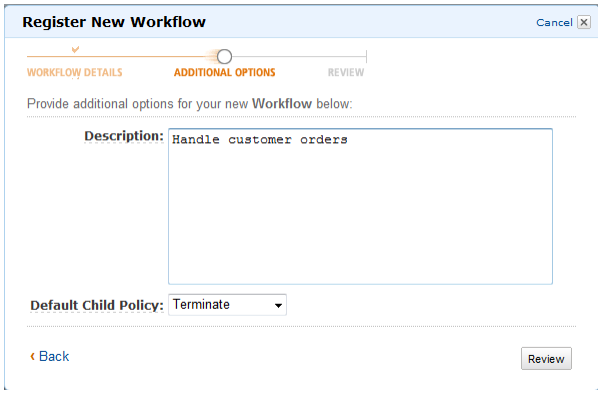
   - Domain
   - Workflow Name
   - Workflow Version
   - Default Task List
   - Default Execution Run Time
   - Default Task Run Time

   Fields marked with an asterisk ("*") are required.

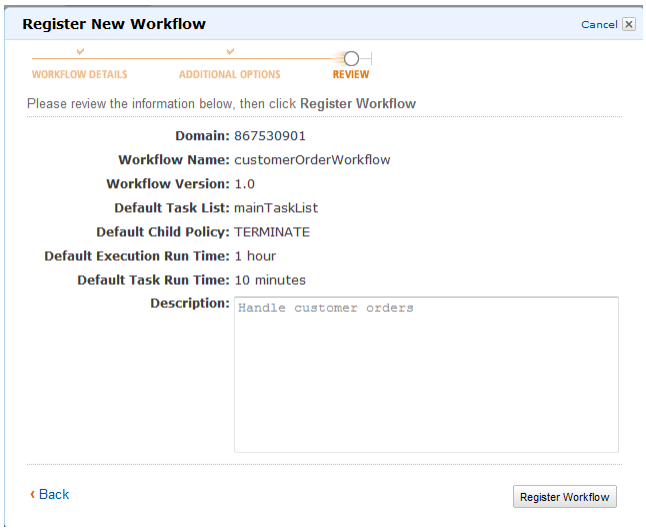


   Click **Continue**.

2. In the **Additional Options** dialog box, enter a **Description** and specify a **Default Child Policy**. Click **Review**.



3. In the **Review** dialog box, review the information that you entered in the previous dialog boxes. If the information is correct, click **Register Workflow**. Otherwise, click **Back** to change the information.



Registering an Activity Type

You can register activity types using the Amazon Simple Workflow Service console. You are not able to register an activity type until at least one domain is registered.

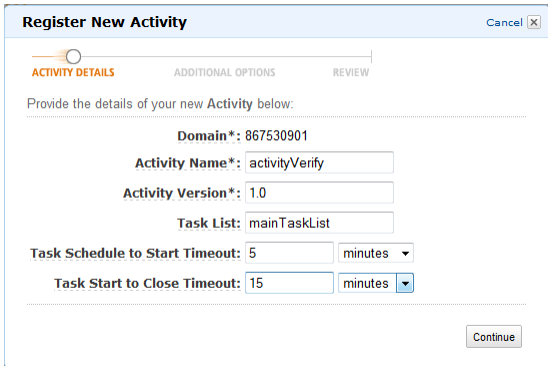
To register an Amazon SWF activity type using the console

1. In the **Amazon Simple Workflow Service Dashboard**, under **Quick Links**, click **Register New Activity Type**.

   In the **Activity Details** dialog box, enter the following information.

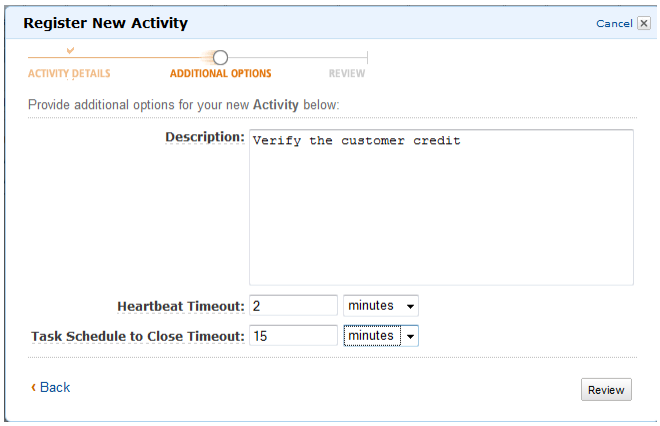
   - Domain
   - Activity Name
   - Activity Version
   - Default Task List
   - Task Schedule to Start Timeout
   - Task Start to Close Timeout

   Fields marked with an asterisk ("*") are required.

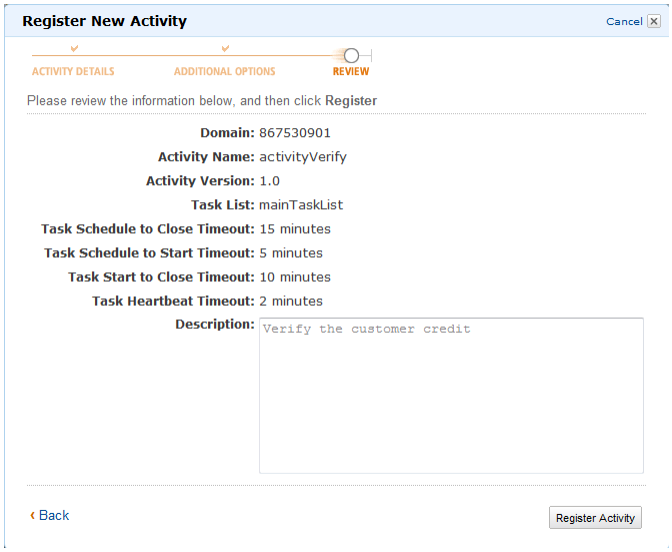


Click **Continue**.

2. In the **Additional Options** dialog box, enter a **Description** and specify a **Heartbeat Timeout** and a **Task Schedule to Close Timeout**. Click **Review**.



3. In the **Review** dialog box, review the information that you entered in the previous dialog boxes. If the information is correct, click **Register Activity**. Otherwise, click **Back** to change the information.



Starting a Workflow Execution

You can start a workflow execution from the Amazon Simple Workflow Service console. You are not able to start a workflow execution until you have registered at least one workflow type.

To start a workflow execution using the console

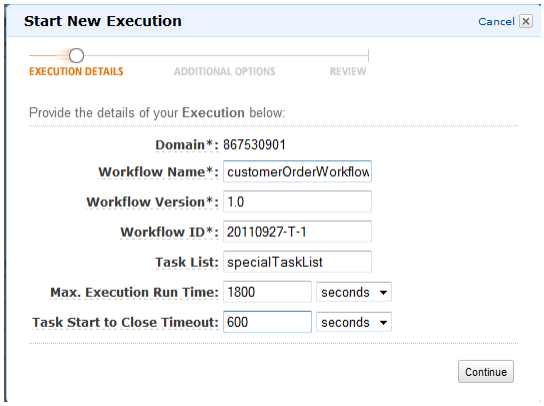
1. In the **Amazon Simple Workflow Service Dashboard**, under **Quick Links**, click **Start a New Workflow Execution**.

   In the **Execution Details** dialog box, enter the following information.

   - Domain

- Workflow Name
- Workflow Version
- Workflow ID
- Task List
- Maximum Execution Run Time
- Task Start to Close Timeout

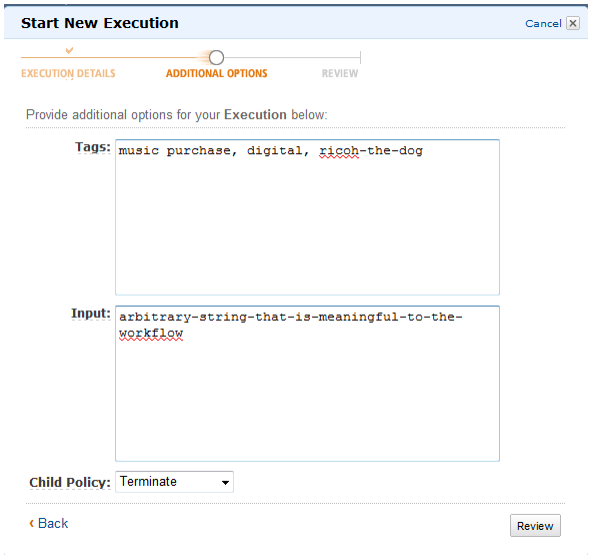
Fields marked with an asterisk ("*") are required.



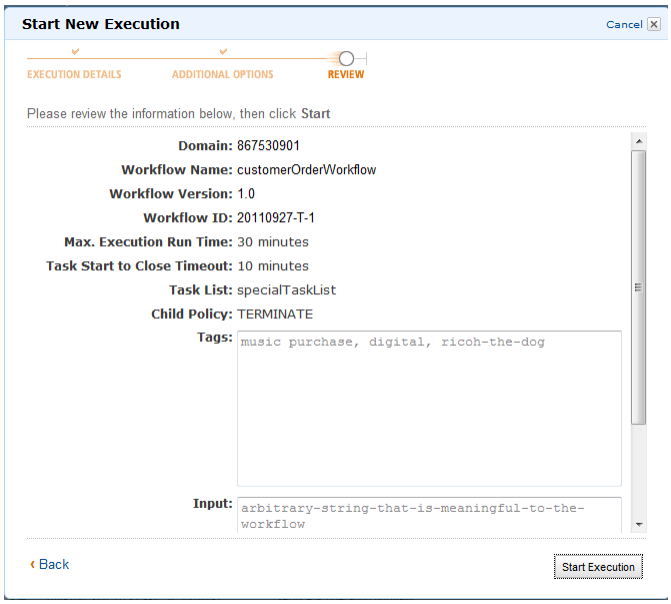
Click **Continue**.

2. In the **Additional Options** dialog box, specify:

   - A set of **Tags** to associate with the workflow execution. You can use these tags to query information about your workflow executions.
   - An **Input** string that is meaningful to the execution. This string is not interpreted by Amazon SWF.
   - A **Child Policy**.



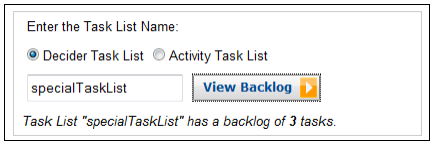
3. In the **Review** dialog box, review the information that you entered in the previous dialog boxes. If the information is correct, click **Start Execution**. Otherwise, click **Back** to change the information.



Viewing Pending Tasks

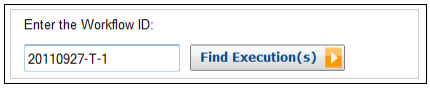
From the Amazon Simple Workflow Service Dashboard, you can view the number of pending tasks that are associated with a particular task list.

1. Select whether the task list is a **Decider Task List** or **Activity Task List**.
2. Enter the task list name in the text box.
3. Click **View Backlog**.



Managing Your Workflow Executions

The **My Workflow Executions** view in the Amazon SWF console provides functionality for managing your workflow executions both those that are currently running, and those that are closed. To access this view, click the **Find Execution(s)** button in the Amazon SWF Dashboard.



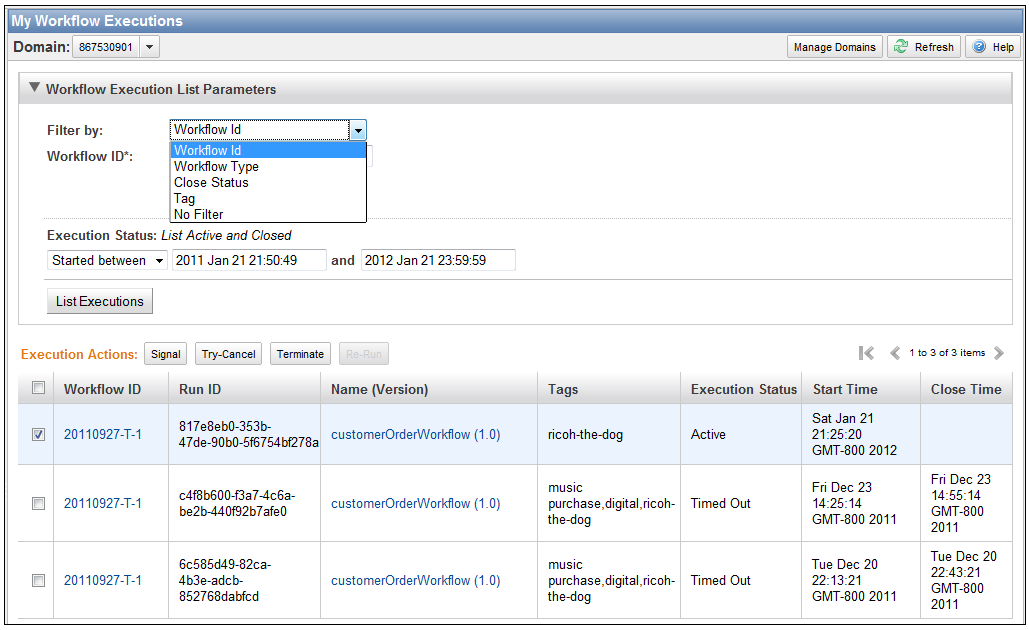
If you first enter a workflow ID, the console will display executions with that workflow ID. Otherwise, if you just click **Find Execution(s)**, the **My Workflow Executions** view will enable you to query for workflow executions based on when they were started, whether they are still running, and their associated metadata. For a given query, you can select from any one of the following types of metadata:

- Workflow ID
- Workflow Type
- Tags
- Close Status

If the workflow execution is closed, the close status is one of the following values, which indicate the circumstance in which the workflow execution closed:

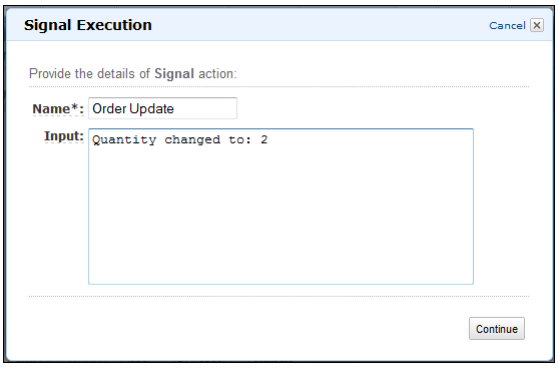
- Completed
- Failed
- Canceled
- Timed Out
- Continued as New

> **Note**
> You must select a domain from the **Domain** drop-down list before you can enumerate workflow executions.

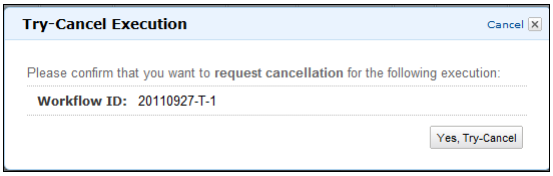


After enumerating a list of workflow executions, you can perform the following operations.

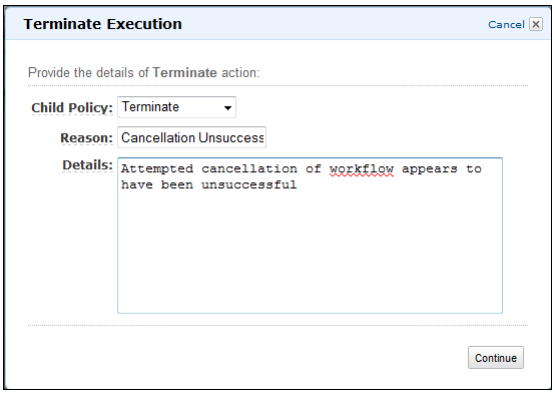
- Signal a workflow execution—that is, send a running workflow execution additional data.



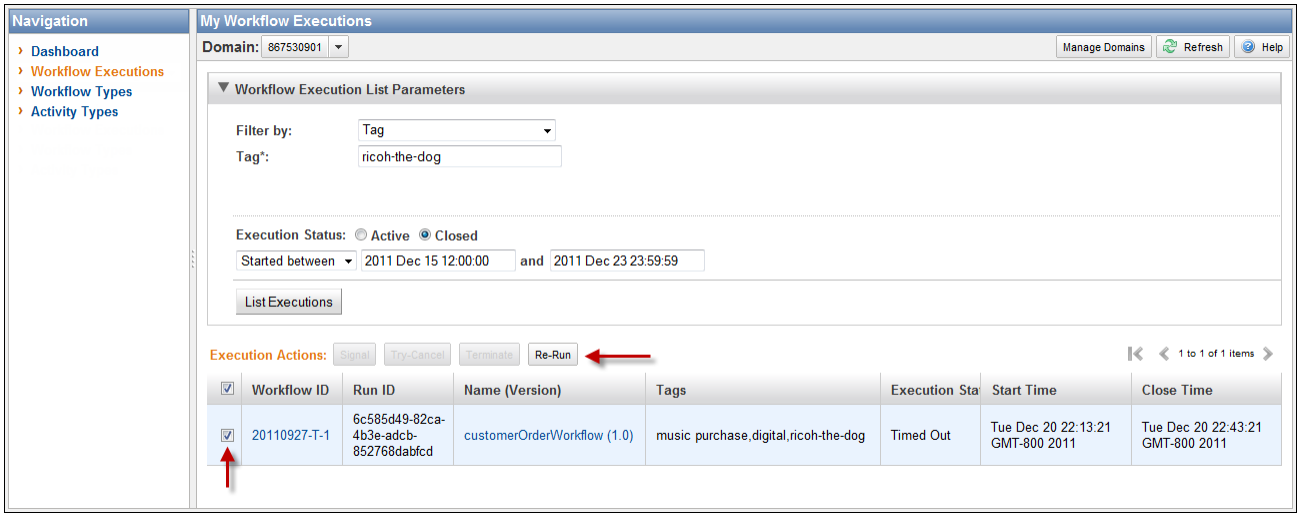
- Try to cancel a workflow execution. It is preferable to cancel a workflow execution rather than terminate it. Canceling provides the workflow execution an opportunity to perform any clean-up tasks and then close properly.



- Terminate a workflow execution. Note that it is preferable to cancel a workflow execution rather than terminate it.



- Re-run a closed workflow execution.



To re-run a closed workflow execution

1. In the list of workflow executions, select the closed execution to re-run. When you select a closed execution, the **Re-Run** button becomes enabled. Click **Re-Run**.

   The **Re-Run Execution** sequence of dialog boxes appears.
2. In the **Execution Details** dialog box, specify the following information. The dialog box has the information from the original execution already filled in.

   - Domain
   - Workflow Name
   - Workflow Version
   - Workflow ID

   By clicking the **Advanced Options** link, you can specify the following additional options.

   - Task List
   - Maximum Execution Run Time
   - Task Start to Close Timeout

   Click **Continue**
3. In the **Additional Options** dialog box, specify an input string for the execution. By clicking the **Advanced Options** link, you can specify **Tags** to associate with this run or the workflow execution as well as change the executions **Child Policy**. As with the previous dialog box, the information from the original execution is already filled in.

   Click **Review**.
4. In the **Review** dialog box, verify that all the information is correct. If the information is correct, click **Re-Run Execution**. Otherwise, click **Back** to change the information.

Viewing Amazon SWF Metrics for CloudWatch using the AWS Management Console

Amazon CloudWatch provides a number of viewable metrics for Amazon SWF workflows and activities. You can view the metrics and set alarms for your Amazon SWF workflow executions using the AWS Management Console. *You must be logged in to the console to proceed*.

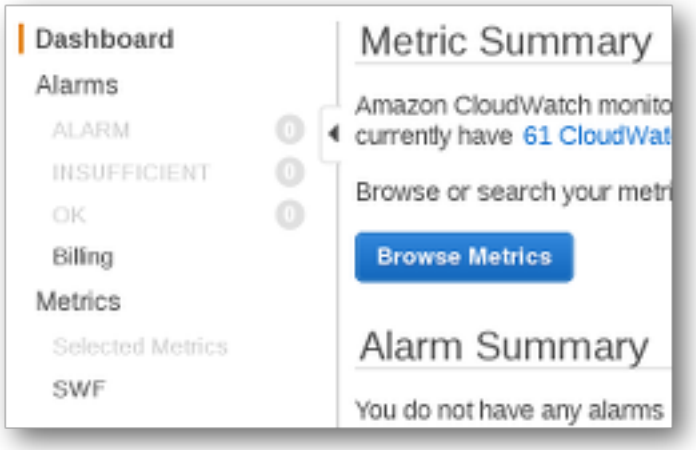
For a description of each of the available metrics, see Amazon SWF Metrics for CloudWatch (p. 121).

Topics

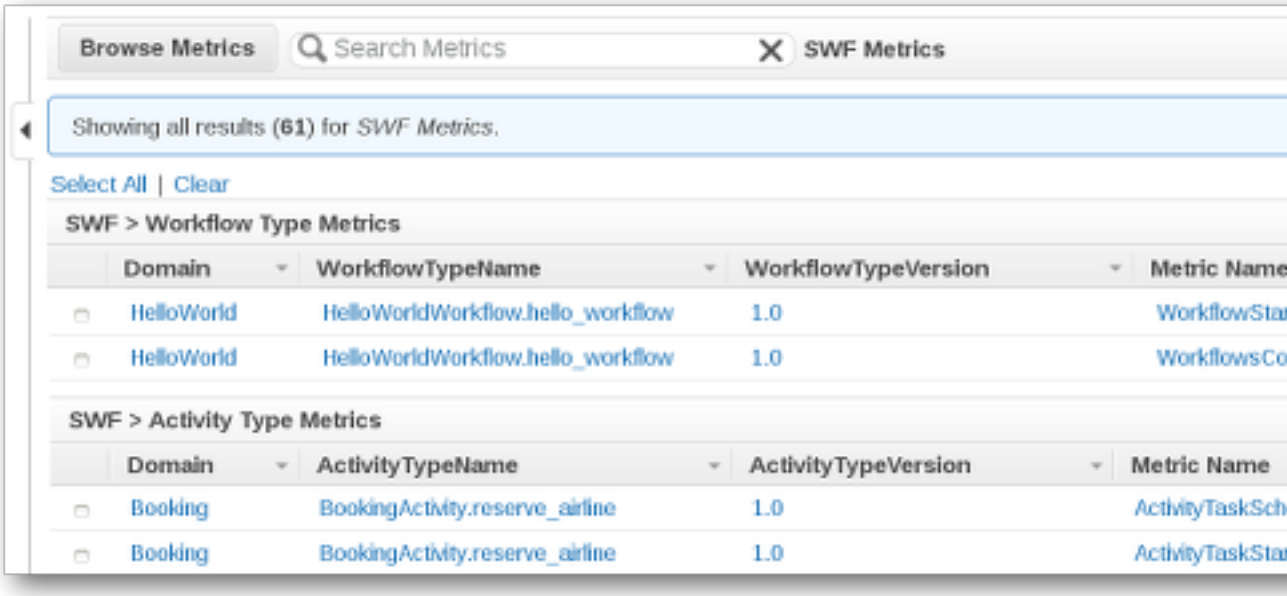
Viewing Metrics

To view your metrics for Amazon SWF

1. Sign in to the AWS Management Console and open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, under **Metrics**, click the **SWF** item.



If you have run any workflow executions recently, you will see two lists of metrics presented: **Workflow Type Metrics** and **Activity Type Metrics**.



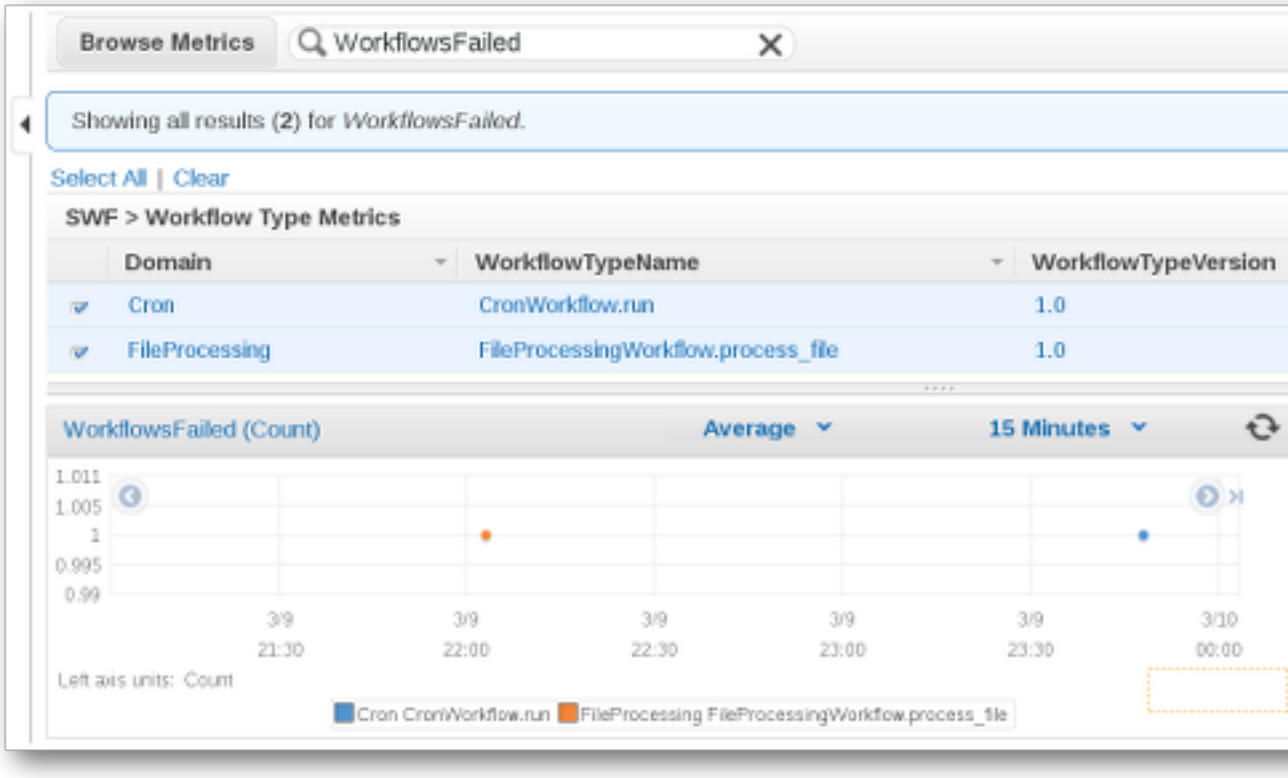
Note
Initially you might only see the **Workflow Type Metrics**; **Activity Type Metrics** are presented in the same view, but you may need to scroll down to see them.

Up to 50 of the most recent metrics will be shown at a time, divided among workflow and activity metrics.

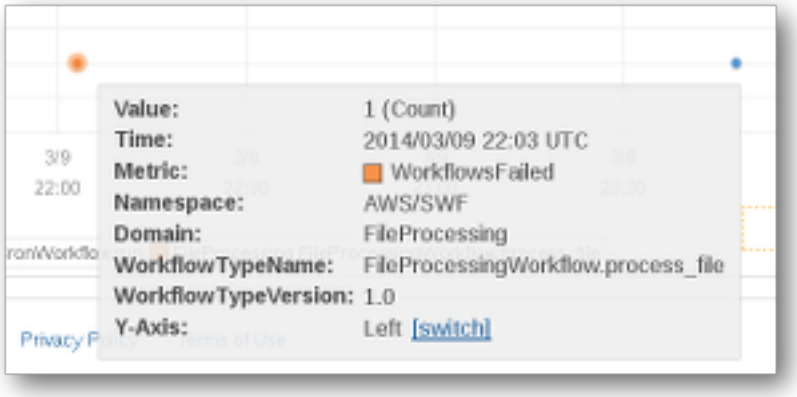
You can use the interactive headings above each column in the list to sort your metrics using any of the provided dimensions. For workflows, the dimensions are **Domain**, **WorkflowTypeName**, **WorkflowTypeVersion**, and **Metric Name**. For activities, the dimensions are **Domain**, **ActivityTypeName**, **ActivityTypeVersion**, and **Metric Name**.

The various types of metrics are described in Amazon SWF Metrics for CloudWatch (p. 121).

You can view graphs for metrics by clicking the boxes next to the metric row in the list, and change the graph parameters using the **Time Range** controls to the right of the graph view.



For details about any point on the graph, place your cursor over the graph point. A detail of the point's dimensions will be shown.



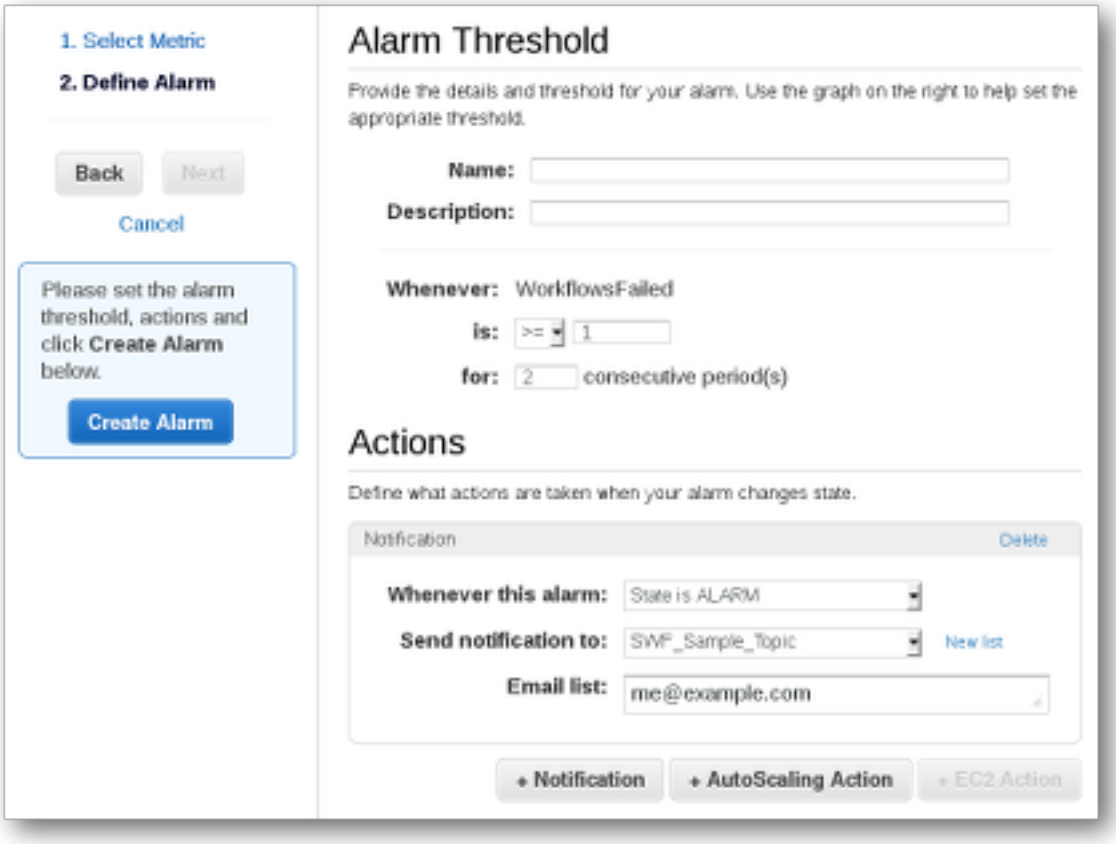
For more information about working with CloudWatch metrics, see Viewing, Graphing, and Publishing Metrics in the *Amazon CloudWatch Developer Guide*.

Setting Alarms

You can use CloudWatch alarms to perform actions such as notifying you when an alarm threshold is reached. For example, you can set an alarm to send a notification to an SNS topic or to send an email when the **WorkflowsFailed** metric rises above a certain threshold.

To set an alarm on any of your metrics

1. Choose a single metric by clicking its box to view its graph.
2. To the right of the graph, in the **Tools** controls, click **Create Alarm**.
3. On the **Define Alarm** screen, enter the alarm threshold value, period parameters, and actions to take.



For more information about setting and using CloudWatch alarms, see Creating Amazon CloudWatch Alarms in the *Amazon CloudWatch Developer Guide*.

Using the AWS CLI with Amazon Simple Workflow Service

Many of the features of Amazon Simple Workflow Service can be accessed from the AWS CLI. The AWS CLI provides an alternative to using Amazon SWF with the AWS Management Console or in some cases, to programming with the Amazon SWF API and the AWS Flow Framework.

For example, you can use the AWS CLI to register a new workflow type:

```
aws swf register-workflow-type --domain DataFrobtzz --name "MySimpleWorkflow"
--workflow-version "v1"
```

You can also list your registered workflow types:

```
aws swf list-workflow-types --domain DataFrobtzz --registration-status REGISTERED
```

The following shows an example of the default output in JSON:

```
{
    "typeInfos": [
        {
            "status": "REGISTERED",
            "creationDate": 1377471607.752,
            "workflowType": {
                "version": "v1",
                "name": "MySimpleWorkflow"
            }
        },
        {
            "status": "REGISTERED",
            "creationDate": 1371454149.598,
            "description": "DataFrobtzz subscribe workflow",
            "workflowType": {
                "version": "v3",
```

```
                "name": "subscribe"
            }
        }
    ]
}
```

The Amazon SWF commands in AWS CLI provide the ability to start and manage workflow executions, poll for activity tasks, record task heartbeats, and more! For a complete list of Amazon SWF commands, with descriptions of the available arguments and examples showing their use, see Amazon SWF commands in the *AWS Command Line Interface Reference*.

The AWS CLI commands follow the Amazon SWF API closely, so you can use the AWS CLI to learn about the underlying Amazon SWF API. You can also use your existing API knowledge to prototype code or perform Amazon SWF actions on the command line.

To learn more about the AWS CLI, see the AWS Command Line Interface User Guide.

Using Advanced Workflow Features in Amazon SWF

Topics

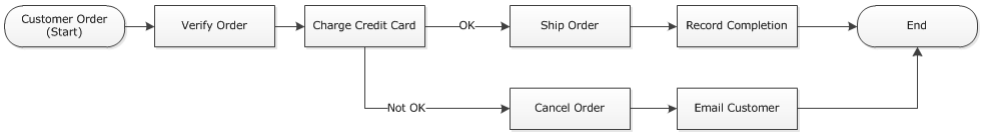
Implementing Exclusive Choice

In some scenarios, you might want to schedule a different set of activities based on the outcome of a previous activity. The exclusive choice pattern enables you to create flexible workflows that meet the complex requirements of your application.

The Amazon Simple Workflow Service (Amazon SWF) does not have a specific exclusive choice action. To use exclusive choice, you simply write your decider logic to make different decisions based on the results of the previous activity. Some applications for exclusive choice include the following:

- Performing cleanup activities if the results of a previous activity were unsuccessful
- Scheduling different activities based on whether the customer purchased a basic or advanced plan
- Performing different customer authentication activities based on the customer's ordering history

In the e-commerce example, you might use exclusive choice to either ship or cancel an order based on the outcome of charging the credit card. In the following figure, the decider schedules the Ship Order and Record Completion activity tasks if the credit card is successfully charged. Otherwise, it schedules the Cancel Order and Email Customer activity tasks.



The decider schedules the `ShipOrder` activity if the credit card is successfully charged. Otherwise, the decider schedules the `CancelOrder` activity.

In this case, program the decider to interpret the history and determine whether the credit card was successfully charged. To do this, you might have logic similar to the following

```
IF lastEvent = "WorkflowExecutionStarted"
 addToDecisions ScheduleActivityTask(ActivityType = "VerifyOrderActivity")

ELSIF lastEvent = "ActivityTaskCompleted"
     AND ActivityType = "VerifyOrderActivity"
 addToDecisions ScheduleActivityTask(ActivityType = "ChargeCreditCardActivity")

#Successful Credit Card Charge Activities
ELSIF lastEvent = "ActivityTaskCompleted"
      AND ActivityType = "ChargeCreditCardActivity"
  addToDecisions ScheduleActivityTask(ActivityType = "ShipOrderActivity")

ELSIF lastEvent = "ActivityTaskCompleted"
     AND ActivityType = "ShipOrderActivity"
  addToDecisions ScheduleActivityTask(ActivityType = "RecordOrderCompletionActiv
ity")

ELSIF lastEvent = "ActivityTaskCompleted"
     AND ActivityType = "RecordOrderCompletionActivity"
 addToDecisions CompleteWorkflowExecution

#Unsuccessful Credit Card Charge Activities
ELSIF lastEvent = "ActivityTaskFailed"
      AND ActivityType = "ChargeCreditCardActivity"
  addToDecisions ScheduleActivityTask(ActivityType = "CancelOrderActivity")

ELSIF lastEvent = "ActivityTaskCompleted"
     AND ActivityType = "CancelOrderActivity"
 addToDecisions ScheduleActivityTask(ActivityType = "EmailCustomerActivity")

ELSIF lastEvent = "ActivityTaskCompleted"
     AND ActivityType = "EmailCustomerActivity"
 addToDecisions CompleteWorkflowExecution

ENDIF
```

If the credit card was successfully charged, the decider should respond with `RespondDecisionTaskCompleted` to schedule the `ShipOrder` activity.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "taskToken": "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "decisions":[
      {
```

```
          "decisionType":"ScheduleActivityTask",
          "scheduleActivityTaskDecisionAttributes":{
              "control":"OPTIONAL_DATA_FOR_DECIDER",
              "activityType":{
                  "name":"ShipOrder",
                  "version":"2.4"
              },
              "activityId":"3e2e6e55-e7c4-fee-deed-aa815722b7be",
              "scheduleToCloseTimeout":"3600",
              "taskList":{
                  "name":"SHIPPING"
              },
              "scheduleToStartTimeout":"600",
              "startToCloseTimeout":"3600",
              "heartbeatTimeout":"300",
              "input": "123 Main Street, Anytown, United States"
          }
      }
  ]
}
```

If the credit card was not successfully charged, the decider should respond with `RespondDecisionTaskCompleted` to schedule the `CancelOrder` activity.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "taskToken": "12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "decisions":[
      {
          "decisionType":"ScheduleActivityTask",
          "scheduleActivityTaskDecisionAttributes":{
              "control":"OPTIONAL_DATA_FOR_DECIDER",
              "activityType":{
                  "name":"CancelOrder",
                  "version":"2.4"
              },
              "activityId":"3e2e6e55-e7c4-fee-deed-aa815722b7be",
              "scheduleToCloseTimeout":"3600",
              "taskList":{
                  "name":"CANCELLATIONS"
              },
              "scheduleToStartTimeout":"600",
              "startToCloseTimeout":"3600",
              "heartbeatTimeout":"300",
              "input": "Out of Stock"
          }
      }
  ]
}
```

If Amazon SWF is able to validate the data in the `RespondDecisionTaskCompleted` action, Amazon SWF returns a successful HTTP response similar to the following.

```
HTTP/1.1 200 OK
Content-Length: 11
```

```
Content-Type: application/json
x-amzn-RequestId: 93cec6f7-0747-11e1-b533-79b402604df1
```

Timers

A timer enables you to notify your decider when a certain amount of time has elapsed. When responding to a decision task, the decider has the option to respond with a `StartTimer` decision. This decision specifies an amount of time after which the timer should fire. After the specified time has elapsed, Amazon SWF will add a `TimerFired` event to the workflow execution history and schedule a decision task. The decider can then use this information to inform further decisions. One common application for a timer is to delay the execution of an activity task. For example, a customer might want to take delayed delivery of an item.

Signals

Signals enable you to inform a workflow execution of external events and inject information into a workflow execution while it is running. Any program can send a signal to a running workflow execution by calling the `SignalWorkflowExecution` API. When a signal is received, Amazon SWF records it in the workflow execution's history as `WorkflowExecutionSignaled` event and alerts the decider by scheduling a decision task.

> **Note**
> An attempt to send a signal to a workflow execution that is not open results in `SignalWorkflowExecution` failing with `UnknownResourceFault`.

In this example, the workflow execution is sent a signal to cancel an order.

```
https://swf.us-east-1.amazonaws.com
SignalWorkflowExecution
{"domain": "867530901",
 "workflowId": "20110927-T-1",
 "runId": "f5ebbac6-941c-4342-ad69-dfd2f8be6689",
 "signalName": "CancelOrder",
 "input": "order 3553"}
```

If the workflow execution receives the signal, Amazon SWF returns a successful HTTP response similar to the following. Amazon SWF will generate a decision task to inform the decider to process the signal.

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json
x-amzn-RequestId: bf78ae15-3f0c-11e1-9914-a356b6ea8bdf
```

Sometimes you might want to wait for a signal. For example, a user could cancel an order by sending a signal, but only within one hour of placing the order. Amazon SWF does not have a primitive to enable a decider to wait for a signal from the service. Pause functionality needs to be implemented in the decider itself. In order to pause, the decider should start a timer, using the `StartTimer` decision, which specifies the duration for which the decider will wait for the signal while continuing to poll for decision tasks. When the decider receives a decision task, it should check the history to see if either the signal has been received or the timer has fired. If the signal has been received, then the decider should cancel the timer. However, if instead, the timer has fired, then it means that the signal did not arrive within the specified time. To summarize, in order to wait for a specific signal, do the following.

1. Create a timer for the amount of time the decider should wait.
2. When a decision task is received, check the history to see if the signal has arrived or if the timer has fired.
3. If a signal has arrived, cancel the timer using a `CancelTimer` decision and process the signal. Depending on the timing, the history may contain both `TimerFired` and `WorkflowExecutionSignaled` events. In such cases, you can rely on the relative order of the events in the history to determine which occurred first.
4. If the timer has fired, before a signal is received, then the decider has timed out waiting for the signal. You can fail the execution or do whatever other logic is appropriate to your use case.

Activity Task Cancellation

Activity task cancellation enables the decider to end activities that no longer need to be performed. Amazon SWF uses a cooperative cancellation mechanism and does not forcibly interrupt running activity tasks. You must program your activity workers to handle cancellation requests.

The decider can decide to cancel an activity task while it is processing a decision task. To cancel an activity task, the decider uses the `RespondDecisionTaskCompleted` action with the `RequestCancelActivityTask` decision.

If the activity task has not yet been acquired by an activity worker, the service will cancel the task. Note that there is a potential race condition in that an activity worker could acquire the task at any time. If the task has already been assigned to an activity worker, then the activity worker will be requested to cancel the task.

In this example, the workflow execution is sent a signal to cancel the order.

```
https://swf.us-east-1.amazonaws.com
SignalWorkflowExecution
{"domain": "867530901",
 "workflowId": "20110927-T-1",
 "runId": "9ba33198-4b18-4792-9c15-7181fb3a8852",
 "signalName": "CancelOrder",
 "input": "order 3553"}
```

If the workflow execution receives the signal, Amazon SWF returns a successful HTTP response similar to the following. Amazon SWF will generate a decision task to inform the decider to process the signal.

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json
x-amzn-RequestId: 6c0373ce-074c-11e1-9083-8318c48dee96
```

When the decider processes the decision task and sees the signal in the history, the decider attempts to cancel the outstanding activity that has the `ShipOrderActivity0001` activity ID. The activity ID is provided in the workflow history from the schedule activity task event.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "taskToken":"12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "decisions":[{
          "decisionType":"RequestCancelActivityTask",
```

```
          "RequestCancelActivityTaskDecisionAttributes":{
              "ActivityID":"ShipOrderActivity0001"
          }
      }
  ]
}
```

If Amazon SWF successfully receives the cancellation request, it returns a successful HTTP response similar to the following:

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json
x-amzn-RequestId: 6c0373ce-074c-11e1-9083-8318c48dee96
```

The cancellation attempt is recorded in the history as the `ActivityTaskCancelRequested` event.

If the task is successfully canceled—as indicated by an `ActivityTaskCanceled` event—program your decider to take the appropriate steps that should follow task cancellation such as closing the workflow execution.

If the activity task could not be canceled—for example, if the task completes, fails, or times out instead of canceling—your decider should accept the results of the activity or perform any cleanup or mitigation necessitated by your use case.

If the activity task has already been acquired by an activity worker, then the request to cancel is transmitted through the task-heartbeat mechanism. Activity workers can periodically use `RecordActivityTaskHeartbeat` to report to Amazon SWF that the task is still in progress.

Note that activity workers are not required to heartbeat, although it is recommended for long-running tasks. Task cancellation requires periodic heartbeat to be recorded; if the worker does not heartbeat, the task cannot be canceled.

If the decider requests a cancellation of the task, Amazon SWF sets the value of the `cancelRequest` object to true. The `cancelRequest` object is part of the `ActivityTaskStatus` object which is returned by the service in response to `RecordActivityTaskHeartbeat`.

Amazon SWF does not prevent the successful completion of an activity task whose cancellation has been requested; it is up to the activity to determine how to handle the cancellation request. Depending on your requirements, program the activity worker to either cancel the activity task or ignore the cancellation request.

If you want the activity worker to indicate that the work for the activity task was canceled, program it to respond with a `RespondActivityTaskCanceled`. If you want the activity worker to complete the task, program it to respond with a standard `RespondActivityTaskCompleted`.

When Amazon SWF receives the `RespondActivityTaskCompleted` or `RespondActivityTaskCanceled` request, it updates the workflow execution history and schedules a decision task to inform the decider.

Program the decider to process the decision task and return any additional decisions. If the activity task is successfully canceled, program the decider to perform the tasks needed to continue or close the workflow execution. If the activity task is not successfully canceled, program the decider to accept the results, ignore the results, or schedule any required cleanup.

Markers

You can use markers to record events in the workflow execution history for application specific purposes. Markers are useful when you want to record custom information to help implement decider logic. For example, you could use a marker to count the number of loops in a recursive workflow.

In the following example, the decider completes a decision task and responds with a `RespondDecisionTaskCompleted` action that contains a `RecordMarker` decision.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "taskToken":"12342e17-80f6-FAKE-TASK-TOKEN32f0223",
  "decisions":[{
          "decisionType":"RecordMarker",
          "recordMarkerDecisionAttributes":{
              "markerName":"customer elected special shipping offer"
          }
      },
  ]
}
```

If Amazon SWF successfully records the marker, it returns a successful HTTP response similar to the following.

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json
x-amzn-RequestId: 6c0373ce-074c-11e1-9083-8318c48dee96
```

Recording a marker does not, by itself, initiate a decision task. To prevent the workflow execution from becoming stuck, something must occur that continues the execution of the workflow. For example, this might include the decider scheduling another activity task, the workflow execution receiving a signal, or a previously scheduled activity task completing.

Tagging

As described in the section Tags (p. 68), you can associate up to five tags with a workflow execution when you start the execution using the `StartWorkflowExecution` action, `StartChildWorkflowExecution` decision, or `ContinueAsNewWorkflowExecution` decision. Tagging enables you to filter your results when you use visibility actions to list or count workflow executions.

To use tagging

1. Devise a tagging strategy. Think about your business requirements and create a list of tags that are meaningful to you. Determine which executions will get which tags. Even though an execution can be assigned a maximum of five tags, your tag library can have any number of tags. Because each tag can be any string value up to 256 characters in length, a tag can describe almost any business concept.
2. Tag an execution with up to five tags when you create it.
3. List or count the executions that are tagged with a particular tag by specifying the *tagFilter* parameter with the `ListOpenWorkflowExecutions`, `ListClosedWorkflowExecutions`,

`CountOpenWorkflowExecutions`, and `CountClosedWorkflowExecutions` actions. The action will filter the executions based on the tags specified.

When you associate a tag with a workflow execution, it is permanently associated with that execution, and cannot be removed.

You can specify only one tag in the *`tagFilter`* parameter with `ListWorkflowExecutions`. Also, tag matching is case sensitive, and only exact matches return results.

Assume you have already set up two executions that are tagged as follows.

Execution Name	Assigned Tags
Execution-One	Consumer, 2011-February
Execution-Two	Wholesale, 2011-March

You can filter the list of executions returned by `ListOpenWorkflowExecutions` on the Consumer tag. The *`oldestDate`* and *`latestDate`* values are specified as Unix Time values.

```
https://swf.us-east-1.amazonaws.com
RespondDecisionTaskCompleted
{
  "domain":"867530901",
  "startTimeFilter":{
      "oldestDate":1262332800,
      "latestDate":1325348400
  },
  "tagFilter":{
    "tag":"Consumer"
    }
}
```

Making HTTP Requests to Amazon SWF

Topics

If you don't use one of the AWS SDKs, you can perform Amazon Simple Workflow Service (Amazon SWF) operations over HTTP using the POST request method. The POST method requires that you specify the operation in the header of the request and provide the data for the operation in JSON format in the body of the request.

HTTP Header Contents

Amazon SWF requires the following information in the header of an HTTP request:

- *`host`* The Amazon SWF endpoint.

- *x-amz-date* You must provide the time stamp in either the HTTP *Date* header or the AWS *x-amz-date header* (some HTTP client libraries don't let you set the *Date* header). When an *x-amz-date* header is present, the system ignores any *Date* header when authenticating the request.

  The date must be specified in one of the following three formats, as specified in the HTTP/1.1 RFC:
  - Sun, 06 Nov 1994 08:49:37 GMT (RFC 822, updated by RFC 1123)
  - Sunday, 06-Nov-94 08:49:37 GMT (RFC 850, obsoleted by RFC 1036)
  - Sun Nov 6 08:49:37 1994 (ANSI C's asctime() format)
- *x-amzn-authorization* The signed request parameters in the format:

```
AWS3 AWSAccessKeyId=####,Algorithm=HmacSHA256, [,SignedHeaders=Header1;Head
er2;...]
Signature=S(StringToSign)
```

  AWS3 - This is an AWS implementation-specific tag that denotes the authentication version used to sign the request (currently, for Amazon SWF this value is always *AWS3*).

  AWSAccessKeyId - Your AWS Access Key ID.

  Algorithm - The algorithm used to create the HMAC-SHA value of the string-to-sign, such as *HmacSHA256* or *HmacSHA1*.

  Signature - Base64(Algorithm(StringToSign, SigningKey)). For details see Calculating the HMAC-SHA Signature (p. 115)

  SignedHeaders - Optional. If present, must contain a list of all the HTTP Headers used in the Canonicalized HttpHeaders calculation. A single semicolon character (;) (ASCII character 59) must be used as the delimiter for list values.
- *x-amz-target* The destination service of the request and the operation for the data, in the format *com.amazonaws.swf.service.model.SimpleWorkflowService.* + *<action>*
  For example, *com.amazonaws.swf.service.model.SimpleWorkflowService.RegisterDomain*
- *content-type* The type needs to specify JSON and the character set, as *application/json; charset=UTF-8*

The following is an example header for an HTTP request to create a domain.

```
POST http://swf.us-east-1.amazonaws.com/ HTTP/1.1
Host: swf.us-east-1.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.25)
Gecko/20111212 Firefox/3.6.25 ( .NET CLR 3.5.30729; .NET4.0E)
Accept: application/json, text/javascript, */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
X-Amz-Date: Fri, 13 Jan 2012 18:42:12 GMT
X-Amz-Target: com.amazonaws.swf.service.model.SimpleWorkflowService.RegisterDo
main
Content-Encoding: amz-1.0
X-Amzn-Authorization: AWS3 AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE,Algorithm=Hmac
```

```
SHA256,SignedHeaders=Host;X-Amz-Date;X-Amz-Target;Content-Encoding,Signa
ture=tzjkF55lxAxPhzp/BRGFYQRQRq6CqrM254dTDE/EncI=
Referer: http://swf.us-east-1.amazonaws.com/explorer/index.html
Content-Length: 91
Pragma: no-cache
Cache-Control: no-cache

{"name": "867530902",
 "description": "music",
 "workflowExecutionRetentionPeriodInDays": "60"}
```

Here is an example of the corresponding HTTP response.

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json
x-amzn-RequestId: 4ec4ac3f-3e16-11e1-9b11-7182192d0b57
```

HTTP Body Content

The body of an HTTP request contains the data for the operation specified in the header of the HTTP request. Use the JSON data format to convey data values and data structure, simultaneously. Elements can be nested within other elements using bracket notation. For example, the following shows a request to list all workflow executions that started between two specified points in time—using Unix Time notation.

```
{
 "domain": "867530901",
 "startTimeFilter":
 {
   "oldestDate": 1325376070,
  "latestDate": 1356998399
 },
 "tagFilter":
 {
   "tag": "music purchase"
 }
}
```

Sample Amazon SWF JSON Request and Response

The following example shows a request to Amazon SWF for a description of the domain that we created previously. Then it shows the Amazon SWF response.

HTTP POST Request:

```
POST http://swf.us-east-1.amazonaws.com/ HTTP/1.1
Host: swf.us-east-1.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.25)
Gecko/20111212 Firefox/3.6.25 ( .NET CLR 3.5.30729; .NET4.0E)
Accept: application/json, text/javascript, */*
Accept-Language: en-us,en;q=0.5
```

```
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
X-Amz-Date: Sun, 15 Jan 2012 03:13:33 GMT
X-Amz-Target: com.amazonaws.swf.service.model.SimpleWorkflowService.DescribeDo
main
Content-Encoding: amz-1.0
X-Amzn-Authorization: AWS3 AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE,Algorithm=Hmac
SHA256,SignedHeaders=Host;X-Amz-Date;X-Amz-Target;Content-Encoding,Signa
ture=IFJtq3M366CHqMlTpyqYqd9z0ChCoKDC5SCJBsLifu4=
Referer: http://swf.us-east-1.amazonaws.com/explorer/index.html
Content-Length: 21
Pragma: no-cache
Cache-Control: no-cache

{"name": "867530901"}
```

Amazon SWF Response:

```
HTTP/1.1 200 OK
Content-Length: 137
Content-Type: application/json
x-amzn-RequestId: e86a6779-3f26-11e1-9a27-0760db01a4a8

{"configuration":
  {"workflowExecutionRetentionPeriodInDays": "60"},
 "domainInfo":
  {"description": "music",
   "name": "867530901",
   "status": "REGISTERED"}
}
```

Notice the protocol (*`HTTP/1.1`*) is followed by a status code (*`200`*). A code value of *`200`* indicates a successful operation.

Amazon SWF does not serialize null values. If your JSON parser is set to serialize null values for requests, Amazon SWF ignores them.

Calculating the HMAC-SHA Signature for Amazon SWF

Required Authentication Information

Every request to Amazon Simple Workflow Service (Amazon SWF) must be authenticated. The AWS SDKs automatically sign your requests and manage your token-based authentication as required for Amazon SWF. However, if you want to write your own HTTP POST requests, you need to create an *`x-amzn-authorization`* value for the HTTP POST Header content as part of authenticating your request. For more information about formatting headers, see HTTP Header Contents (p. 112).

Signature Process

Following is the series of tasks required to create an HMAC-SHA (Hash-based Message Authentication Code-Secure Hash Algorithm) request signature. It is assumed you have already have received your AWS access keys (Access Key ID and Secret Key).

> **Note**
> You can use either a SHA1 or SHA256 method for signing. Use the same one throughout the signing process, and it must match the value for the *`Algorithm`* name provided in the HTTP header.

You perform the following tasks to sign and submit a request to Amazon SWF.

Signing Process

1. Create a canonical form of the HTTP request headers. The canonical form of the HTTP header includes the following:
   - *`host`*
   - Any header element starting with *`x-amz-`*

   For more information about the included headers, see HTTP Header Contents (p. 112).
   a. For each header name-value pair, convert the header name to lowercase (not the header value).
   b. Build a map of header name to comma separated header values as prescribed by RFC 2616, section 4.2.

   ```
   x-amz-example: value1
   x-amz-example: value2  =>  x-amz-example:value1,value2
   ```

   c. For each header name-value pair, convert the name-value pair into a string in the format `headerName:headerValue`. Trim any whitespace from the beginning and end of both `headerName` and `headerValue`, with no space on each side of the colon.

   ```
   x-amz-example1:value1,value2
   x-amz-example2:value3
   ```

   d. Insert a new line (U+000A) after each converted string, including the last string.
   e. Sort the collection of converted strings by header name, alphabetically.
2. Create a *`string-to-sign`* value that includes the following:
   - Line 1: The HTTP method (*`POST`*), followed by a newline.
   - Line 2: The request URI (*`/`*), followed by a newline.
   - Line 3: An empty string. Typically, a query string goes here, but Amazon SWF doesn't use a query string. Follow with a newline
   - Line 4–*n*: The string representing that canonicalized request headers you computed in step 1, followed by a newline. This newline will create a blank line between the headers and the body of the HTTP request per RFC 2616.
   - The request body. Do not follow the request body with a newline.
3. Compute the SHA256 or SHA1 digest of the *`string-to-sign`* value. Use the same SHA method throughout the process.
4. Compute and Base64 encode the HMAC-SHA using either a SHA256 or a SHA1 digest (depending on which one you've chosen to use) of the resulting value from the previous step using the temporary secret access key you received from the AWS Security Token Service using the GetSessionToken API. For more information about using temporary security credentials with Amazon SWF and other Amazon Web Services, go to the Identity and Access Management documentation.

Note
Amazon SWF expects an equal sign (=) at the end of the Base64-encoded HMAC-SHA value. If your Base64 encoding routine doesn't include appending an equal sign, append one to the end.

5. Put the resulting value as the value for the *`Signature`* name in the *`x-amzn-authorization`* header of the HTTP request to Amazon SWF.
6. Amazon SWF verifies the request and performs the specified operation.

For the AWS SDK for Java implementation of AWS version 3 signing, see the AWSSigner.java class.

Amazon Simple Workflow Service Resources

This chapter provides additional resources and reference information that is useful when developing workflows with Amazon SWF.

Topics

Amazon SWF Timeout Types

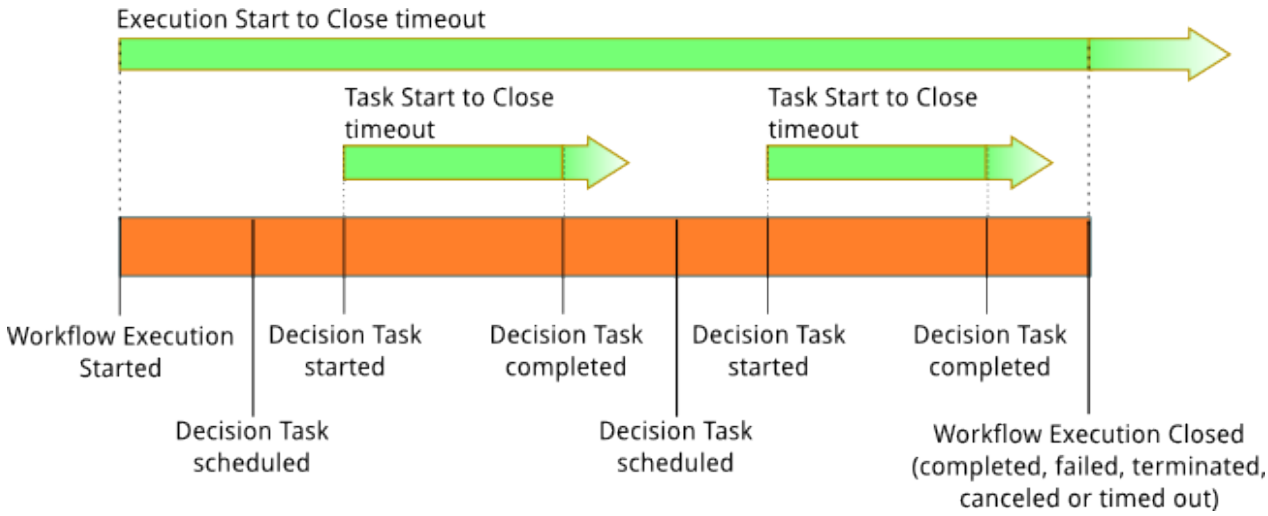
To ensure that workflow executions run correctly, Amazon SWF enables you to set different types of timeouts. Some timeouts specify how long the workflow can run in its entirety. Other timeouts specify how long activity tasks can take before being assigned to a worker and how long they can take to complete from the time they are scheduled. All timeouts in the Amazon SWF API are specified in seconds. Amazon SWF also supports the string "NONE" as a timeout value, which indicates no timeout.

For timeouts related to decision tasks and activity tasks, Amazon SWF adds an event to the workflow execution history. The attributes of the event provide information about what type of timeout occurred and which decision task or activity task was affected. Amazon SWF also schedules a decision task. When the decider receives the new decision task, it will see the timeout event in the history and take an appropriate action by calling the RespondDecisionTaskCompleted action.

A task is considered open from the time that it is scheduled until it is closed. Therefore a task is reported as open while a worker is processing it. A task is closed when a worker reports it as completed, canceled, or failed. A task may also be closed by Amazon SWF as the result of a timeout.

Timeouts in Workflow and Decision Tasks

The following diagram shows how workflow and decision timeouts are related to the lifetime of a workflow:

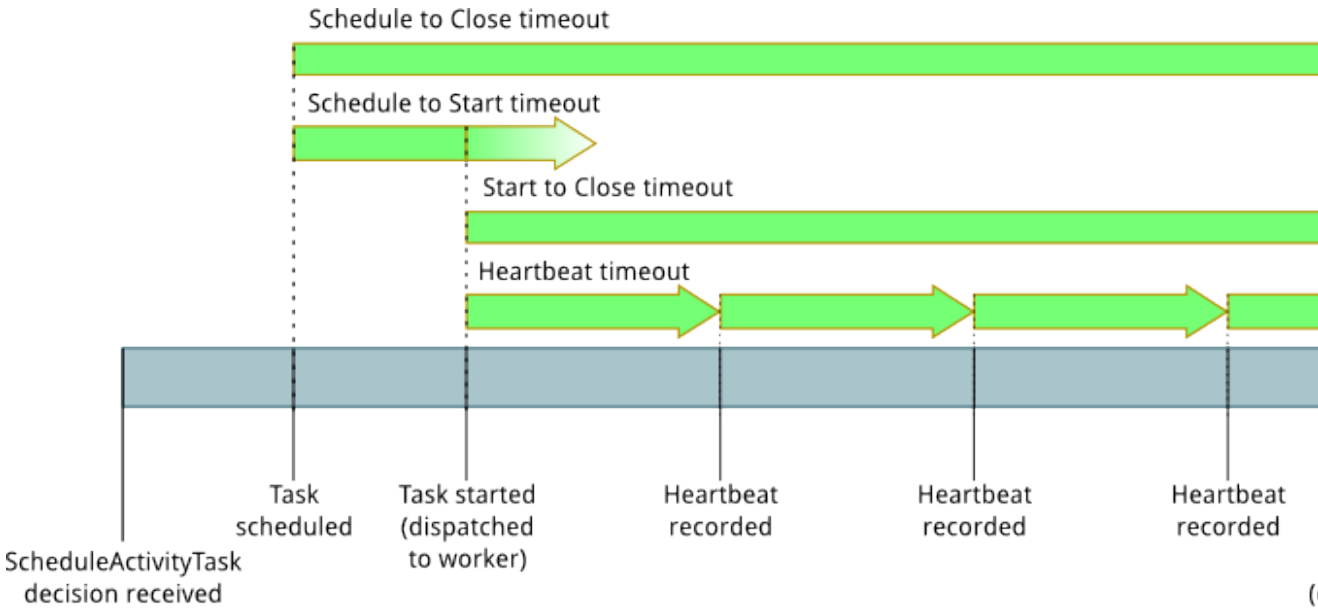


There are two timeout types that are relevant to workflow and decision tasks:

- **Workflow Start to Close (timeoutType: START_TO_CLOSE):** This timeout specifies the maximum time that a workflow execution can take to complete. It is set as a default during workflow registration, but it can be overridden with a different value when the workflow is started. If this timeout is exceeded, Amazon SWF closes the workflow execution and adds an event of type WorkflowExecutionTimedOut to the workflow execution history. In addition to the `timeoutType`, the event attributes specify the `childPolicy` that is in effect for this workflow execution. The child policy specifies how child workflow executions are handled if the parent workflow execution times out or otherwise terminates. For example, if the `childPolicy` is set to TERMINATE, then child workflow executions will be terminated. Once a workflow execution has timed out, you cannot take any action on it other than visibility calls.
- **Decision Task Start to Close (timeoutType: START_TO_CLOSE):** This timeout specifies the maximum time that the corresponding decider can take to complete a decision task. It is set during workflow type registration. If this timeout is exceeded, the task is marked as timed out in the workflow execution history, and Amazon SWF adds an event of type DecisionTaskTimedOut to the workflow history. The event attributes will include the IDs for the events that correspond to when this decision task was scheduled (`scheduledEventId`) and when it was started (`startedEventId`). In addition to adding the event, Amazon SWF also schedules a new decision task to alert the decider that this decision task timed out. After this timeout occurs, an attempt to complete the timed-out decision task using `RespondDecisionTaskCompleted` will fail.

Timeouts in Activity Tasks

The following diagram shows how timeouts are related to the lifetime of an activity task:



There are four timeout types that are relevant to activity tasks:

- **Activity Task Start to Close (timeoutType: START_TO_CLOSE):** This timeout specifies the maximum time that an activity worker can take to process a task after the worker has received the task. Attempts to close a timed out activity task using RespondActivityTaskCanceled, RespondActivityTaskCompleted, and RespondActivityTaskFailed will fail.
- **Activity Task Heartbeat (timeoutType: HEARTBEAT):** This timeout specifies the maximum time that a task can run before providing its progress through the `RecordActivityTaskHeartbeat` action.
- **Activity Task Schedule to Start (timeoutType: SCHEDULE_TO_START):** This timeout specifies how long Amazon SWF waits before timing out the activity task if no workers are available to perform the task. Once timed out, the expired task will not be assigned to another worker.
- **Activity Task Schedule to Close (timeoutType: SCHEDULE_TO_CLOSE):** This timeout specifies how long the task can take from the time it is scheduled to the time it is complete. As a best practice, this value should not be greater than the sum of the task schedule-to-start timeout and the task start-to-close timeout.

> **Note**
> Each of the timeout types has a default value, which is generally set to NONE (infinite). The maximum time for any activity execution is limited to one year, however.

You set default values for these during activity type registration, but you can override them with new values when you schedule the activity task. When one of these timeouts occurs, Amazon SWF will add an event of type ActivityTaskTimedOut to the workflow history. The `timeoutType` value attribute of this event will specify which of these timeouts occurred. For each of the timeouts, the value of `timeoutType` is shown in parentheses. The event attributes will also include the IDs for the events that correspond to when the activity task was scheduled (`scheduledEventId`) and when it was started (`startedEventId`). In addition to adding the event, Amazon SWF also schedules a new decision task to alert the decider that the timeout occurred.

Amazon SWF Metrics for CloudWatch

Amazon SWF now provides metrics for CloudWatch that you can use to track your workflows and activities and set alarms on threshold values that you choose. You can view metrics using the AWS Management Console. For more information, see Viewing Amazon SWF Metrics (p. 99).

Topics

Metrics that Report a Time Interval

Some of the Amazon SWF metrics for CloudWatch are *time intervals*, always measured in milliseconds. These metrics generally correspond to stages of your workflow execution for which you can set workflow and activity timeouts, and have similar names.

For example, the **DecisionTaskStartToCloseTime** metric measures the time it took for the decision task to complete after it began executing, which is the same time period for which you can set a **DecisionTaskStartToCloseTimeout** value.

For a diagram of each of these workflow stages and to learn when they occur over the workflow and activity lifecycles, see Timeout Types (p. 118).

Metrics that Report a Count

Some of the Amazon SWF metrics for CloudWatch report results as a *count*. For example, **WorkflowsCanceled**, records a result as either *one* or *zero*, indicating whether or not the workflow was canceled. A value of zero does not indicate that the metric was not reported, only that the condition described by the metric did not occur.

For count metrics, minimum and maxiumum will always be either zero or one, but average will be a value ranging from zero to one.

Workflow Metrics

Dimensions for Amazon SWF workflow metrics include the **Domain**, **WorkflowTypeName**, **WorkflowTypeVersion** and **Metric Name**.

The following metrics are available for Amazon SWF workflows:

- **DecisionTaskScheduleToStartTime** – the time interval, in milliseconds, between the time that the decision task was scheduled and the time it was picked up by a worker and started.
- **DecisionTaskStartToCloseTime** – the time interval, in milliseconds, between the time that the decision task was started and the time it was closed.
- **DecisionTasksCompleted** – the count of decision tasks that have been completed.
- **StartedDecisionTasksTimedOutOnClose** – the count of decision tasks that started but timed out on closing.
- **WorkflowStartToCloseTime** – the time, in milliseconds, between the time the workflow started and the time it closed.
- **WorkflowsCanceled** – the count of workflows that were canceled.

- **WorkflowsCompleted** – the count of workflows that completed.
- **WorkflowsContinuedAsNew** – the count of workflows that continued as new.
- **WorkflowsFailed** – the count of workflows that failed.
- **WorkflowsTerminated** – the count of workflows that were terminated.
- **WorkflowsTimedOut** – the count of workflows that timed out, for any reason.

Activity Metrics

Dimensions for Amazon SWF activity metrics include the **Domain**, **ActivityTypeName**, **ActivityTypeVersion** and **Metric Name**.

The following metrics are available for Amazon SWF activities:

- **ActivityTaskScheduleToCloseTime** – the time interval, in milliseconds, between the time when the activity was scheduled to when it closed.
- **ActivityTaskScheduleToStartTime** – the time interval, in milliseconds, between the time when the activity task was scheduled and when it started.
- **ActivityTaskStartToCloseTime** – the time interval, in milliseconds, between the time when the activity task started and when it was closed.
- **ActivityTasksCanceled** – the count of activity tasks that were canceled.
- **ActivityTasksCompleted** – the count of activity tasks that completed.
- **ActivityTasksFailed** – the count of activity tasks that failed.
- **ScheduledActivityTasksTimedOutOnClose** – the count of activity tasks that were scheduled but timed out on close.
- **ScheduledActivityTasksTimedOutOnStart** – the count of activity tasks that were scheduled but timed out on start.
- **StartedActivityTasksTimedOutOnClose** – the count of activity tasks that were started but timed out on close.
- **StartedActivityTasksTimedOutOnHeartbeat** – the count of activity tasks that were started but timed out due to a heartbeat timeout.

Amazon Simple Workflow Service Limits

Amazon SWF places limits on the sizes of certain workflow parameters, such as on the number of domains per account and on the size of the workflow execution history.

This section lists the limits that can affect your application design, and provides information about how to request a limit increase if you need it for your application.

Topics

General Account Limits for Amazon SWF

- **Maximum registered domains:** 100

This limit includes both registered and deprecated domains.

- **Maximum workflow and activity types:** 10,000 each per domain

  This limit includes both registered and deprecated types.
- **API call limit:** Beyond infrequent spikes, applications may be throttled if they make a large number of API calls in a very short period of time.
- **Maximum request size:** 1 MB per request

  This is the *total* data size per Amazon SWF API request, including the request header and all other associated request data.

Limits on Workflow Executions

- **Maximum open workflow executions:** 100,000 per domain

  This count includes child workflow executions.
- **Maximum workflow execution time:** 1 year
- **Maximum workflow execution history size:** 25,000 events
- **Maximum child workflow executions:** 1,000 per workflow execution.
- **Workflow execution idle time limit:** 1 year (constrained by workflow execution time limit)

  You can configure workflow timeouts (p. 118) to cause a timeout event to occur if a particular stage of your workflow takes too long.
- **Workflow retention time limit:** 90 days

  After this time, the workflow history can no longer be retrieved or viewed. There is no further limit to the number of closed workflow executions that are retained by Amazon SWF.

If your use case requires you to go beyond these limits, you can use features Amazon SWF provides to continue executions and structure your applications using child workflow (p. 66) executions. If you find that you still need a limits increase, see How to Request an Amazon SWF Service Limits Increase (p. 124).

Limits on Task Executions

- **Maximum task execution time:** 1 year (constrained by workflow execution time limit)

  You can configure activity timeouts (p. 118) to cause a timeout event to occur if a particular stage of your activity task (p. 44) execution takes too long.
- **Maximum time SWF will keep a task in the queue:** 1 year (constrained by workflow execution time limit)

  You can configure default activity timeouts (p. 118) during activity registration that will cause a timeout event to occur if a particular stage of your activity task (p. 44) execution takes too long. You can also override the default activity timeouts when you schedule an activity task in your decider code.
- **Maximum open activity tasks:** 1,000 per workflow execution.

  This limit includes both activity tasks that have been scheduled and those being processed by workers.
- **Maximum open timers:** 1,000 per workflow execution
- **Maximum input/result data size:** 32,000 characters

  This limit affects activity or workflow execution result data, input data when scheduling activity tasks or workflow executions, and input sent with a workflow execution signal (p. 66).
- **Maximum decisions in a decision task response:** varies

Due to the 1 MB limit on the maximum API request size (p. 122), the number of decisions returned in a single call to RespondDecisionTaskCompleted will be limited according to the size of the data used by each decision, including the size of any input data provided to scheduled activity tasks or to workflow executions.

How to Request an Amazon SWF Service Limits Increase

If you think that you will exceed any of the above limits, please use the Amazon SWF Limit Increase Form to contact the Amazon SWF team to discuss your scenario and request higher limits.

Amazon Simple Workflow Service Endpoints

A list of the current Amazon SWF Regions and Endpoints are provided in the *Amazon Web Services General Reference*, along with the endpoints for other services.

Amazon SWF domains and all related workflows and activities must exist within the same region to communicate with each other. Furthermore, any registered domains, workflows and activities within a region don't exist in other regions. For example, if you create a domain named "MySampleDomain" in both *us-east-1* and in *us-west-2*, they exist as *separate domains*: none of the workflows, task lists, activities, or data associated with your domains are shared across regions.

If you use other AWS resources in your workflows, such as Amazon EC2 instances, these must also exist in the same region as your Amazon SWF resources. The only exceptions to this are services that span regions, such as Amazon S3 and IAM. You can access these services from workflows that exist in any region that supports them.

Additional Documentation for the Amazon Simple Workflow Service

In addition to this Developer Guide, you may find the following documentation useful.

Topics

Amazon Simple Workflow Service API Reference

The Amazon Simple Workflow Service API Reference provides detailed information about the Amazon SWF HTTP API, including actions, request and response structures and error codes.

AWS Flow Framework Documentation

The AWS Flow Framework is a programming framework that simplifies the process of implementing distributed asynchronous applications that use Amazon SWF to manage their workflows and activities, so you can focus on implementing your workflow logic.

Each AWS Flow Framework is designed to work idiomatically in the language for which it is designed, so you can work naturally with your language of choice to implement workflows with all of the benefits of Amazon SWF.

There are AWS Flow Frameworks for the following languages:

Java

The AWS Flow Framework for Java Developer Guide provides information about how to obtain, set up and use the AWS Flow Framework for Java.

For a list of the classes, methods, and annotations used by the framework, refer to the AWS Flow Framework for Java API Reference.

Ruby

The AWS Flow Framework for Ruby Developer Guide provides information about how to obtain, set up and use the AWS Flow Framework for Ruby.

For a list of the classes and methods used by the framework, refer to the AWS Flow Framework for Ruby API Reference.

The aws-flow-ruby-samples project on GitHub provides code examples that demonstrate many of the features of the AWS Flow Framework for Ruby. You can use this code to learn more about the framework and as an aid for designing and implementing your own workflows.

AWS SDK Documentation

The AWS Software Development Kits (SDKs) provide access to Amazon SWF in many different programming languages. The SDKs follow the HTTP API closely, but also provide language-specific programming interfaces for some Amazon SWF features. You can find out more information about each SDK by visiting the following links.

Note
Only SDKs that have support for Amazon SWF at the time of writing are listed here. For a full list of the available AWS SDKs, visit the Tools for Amazon Web Services page.

Java

The AWS SDK for Java provides a Java API for AWS infrastructure services.

To view the available documentation, see the AWS SDK for Java Documentation page. You can also go directly to the Amazon SWF sections in the SDK reference by following these links:

- Class: AmazonSimpleWorkflowClient
- Class: AmazonSimpleWorkflowAsyncClient
- Interface: AmazonSimpleWorkflow
- Interface: AmazonSimpleWorkflowAsync

JavaScript

The AWS SDK for JavaScript allows developers to build libraries or applications that make use of AWS services using a simple and easy-to-use API available both in the browser or inside of Node.js applications on the server.

To view the available documentation, see the AWS SDK for JavaScript Documentation page. You can also go directly to the Amazon SWF section in the SDK reference by following this link:

- Class: AWS.SimpleWorkflow

.NET

The AWS SDK for .NET is a single, downloadable package that includes Visual Studio project templates, the AWS .NET library, C# code samples, and documentation. The AWS SDK for .NET makes it easier for Windows developers to build .NET applications for Amazon SWF and other services.

To view the available documentation, see the AWS SDK for .NET Documentation page. You can also go directly to the Amazon SWF sections in the SDK reference by following these links:

- Namespace: Amazon.SimpleWorkflow
- Namespace: Amazon.SimpleWorkflow.Model

PHP

The AWS SDK for PHP provides a PHP programming interface to Amazon SWF.

To view the available documentation, see the AWS SDK for PHP Documentation page. You can also go directly to the Amazon SWF section in the SDK reference by following this link:

- Class: SwfClient

Python

The AWS SDK for Python (Boto) provides a Python programming interface to Amazon SWF.

To view the available documentation, see the boto: A Python interface to Amazon Web Services page. You can also go directly to the Amazon SWF sections in the documentation by following these links:

- Amazon SWF Tutorial
- Amazon SWF Reference

Ruby

The AWS SDK for Ruby provides a Ruby programming interface to Amazon SWF.

To view the available documentation, see the AWS SDK for Ruby Documentation page. You can also go directly to the Amazon SWF section in the SDK reference by following this link:

- Class: AWS::SimpleWorkflow

AWS CLI Documentation

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

For more information about the AWS CLI, see the AWS Command Line Interface page.

For an overview of the available commands for Amazon SWF, see swf in the *AWS Command Line Interface Reference*.

Web Resources for the Amazon Simple Workflow Service

There are a number of Web resources that you can use to learn more about Amazon SWF or to get help with using the service and developing workflows.

Topics

Amazon SWF Forum

The Amazon SWF forum provides a place for you to communicate with other Amazon SWF developers and members of the Amazon SWF development team at Amazon to ask questions and to get answers.

You can visit the forum at: Forum: Amazon Simple Workflow Service. *You must be signed in to your AWS account to view the forum*.

Amazon SWF FAQ

The Amazon SWF FAQ provide answers to frequently-asked questions about Amazon SWF, including an overview of common use cases, differences between Amazon SWF and other services, and more.

You can access the FAQ here: Amazon SWF FAQ.

Amazon SWF Videos

The Amazon Web Services channel on YouTube provides video training for all of Amazon's Web Services, including Amazon SWF.

Videos are updated frequently; for a full list of Amazon SWF-related videos, you can use the following query: *Simple Workflow* in Amazon Web Services

Amazon SWF Source Code and Samples

The source code for the AWS Flow Framework for Ruby is available on GitHub. You can use the following links to access it and its associated samples and recipes.

- AWS Flow Framework for Ruby
- AWS Flow Framework for Ruby Samples and Recipes

Amazon Simple Workflow Service Developer Guide History

The following table describes the important changes to the documentation since the last release of the *Amazon Simple Workflow Service Developer Guide*.

- **API version:** 2012-01-25
- **Latest documentation update:** April 28, 2014

Change	Description	Date Changed
Update	Two new topics related to CloudWatch metrics for Amazon SWF have been added: Amazon SWF Metrics for CloudWatch (p. 121), which provides a list and descriptions of the supported metrics, and Viewing Amazon SWF Metrics (p. 99), which provides information about how to view metrics and set alarms with the AWS Management Console.	April 28, 2014
Update	Added a new section: Resources (p. 118). This section provides some service reference information and provides information about additional documentation, samples, code and other web resources for Amazon SWF developers.	March 19, 2014
Update	Added a workflow tutorial. See Tutorial: A Subscription Workflow with Amazon SWF and Amazon SNS (p. 9).	October 25, 2013
Update	Added AWS CLI information and example (p. 103).	August 26, 2013
Update	Updates and fixes.	August 1, 2013
Update	Updated the document to describe how to use IAM for access control.	February 22, 2013
Initial Release	This is the first release of the *Amazon Simple Workflow Service Developer Guide*.	October 16, 2012

AWS Glossary

Numbers and Symbols

100-continue
: A method that enables a client to see if a server can accept a request before actually sending it. For large PUTs, this method can save both time and bandwidth charges.

A

ABT
: Amazon Payments Balance Transfer. A method to approve or deny payments synchronously. It requires payment approval before processing continues.

access control list
: A document that defines who can access a particular bucket or object. Each bucket and object in Amazon S3 has an ACL. The document defines what each type of user can do, such as write and read permissions.

access identifiers
: See credentials.

access key ID
: A string that uniquely identifies your AWS account as well as individual IAM users in your account. It's used in conjunction with the secret access key (p. 156) to cryptographically sign programmatic AWS requests.

access key rotation
: A method to increase security by changing the AWS access key ID. This method enables you to retire an old key at your discretion.

access policy language
: A language for writing documents (that is, *policies*) that specify who can access a particular AWS resource and under what conditions.

account
: The AWS account associated with a particular AWS login ID and password.

  IAM: The AWS account that centrally controls all the resources created under its umbrella and pays for all AWS activity for those resources. The AWS account has permission to do anything and everything with all the AWS account resources. This is in contrast to the user (p. 161).

account activity	A web page showing your month-to-date AWS usage and costs. The account activity page is located at http://aws.amazon.com/account-activity.
ACH	Amazon bank account debit (aka *bank account withdrawal*, *bank account transaction*, and *Automated Clearing House*). An asynchronous bank account debit payment method.
action	An API function. Also called *operation* or *call*. The activity the principal (p. 151) has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." For example, Jane sends a request to Amazon SQS with Action=ReceiveMessage. Amazon CloudWatch: The response initiated by the change in an alarm's state: for example, from OK to ALARM. The state change may be triggered by a metric reaching the alarm threshold, or by a SetAlarmState request. Each alarm can have one or more actions assigned to each state. Actions are performed once each time the alarm changes to a state that has an action assigned, such as an Amazon Simple Notification Service notification, an Auto Scaling policy execution or an Amazon EC2 instance stop/terminate action.
activate URL	The location where your customers can generate a new activation key (p. 130) for your product, should you request it. The activate URL is http://www.amazon.com/dp-activate.
activation	The process your product goes through to prepare itself for the customer's use. During activation, your product obtains the required credentials for the customer.
activation key	An encoded string that represents the relationship between a customer and a Amazon DevPay product the customer has purchased. AWS generates this value when the customer completes the purchase of the product. You use the key to obtain credentials related to the customer and product.
active authorization	When you inform customers of a price change for your product, they have to take action to accept the price change. If they don't take the action to accept the price change, their access to your product is canceled when the price change takes effect.
active trusted signers	A list showing each of the trusted signers you've specified and the IDs of the corresponding active key pairs that CloudFront is aware of. To be able to create working signed URLs, a trusted signer must appear in this list with at least one key pair ID.
administrative suspension	Auto Scaling might suspend processes for Auto Scaling group (p. 133) that repeatedly fail to launch instances. Auto Scaling groups that most commonly experience administrative suspension have zero running instances, have been trying to launch instances for more than 24 hours, and have not succeeded in that time.
alarm	An item that watches a single metric over a specified time period, and triggers an Amazon SNS topic (p. 161) or an Auto Scaling policy (p. 150) if the value of the metric crosses a threshold value over a predetermined number of time periods.
allow	An allow results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests received before 1:00 p.m. on April 30, 2010. An allow overrides any default deny (p. 138), but never an explicit deny (p. 140).
Amazon CloudFront	An AWS content delivery service that helps you improve the performance, reliability, and availability of your websites and applications. See Also http://aws.amazon.com/cloudfront.

Amazon CloudSearch	A fully-managed service in the AWS cloud that makes it easy to set up, manage, and scale a search solution for your website or application.
Amazon CloudWatch	A web service that enables you to monitor and manage various metrics, and configure alarm actions based on data from those metrics. See Also http://aws.amazon.com/cloudwatch.
Amazon DevPay	An easy-to-use online billing and account management service that makes it easy for you to sell an Amazon EC2 AMI or an application built on Amazon S3. See Also http://aws.amazon.com/devpay.
Amazon Elastic Block Store	A service that provides block level storage volumes for use with EC2 instances. See Also http://aws.amazon.com/ebs.
Amazon EBS-backed AMI	Instances launched from this type of AMI use an Amazon EBS volume as their root device. Compare this with instances launched from Amazon S3-backed AMIs, which use the instance store as the root device.
Amazon Elastic Compute Cloud	A web service that enables you to launch and manage Linux/UNIX and Windows server instances in Amazon's data centers. See Also http://aws.amazon.com/ec2.
Amazon EC2 VM Import Connector	See http://aws.amazon.com/ec2/vmimport.
Amazon Elastic MapReduce	A web service that makes it easy to process large amounts of data efficiently. Amazon EMR uses Hadoop processing combined with several AWS products to do such tasks as web indexing, data mining, log file analysis, machine learning, scientific simulation, and data warehousing. See Also http://aws.amazon.com/elasticmapreduce.
Amazon Flexible Payments Service	A web service that enables developers to accept payments on their websites. See Also http://aws.amazon.com/fps.
Amazon Machine Image	An encrypted machine image stored in Amazon Elastic Block Store (p. 131) or Amazon Simple Storage Service. AMIs are like a template of a computer's root drive. They contain the operating system and can also include software and layers of your application, such as database servers, middleware, web servers, and so on.
Mechanical Turk	Provides an on-demand, scalable, human workforce to complete jobs that humans can do better than computers. See Also http://aws.amazon.com/mturk.
Amazon Payments	See http://payments.amazon.com.
Amazon Relational Database Service	A web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks. See Also http://aws.amazon.com/rds.
Amazon Resource Name	A standardized way to refer to an AWS resource. For example: arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Bob.
Amazon Route 53	A web service you can use to create a new DNS service or to migrate your existing DNS service to the cloud. See Also http://aws.amazon.com/route53.
Amazon S3	See Amazon Simple Storage Service.

Amazon S3-Backed AMI	Instances launched from this type of AMI use the instance store as their root device. Compare this with instances launched from Amazon EBS-backed AMIs, which use an Amazon EBS volume as the root device.
Amazon Simple Email Service	An easy-to-use, cost-effective email solution for applications. See Also http://aws.amazon.com/ses.
Amazon Simple Notification Service	A web service that enables applications, end-users, and devices to instantly send and receive notifications from the cloud. See Also http://aws.amazon.com/sns.
Amazon Simple Pay	Incorporates a subset of Amazon FPS functionality. See Also https://payments.amazon.com.
Amazon Simple Queue Service	Reliable and scalable hosted queues for storing messages as they travel between computers. See Also http://aws.amazon.com/sqs.
Amazon Simple Storage Service	Storage for the internet. You can use it to store and retrieve any amount of data at any time, from anywhere on the web. See Also http://aws.amazon.com/s3.
Amazon SimpleDB	A highly-available, scalable, and flexible non-relational data store that enables you to store and query data items using web service requests. See Also http://aws.amazon.com/simpledb.
Amazon Virtual Private Cloud	A web service that enables you to create a virtual network for your AWS resources. See Also http://aws.amazon.com/vpc.
Amazon Web Services	An infrastructure web services platform in the cloud for companies of all sizes. See Also http://aws.amazon.com.
AMI	See Amazon Machine Image.
application	A logical collection of AWS Elastic Beanstalk components, including environments, versions, and environment configurations. An application is conceptually similar to a folder.
analysis scheme	Language-specific Amazon CloudSearch text analysis options that are applied to a text field to control stemming and configure stopwords and synonyms.
Application Billing	The location where your customers manage the Amazon DevPay products they've purchased. This is the URL http://www.amazon.com/dp-applications.
application version	A specific, labeled iteration of an application that represents a functionally consistent set of deployable application code. A version points to an Amazon S3 object (a JAVA WAR file) that contains the application code.
approval	If a Worker's response satisfies your Human Intelligence Task (p. 143), you approve the assignment. When you approve an assignment Mechanical Turk transfers the HIT reward from your Mechanical Turk account to the Worker's Amazon Payments account.
ARN	See Amazon Resource Name.
assignment	When a worker (p. 163) finds a Human Intelligence Task (p. 143) (HIT) to complete, the worker accepts the HIT. Mechanical Turk creates an *assignment* to track the work to completion and store the answer the worker submits. The assignment belongs exclusively to the worker who accepted it and guarantees that the worker

	can submit results and be eligible for a reward—up until the HIT or assignment expires.
asynchronous bounce	A type of bounce (p. 134) that occurs when a receiver (p. 153) initially accepts an email message for delivery and then subsequently fails to deliver it.
attribute	Similar to a column on a spreadsheet, an attribute represents a data category. In Amazon SimpleDB, an attribute has a name (such as *color*), which has a value (such as *blue*) when applied to a data item.
authentication	The process of proving your identity to a system.
Auto Scaling	A web service designed to launch or terminate instance (p. 143)s automatically based on user-defined policies, schedules, and health checks. See Also http://aws.amazon.com/autoscaling.
Auto Scaling group	A representation of multiple Amazon Elastic Compute Cloud (p. 131) instance (p. 143)s that share similar characteristics, and that are treated as a logical grouping for the purposes of instance scaling and management.
Availability Zone	A distinct location within a region (p. 153) that is insulated from failures in other Availability Zones, and provides inexpensive, low-latency network connectivity to other Availability Zones in the same region.
AWS	See Amazon Web Services.
AWS CloudFormation	A service for writing or changing templates that create and delete related AWS resources together as a unit. See Also http://aws.amazon.com/cloudformation.
AWS Consolidated Billing	A billing option that lets you get a single bill for multiple AWS accounts. See Also http://aws.amazon.com/consolidated-billing.
AWS Elastic Beanstalk	See Also http://aws.amazon.com/elasticbeanstalk.
AWS Import/Export	A service for transferring large amounts of data between AWS and portable storage devices. See Also http://aws.amazon.com/importexport.
AWS Identity and Access Management	A web service that enables Amazon Web Services (p. 132) customers to manage users and user permissions within AWS. See Also http://aws.amazon.com/iam.
AWS Management Console	A graphical interface to manage compute, storage, and other cloud resources. See Also http://aws.amazon.com/console.
AWS Multi-Factor Authentication	An optional AWS account security feature. Once you enable AWS MFA, you must provide a six-digit, single-use code in addition to your sign-in credentials whenever you access secure AWS web site pages or the AWS Management Console. You get this single-use code from an authentication device that you keep in your physical possession. See Also http://aws.amazon.com/mfa.
AWS Resources	See resource.
AWS VPN CloudHub	Enables secure communication between branch offices using a simple hub-and-spoke model, with or without a VPC.

B

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

basic monitoring	Monitoring of AWS-provided metrics derived at a 5-minute frequency.
batch	See document batch.
BGP ASN	Border Gateway Protocol Autonomous System Number. A unique identifier for a network, for use in BGP routing. Amazon EC2 supports all 2-byte ASN numbers in the range of 1 - 65334, with the exception of 7224, which is reserved.
blacklist	A list of IP addresses, email addresses, or domains that an Internet Service Provider (p. 144) suspects to be the source of spam (p. 158). The ISP blocks incoming emails from these addresses or domains.
block	A data set. Amazon EMR breaks large amounts of data into subsets. Each subset is called a data block. Amazon EMR assigns an ID to each block and uses a hash table to keep track of block processing.
block device	A storage device that supports reading and (optionally) writing data in fixed-size blocks, sectors, or clusters.
block device mapping	A mapping structure for every AMI and instance that specifies the block devices attached to the instance.
bootstrap action	A user-specified default or custom action that runs a script or an application on all nodes of a job flow before Hadoop starts.
Border Gateway Protocol Autonomous System Number	See BGP ASN.
bounce	A failed email delivery attempt.
breach	The condition in which a user-set threshold (upper or lower boundary) is passed. If the duration of the breach is significant, as set by a breach duration parameter, it can possibly start a scaling activity (p. 156).
bucket	A container for objects stored in Amazon S3. Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `johnsmith` bucket, then authorized users can access the object with the URL `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`.
bucket owner	Just as Amazon is the only owner of the domain name Amazon.com, only one person or organization can own a bucket in Amazon S3.
bundling	A commonly used term for creating an Amazon Machine Image (p. 131). It specifically refers to creating Amazon S3-backed AMIs.
buyer	Amazon FPS: The customer making a purchase. Also called a sender (p. 156). The buyer pays the seller (p. 156) for a product or service. Amazon Simple Pay: The customer who sends a payment using an Amazon Simple Pay button.

C

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

cache cluster	A logical cache distributed over multiple cache node (p. 135)s. A cache cluster can be set up with a specific number of cache nodes.
cache cluster identifier	Customer-supplied identifier for the cache cluster that must be unique for that customer in an AWS region.
cache engine version	The version of the Memcached service that is running on the cache node.
cache node	A fixed-size chunk of secure, network-attached RAM. Each cache node runs an instance of the Memcached service, and has its own DNS name and port. Multiple types of cache nodes are supported, each with varying amounts of associated memory.
cache node type	EC2 instance type used to run the cache node.
cache parameter group	A container for cache engine parameter values that can be applied to one or more cache clusters.
cache security group	A group maintained by ElastiCache that combines ingress authorizations to cache nodes for hosts belonging to Amazon EC2 security groups specified through the console or the API or command line tools.
caller	A developer who facilitates payment between a sender (p. 156) and a recipient (p. 153).
caller reference	A unique value that you provide and AWS uses to prevent replays of your request.
canned access policy	A standard access control policy that you can apply to a bucket or object. Options include: private, public-read, public-read-write, and authenticated-read.
canonicalization	The process of converting data into a standard format that a service such as Amazon S3 can recognize.
capacity	Each Auto Scaling group (p. 133) is defined with a minimum and maximum compute size. The amount of available compute size at any time is the current capacity. A scaling activity (p. 156) increases or decreases the capacity—within the defined minimum and maximum values.
Cascading	Cascading is an open-source Java library that provides a query API, a query planner, and a job scheduler for creating and running Hadoop MapReduce applications. Applications developed with Cascading are compiled and packaged into standard Hadoop-compatible JAR files similar to other native Hadoop applications.
CBUI	See Co-Branded User Interface.
certificate	A credential that some AWS products use to authenticate AWS accounts and users. Also known as an X.509 certificate. The certificate is paired with a private key.
chargeable resources	Features or services whose use incurs fees. Although some AWS products are free, others include charges. For example, in an AWS CloudFormation stack (p. 158), AWS resources that have been created incur charges. The amount

	charged depends on the usage load. Use the Amazon Web Services Simple Monthly Calculator at http://calculator.s3.amazonaws.com/calc5.html to estimate your cost prior to creating instances, stacks, or other resources.
chargeback	A payment reversal that a bank issues when the buyer disputes a charge.
CIDR block	Classless Inter-Domain Routing. An Internet protocol address allocation and route aggregation methodology. See Also http://en.wikipedia.org/wiki/CIDR_notation.
CloudHub	See AWS VPN CloudHub.
cluster compute instance	A type of instance (p. 143) that provides a great amount of CPU power coupled with increased networking performance, making it well suited for High Performance Compute (HPC) applications and other demanding network-bound applications.
cluster placement group	A logical cluster compute instance (p. 136) grouping to provide lower latency and high-bandwidth connectivity between the instances.
CNAME	Canonical Name Record. A type of resource record in the Domain Name System (DNS) that specifies that the domain name is an alias of another, canonical domain name. More simply, it is an entry in a DNS table that lets you alias one fully qualified domain name to another.
Co-Branded service	The web service underlying the Co-Branded User Interface (p. 136).
Co-Branded User Interface	For the buyer: A series of web pages hosted by Amazon Payments that enables the buyer to authorize a payment. For the merchant: A series of web pages that Amazon Payments hosts so that a website owner can register a merchant's product for sale on the website.
co-branding	Running a business logo on a site or service that another company provides. Co-branding with Amazon Simple Pay, for example, is merely adding an independent merchant logo to each of the payment authorization web pages.
complaint	The event in which a recipient (p. 153) who does not want to receive an email message clicks "Mark as Spam" within the email client, and the Internet Service Provider (p. 144) sends a notification to Amazon SES.
compound query	A search request that specifies multiple search criteria using the Amazon CloudSearch structured search syntax.
condition	Any restriction or detail about a permission. The condition is *D* in the statement "A has permission to do B to C where D applies." Conditions are always optional.
conditional parameter	See mapping.
configuration API	The Amazon CloudSearch API that you use to create, configure, and manage search domains.
configuration template	A series of key–value pairs that define parameters for various AWS products so that AWS Elastic Beanstalk can provision them for an environment.
confirmation email	The email Amazon Payments sends to your customers to notify them that a price change you scheduled has taken effect.
consistency model	The method a service uses to achieve high availability. For example, it could involve replicating data across multiple servers in a data center. See Also eventual consistency.

consistent read	When data is written or updated successfully, all copies of the data are updated in all AWS regions. However, it takes time for the data to propagate to all storage locations. A consistent read returns a result that reflects any writes that received a successful response before the read request—regardless of the region. By contrast, an eventually consistent read returns data from only one region and might not show the most recent write information. See Also eventual consistency.
console	See AWS Management Console.
Consolidated Billing	See AWS Consolidated Billing.
cooldown period	Amount of time during which Auto Scaling does not allow the desired size of the Auto Scaling group (p. 133) to be changed by any other notification from a CloudWatch alarm (p. 130).
core node	An EC2 instance (p. 139) that runs Hadoop map and reduce tasks and stores data using the Hadoop Distributed File System (HDFS). Core nodes are managed by the master node (p. 147), which assigns Hadoop tasks to nodes and monitors their status. The EC2 instances you assign as core nodes are capacity that must be allotted for the entire job flow run. Because core nodes store data, you can't remove them from a job flow. However, you can add more core nodes to a running job flow. Core nodes run both the DataNodes and TaskTracker Hadoop daemons.
corpus	A collection of data that you want to search.
credentials	Also called *access credentials* or *security credentials*. In authentication and authorization, a system uses credentials to identify who is making a call and whether to allow the requested access. In AWS, these credentials are typically the access key ID (p. 129) and the secret access key (p. 156).
customer gateway	A router or software application on your side of a VPN tunnel that is managed by Amazon VPC. The internal interfaces of the customer gateway are attached to one or more devices in your home network. The external interface is attached to the VPG (p. 162) across the VPN tunnel.

D

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

dashboard	See service health dashboard.
database engine	The database software and version running on the DB instance (p. 137).
database name	The name of a database hosted in a DB instance (p. 137). A DB instance can host multiple databases, but databases hosted by the same DB instance must each have a unique name within that instance.
DB compute class	Size of the database compute platform used to run the instance.
DB instance	An isolated database environment running in the cloud. A DB instance can contain multiple user-created databases.
DB instance identifier	User-supplied identifier for the DB instance. The identifier must be unique for that user in an AWS region (p. 153).

DB parameter group	A container for database engine parameter values that apply to one or more DB instance (p. 137)s.
DB security group	A method that controls access to the DB instance (p. 137). By default, network access is turned off to DB instances. After ingress is configured for a security group, the same rules apply to all DB instances associated with that group.
DB snapshot	A user-initiated point backup of a DB instance.
Dedicated Instance	An instance that is physically isolated at the host hardware level and launched within a VPC.
Dedicated Reserved Instance	An option you purchase to guarantee that sufficient capacity will be available to launch Dedicated Instances into a VPC.
default deny	The default result from a policy (p. 150) in the absence of an allow (p. 130) or explicit deny (p. 140). For example: if a user (p. 161) requests to use Amazon SQS, but the only policy that applies to the user states that the user can use Amazon SimpleDB, then that policy results in a default deny.
delete marker	An object with a key and version ID, but without content. Amazon S3 inserts delete markers automatically into versioned buckets when an object is deleted.
deliverability	The likelihood that an email message will arrive at its intended destination.
deliveries	The number of emails, sent through Amazon SES, that were accepted by an Internet Service Provider (p. 144) for delivery to recipient (p. 153)s over a period of time.
detailed monitoring	Monitoring of AWS-provided metrics derived at a 1-minute frequency.
Description property	A property added to parameters, resources, resource properties, mappings, and outputs, to help you to document AWS CloudFormation template elements.
DevPay Activity	The location (Amazon DevPay Activity) where you manage the Amazon DevPay products you've created.
dimension	A name/value pair (for example, InstanceType=m1.small, or EngineName=mysql), that contains additional information to identify a metric.
discussion forums	A place where AWS users can post technical questions and feedback to help accelerate their development efforts and to engage with the AWS community. The discussion forums are located at http://aws.amazon.com/forums.
distributed cache	A Hadoop feature that allow you to transfer files from a distributed file system to the local file system. It can distribute data and text files as well as more complex types such as archives and JARs.
distribution	A link between an origin server (such as an Amazon S3 bucket) and a domain name, which CloudFront automatically assigns. Through this link, CloudFront identifies the object you have stored in your origin server (p. 149).
DKIM	DomainKeys Identified Mail. A standard that email senders use to sign their messages. ISPs use those signatures to verify that messages are legitimate. For more information, see http://www.dkim.org.
DNS	See Domain Name System (DNS).
document	Represents an item that can be returned as a search result in Amazon CloudSearch. Each document has a collection of fields that contain the data that

	can be searched or returned. The value of a field can be either a string or a number. Each document must have a unique ID and at least one field.
document batch	A collection of add and delete document operations for Amazon CloudSearch. You use the document service API to submit batches to update the data in your search domain.
document service API	The Amazon CloudSearch API that you use to submit document batches to update the data in a search domain.
document service endpoint	The URL that you connect to when sending document updates to an Amazon CloudSearch domain. Each search domain has a unique document service endpoint that remains the same for the life of the domain.
domain	All Amazon SimpleDB information is stored in domains. Domains are like tables that contain similar data. You can execute queries against a domain, but cannot execute joins between domains. See Also search domain.
Domain Name System (DNS)	A distributed naming system that associates network information with human-readable domain names on the Internet.
Donation button	An HTML-coded button to provide an easy and secure way for US-based, IRS-certified 501(c)3 nonprofit organizations to solicit donations.

E

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

EBS	See Amazon Elastic Block Store.
EC2	See Amazon Elastic Compute Cloud.
EC2 compute unit	An AWS standard for compute CPU and memory. This measure enables you to evaluate the CPU capacity of different EC2 instance types.
EC2 instance	In Amazon EC2, this is simply an instance. Other AWS services use the term EC2 instance to distinguish these instances from other types of instances they support.
edge location	A site that CloudFront uses to cache copies of your content for faster delivery to users at any location.
Edit Token API	A Co-Branded service (p. 136) API that enables you to view an existing token's details and to change the payment instrument (p. 149) for the token.
Elastic Block Store	See Amazon Elastic Block Store.
elastic IP address	A fixed (static) IP address that you have allocated in Amazon EC2 or Amazon VPC and then attached to an instance. Elastic IP addresses are associated with your account, not a specific instance. They are *elastic* because you can easily allocate, attach, detach, and free them as your needs change. Unlike traditional static IP addresses, elastic IP addresses allow you to mask instance or Availability Zone failures by rapidly remapping your public IP addresses to another instance.
Elastic Load Balancing	A web service that improves an application's availability by distributing incoming traffic between two or more EC2 instance (p. 139)s.

	See Also http://aws.amazon.com/elasticloadbalancing.
elastic network interface	An additional network interface that can be attached to an instance (p. 143). ENIs include a primary private IP address, one or more secondary private IP addresses, an elastic IP address (optional), a MAC address, membership in specified security groups, a description, and a source/destination check flag. You can create an ENI, attach it to an instance, detach it from an instance, and attach it to another instance.
endpoint	A URL that identifies a host and port as the entry point for a web service. Every web service request contains an endpoint. Most AWS products provide regional endpoints to enable faster connectivity. For more information, see Regions and Endpoints in the *Amazon Web Services General Reference*
	ElastiCache: The DNS name of a cache node (p. 135).
	Amazon RDS: The DNS name of a DB instance (p. 137).
	AWS CloudFormation: The DNS name or IP address of the server that receives an HTTP request.
endpoint port	ElastiCache: The port number used by a cache node (p. 135).
	Amazon RDS: The port number used by a DB instance (p. 137).
environment	A specific running instance of an application (p. 132). The application has a CNAME and includes an application version and a customizable configuration (which is inherited from the default container type).
environment configuration	A collection of parameters and settings that define how an environment and its associated resources behave.
ephemeral store	See instance store.
epoch	The date from which time is measured. For most Unix environments, the epoch is January 1, 1970.
eventual consistency	The method through which AWS products achieve high availability, which involves replicating data across multiple servers in Amazon's data centers. When data is written or updated and "Success" is returned, all copies of the data are updated. However, it takes time for the data to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change. Consistency is usually reached within seconds, but a high system load might increase this time.
eventually consistent read	See consistent read.
eviction	An *eviction* occurs when CloudFront deletes an object from an edge location (p. 139) before its expiration time. If an object in an edge location isn't frequently requested, CloudFront might evict the object (remove the object before its expiration date) to make room for objects that are more popular.
expiration	*Expiration* occurs when CloudFront stops serving an object from an edge location (p. 139). The next time the edge location needs to serve that object, CloudFront gets a new copy from the origin server (p. 149).
explicit deny	An *explicit deny* results from a statement that has effect=deny, assuming that any stated conditions are met. Example: *Deny all requests from Antarctica*. Any request that comes from Antarctica will always be denied no matter what any other policy (p. 150) might allow.

explicit launch permission — An Amazon Machine Image (p. 131) launch permission granted to a specific AWS account.

exponential backoff — A strategy that incrementally increases the wait between retry attempts in order to reduce the load on the system and increase the likelihood that repeated requests will succeed. For example, client applications might wait up to 400 milliseconds before attempting the first retry, up to 1600 milliseconds before the second, up to 6400 milliseconds (6.4 seconds) before the third, and so on.

expression — A numeric expression that you can use to control how search hits are sorted. You can construct Amazon CloudSearch expressions using numeric fields, other rank expressions, a document's default relevance _score, and standard numeric operators and functions. When you use the `sort` option to specify an expression in a search request, the expression is evaluated for each search hit and the hits are listed according to their expression values.

F

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

facet — An Amazon CloudSearch index field that represents a category that you want to use to refine and filter search results.

facet enabled — An Amazon CloudSearch index field option that enables facet information to be calculated for the field.

FBL — See feedback loop.

federated identity management — Allows individuals to sign in to different networks or services, using the same group or personal credentials to access data across all networks. With identity federation in AWS, external identities (federated users) are granted secure access to resources in an AWS account without having to create IAM users. These external identities can come from a corporate identity store (such as LDAP or Windows Active Directory) or from a third party (such as Login with Amazon, Facebook, or Google). AWS federation also supports SAML 2.0.

federated user — See federated identity management.

feedback loop — The mechanism by which a mailbox provider (for example, an Internet Service Provider (p. 144)) forwards a recipient (p. 153)'s complaint (p. 136) back to the sender (p. 156).

field weight — The relative importance of a text field in a search index. Field weights control how much matches in particular text fields affect a document's relevance _score.

filter — A criterion you specify to limit the results when you list or describe your Amazon EC2 resources.

filter query — A way to filter search results without affecting how the results are scored and sorted. Specified with the Amazon CloudSearch `fq` parameter.

FIM — See federated identity management.

format version — See template format version.

forums — See discussion forums.

FPS	See Amazon Flexible Payments Service.
function	See intrinsic function.
funding token	A payment token used to fund a prepaid instrument (p. 150).
fuzzy search	A simple search query that uses approximate string matching (fuzzy matching) to correct for typographical errors and misspellings.

G

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

geospatial search	A search query that uses locations specified as a latitude and longitude to determine matches and sort the results.
gibibyte	A contraction of giga binary byte, a gibibyte is 2^30 bytes or 1,073,741,824 bytes. A gigabyte is 10^9 or 1,000,000,000 bytes.

H

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

Hadoop	See http://hadoop.apache.org.
hard bounce	A persistent email delivery failure such as "mailbox does not exist."
hardware VPN	A hardware-based IPsec VPN connection over the Internet.
HDFS	Hadoop Distributed File System. The HDFS file system stores large files across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts.
health check	A system call to check on the health status of each instance in an Auto Scaling group.
high-quality email	Email that recipients find valuable and want to receive. Value means different things to different recipients and can come in the form of offers, order confirmations, receipts, newsletters, etc.
highlights	Excerpts returned with Amazon CloudSearch results that show where the search terms appear within the text of the matching documents.
highlight enabled	An Amazon CloudSearch index field option that enables matches within the field to be highlighted.
hit	A document that matches the criteria specified in a search request. Also referred to as a *search result*.
HIT	See Human Intelligence Task.
Hive	An open source, data warehouse and analytic package that runs on top of Hadoop. Hive scripts use an SQL-like language called Hive QL (query language) that

	abstracts the MapReduce programming model and supports typical data warehouse interactions.
HMAC	Hash-based Message Authentication Code. A specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret key. You can use it to verify both the data integrity and the authenticity of a message at the same time. AWS calculates the HMAC using a standard, cryptographic hash algorithm, such as SHA-256.
hosted zone	A collection of resource record sets that Amazon Route 53 hosts. Like a traditional DNS zone file, a hosted zone represents a collection of records that are managed together under a single domain name.
Human Intelligence Task	A task that a Requester (p. 154) submits to Mechanical Turk for workers (p. 163) to perform. A HIT represents a single, self-contained task, for example, "Identify the car color in the photo." HITs contain all of the information a worker needs to answer a question, including the kinds of answers you would consider valid.
HVM virtualization	Hardware Virtual Machine virtualization. Lets the guest VM run as though it is on a native hardware platform, except that it still uses para-virtual (PV) network and storage drivers for improved performance.

I

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

image	See Amazon Machine Image.
import/export station	A machine that uploads or downloads your data to, or from, Amazon S3.
import log	A report that contains details about how AWS Import/Export processed your data.
inbound request	A button click or other form request to Amazon Payments.
index	See search index.
index field	A name-value pair that is included in an Amazon CloudSearch domain's index. An index field can contain text or numeric data, dates, or a location.
indexing options	Configuration settings that define an Amazon CloudSearch domain's index fields, how document data is mapped to those index fields, and how the index fields can be used.
instance	A copy of an Amazon Machine Image running as a virtual server in the AWS cloud.
instance family	A general instance type (p. 144) grouping using either storage or CPU capacity.
instance group	A Hadoop cluster contains one master instance group that contains one master node (p. 147), a core instance group containing one or more core node (p. 137) and an optional task node (p. 160) instance group, which can contain any number of task nodes.
instance store	Disk storage that is physically attached to the host computer for an EC2 instance, and therefore has the same lifespan as the instance. When the instance terminates, you lose any data in the instance store.

instance store-backed AMI	Instances launched from this type of AMI use an instance store volume as the root device. Compare this with instances launched from Amazon EBS-backed AMIs, which use an Amazon EBS volume as the root device.
instance type	A specification that defines the memory, CPU, storage capacity, and hourly cost for an instance. Some instance types are designed for standard applications, whereas others are designed for CPU-intensive, memory-intensive applications, and so on.
Instant Payment Notification	Also called IPN. A notification that Amazon Payments sends whenever a payment, refund, or reserved payment completes successfully or fails. The caller must provide Amazon Payments with an endpoint that we can send the IPN to. A notification (separate from the buyer redirect) that is sent whenever a payment, refund, or reserved payment completes successfully or fails. The developer must host this notification service and provide Amazon Simple Pay with an IPN response URL.
Internet gateway	Connects a network to the Internet. You can route traffic for IP addresses outside your VPC (p. 162) to the Internet gateway.
Internet Service Provider	A company that provides subscribers with access to the Internet. Many ISPs are also mailbox provider (p. 146)s. Mailbox providers are sometimes referred to as ISPs, even if they only provide mailbox services.
intrinsic function	A special action in a template that assigns values to properties not available until runtime. These functions follow the format *Fn::Attribute*, such as `Fn::GetAtt`. Arguments for intrinsic functions can be parameters, pseudo parameters, or the output of other intrinsic functions.
IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through network address translation (NAT): a private IP address (following RFC 1918) and a public IP address. Instances launched in a VPC are assigned only a private IP address. Instances launched in your default VPC are assigned both a private IP address and a public IP address.
ISP	See Internet Service Provider.
issuer	The issuer is the person who writes a policy to grant permissions to a resource. The issuer (by definition) is always the resource owner. AWS does not permit Amazon SQS users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.
item	Similar to rows on a spreadsheet, items represent individual objects that contain one or more value-attribute pairs.
item name	An identifier for an item. The identifier must be unique within the domain (p. 139).

J

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

job flow	A job flow specifies the complete processing of the data. It's comprised of one or more steps, which specify all of the functions to be performed on the data.

job ID
A five-character, alphanumeric string that uniquely identifies a storage device in your shipment. AWS issues the job ID in response to a `CREATE JOB` email command.

job prefix
The AWS Import/Export process generates a log file. The log file name always ends with the phrase *import-log-* followed by your Job ID. There is a remote chance that you already have an object with this name. To avoid a key collision, you can add an optional prefix to the log file.
See Also key prefix.

JSON
JavaScript Object Notation. A lightweight data-interchange format. For information about JSON, see http://www.json.org/.

junk folder
The location where email messages that various filters determine to be of lesser value are collected so that they do not arrive in the recipient (p. 153)'s inbox, but are still accessible to the recipient. This is also referred to as a spam (p. 158) or bulk folder.

K

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

key
A credential that identifies an AWS account or user to AWS (such as the AWS secret access key (p. 156)).
Amazon S3, Amazon EMR: The unique identifier for an object in a bucket. Every object in a bucket has exactly one key. Because a bucket and key together uniquely identify each object, you can think of Amazon S3 as a basic data map between the *bucket* + *key*, and the object itself. You can uniquely address every object in Amazon S3 through the combination of the web service endpoint, bucket name, and key, for example:
`http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl`, where `doc` is the name of the bucket, and `2006-03-01/AmazonS3.wsdl` is the key.
AWS Import/Export: The name of an object in Amazon S3. It is a sequence of Unicode characters whose UTF-8 encoding cannot exceed 1024 bytes. If a key, for example, logPrefix + import-log-JOBID, is longer than 1024 bytes, AWS Elastic Beanstalk returns an `InvalidManifestField` error.
IAM: In the context of writing a policy (p. 150): A specific characteristic that is the basis for restricting access (such as the current time, or the IP address of the requester).

key pair
A set of security credentials you use to prove your identity electronically. A key pair consists of a private key and a public key.

key prefix
A logical grouping of the objects in a bucket (p. 134). The prefix value is similar to a directory name that enables you to store similar data under the same directory in a bucket.

L

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

launch configuration
A set of descriptive parameters used to create new EC2 instances in an Auto Scaling activity.

	A template that an Auto Scaling group (p. 133) uses to launch new EC2 instances. The launch configuration contains information such as the Amazon Machine Image (p. 131) ID, the instance type, key pairs, security groups, and block device mappings, among other configuration settings.
launch permission	An Amazon Machine Image (p. 131) (AMI) attribute that allows users to launch an AMI.
lifecycle	The lifecycle state of the EC2 instance (p. 139) contained in an AutoScalingGroup. EC2 instances progress through several states over their lifespan; these include *Pending*, *InService*, *Terminating* and *Terminated*.
load balancer	A load balancer is a combination of a DNS name and a set of ports, which together provide a destination for all requests intended for your application. A load balancer can distribute traffic to multiple application instances across every Availability Zone (p. 133) within a region (p. 153). Load balancers can span multiple Availability Zones within an Amazon EC2 region, but they cannot span multiple regions.
logical name	A case-sensitive unique string within an AWS CloudFormation template that identifies a resource (p. 154), mapping (p. 147), parameter, or output. In an AWS CloudFormation template, each parameter, resource, property, mapping, and output must be declared with a unique logical name. You use the logical name when dereferencing these items using the `Ref` function.

M

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

machine utilization	The amount of machine capacity used to complete a particular request (for example SELECT, GET, PUT, and so on), normalized to the hourly capacity of a standard processor. Machine utilization is measured in machine hour increments.
Mail Transfer Agent (MTA)	Software that transports email messages from one computer to another by using a client-server architecture.
mailbox provider	An organization that provides email mailbox hosting services. Mailbox providers are sometimes referred to as Internet Service Provider (p. 144)s, even if they only provide mailbox services.
mailbox simulator	A set of email addresses that you can use to test an Amazon SES-based email sending application without sending messages to actual recipients. Each email address represents a specific scenario (such as a bounce or complaint) and generates a typical response that is specific to the scenario.
main route table	The default route table that any new VPC subnet uses for routing. You can associate a subnet with a different route table of your choice. You can also change which route table is the main route table.
manifest	When sending a *create job* request for an import or export operation you describe your job in a text file called a manifest. The manifest file is a YAML-formatted file that specifies how to transfer data between your storage device and the AWS cloud.
MapReduce	See http://hadoop.apache.org/docs/r1.2.0/mapred_tutorial.html.
mapper	An executable that splits the raw data into key/value pairs. The reducer uses the output of the mapper, called the *intermediate results*, as its input.

mapping	A way to add conditional parameter values to an AWS CloudFormation template. You specify mappings in the template's optional Mappings section and retrieve the desired value using the `FN::FindInMap` function.
marker	See pagination.
marketplace	Amazon FPS: An environment in which the caller charges a fee for facilitating a transaction between a sender and a recipient. Amazon Simple Pay: A feature that allows a third party to charge for hosting a merchant's offers and facilitating payment.
Marketplace button	An HTML-coded button to display and sell the goods of other sellers, optionally charging them a fee for the service.
master node	A process running on an Amazon Machine Image (p. 131) that keeps track of the work its core and task nodes complete.
maximum price	The maximum price you will pay to launch one or more Spot Instances. If your maximum price exceeds the current Spot Price (p. 158) and your restrictions are met, Amazon EC2 launches instances on your behalf.
maximum send rate	The maximum number of emails that you can send per second using Amazon SES.
member resources	See resource.
message ID	Amazon SES: A unique identifier that is assigned to every email message that is sent. Amazon SQS: The identifier returned when you send a message to a queue.
metadata	Amazon S3, Amazon EMR: A set of name/value pairs that describe the object. These include default metadata such as the date last modified and standard HTTP metadata such as Content-Type. Users can also specify custom metadata at the time they store an object. Amazon EC2: Data about an EC2 instance (p. 139) that the instance can retrieve to determine things about itself, such as, the instance type, the IP address, and so on.
metric	An element of time-series data defined by a unique combination of exactly one namespace, exactly one metric name, and between zero and ten dimensions. Metrics and the statistics derived from them are the basis of Amazon CloudWatch.
metric name	The primary identifier of a metric, used in combination with a namespace and optional dimensions.
MFA	See AWS Multi-Factor Authentication.
micro instance	A type of EC2 instance (p. 139) that is more economical to use if you have occasional bursts of high CPU activity.
MIME	See Multipurpose Internet Mail Extensions (MIME).
MTA	See Mail Transfer Agent (MTA).
Multi-AZ deployment	A primary DB instance (p. 137) that has a synchronous standby replica in a different Availability Zone (p. 133). The primary DB instance is synchronously replicated across Availability Zones to the standby replica.
Multi-Factor Authentication	See AWS Multi-Factor Authentication.

multi-use token — A usage-based payment instrument (p. 149) that allows the caller (p. 135) to charge the sender (p. 156) multiple times, without requiring the sender to repeatedly authorize the payments (compare to a single-use token (p. 157)).

Multi-Use Token API — The Co-Branded service (p. 136) API that creates a multi-use token (p. 148).

multi-valued attribute — An attribute with more than one value.

multipart upload — A feature that allows you to upload a single object as a set of parts.

Multipurpose Internet Mail Extensions (MIME) — An Internet standard that extends the email protocol to include non-ASCII text and non-text elements like attachments.

Multitool — A Cascading (p. 135) application that provides a simple command-line interface for managing large datasets.

N

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

namespace — An abstract container that provides context for the items (names, or technical terms, or words) it holds, and allows disambiguation of homonym items residing in different namespaces.

NAT — Network address translation.

NAT instance — An instance that is configured to perform NAT (p. 148) in a VPC. A NAT instance enables private instances in the VPC to initiate Internet-bound traffic without being directly reachable from the Internet.

network ACL — An optional layer of security that acts as a firewall for controlling traffic in and out of a subnet. You can associate multiple subnets with a single network ACL, but a subnet can be associated with only one network ACL at a time.

node — After an Amazon Machine Image (p. 131) is launched, the resulting running system is referred to as a node. All instances based on the same AMI are identical at start-up. Any information about the node is lost when the node terminates or fails.

NoEcho — A property of AWS CloudFormation parameters that will prevent the otherwise default reporting of names and values of a template parameter. Declaring the *NoEcho* property causes the parameter value to be masked with asterisks in the report by the `cfn-describe-stacks` command.

notification email — The email Amazon Payments sends to your customers to notify them of an upcoming price change you've scheduled.

null object — A null object is one whose version ID is null. Amazon S3 adds a null object to a bucket when versioning (p. 162) for that bucket is suspended. It is possible to have only one null object for each key in a bucket.

O

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

object	Amazon S3: The fundamental entity type stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3. CloudFront: Any entity that can be served either over HTTP or a version of RTMP.
on-demand instance	An Amazon EC2 pricing option that charges you for compute capacity by the hour with no long-term commitment.
one time payment	An Amazon FPS payment processed with a single-use token (p. 157). After the payment is made, that token is no longer valid.
operation	An API function. Also called an *action*.
order pipeline	The process that an order passes through between the time a customer selects an item and the time that customer's payment instrument (p. 149) is charged.
origin access identity	Also called OAI. A virtual identity you use when giving your distribution permission to fetch a private object from your origin server (Amazon S3 bucket).
origin server	The Amazon S3 bucket or custom origin containing the definitive original version of the content you deliver through CloudFront.
outbound notification	A response from Amazon Payments to your Amazon FPS (or Amazon Simple Pay) application via a Return URL or an Instant Payment Notification (IPN).

P

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

pagination	Some APIs that return a potentially large list of records can return a subset by using a value to set the maximum number of returned records. They then provide a marker, which identifies the last record returned so that in a subsequent call, the user can get the next sequence of records.
paid AMI	An Amazon Machine Image (AMI) that you sell to other Amazon EC2 users using Amazon DevPay.
part	In a multipart upload request, each part is a contiguous portion of the object's data.
passive authorization	A passive authorization happens when you inform customers at least 14 days in advance of a price change for your product, and they don't take any action; the price change is accepted automatically.
PAT	Port address translation.
payment instrument	The method of payment a customer chooses to use with Amazon Payments. These include credit cards (CC), Amazon Payments account balance (ABT), and bank account debits (ACH).
payment method failure	An error caused by an irregularity in the customer's chosen payment method, such as an insufficient bank balance or an expired credit card.
period	See sampling period.
permission	A statement within a policy (p. 150) that allows or disallows access to a particular resource. You can state any permission like this: "A has permission to do B to C

	where D applies." For example, Jane (A) has permission to read messages (B) from John's Amazon SQS queue (C), as long as she asks to receive only a maximum of 10 messages from the queue at a time (D). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission.
persistent identifier	Also called PID. An encoded string that represents the relationship between a customer and the owner of Amazon DevPay products. After a customer purchases one of your products, you can use the PID to confirm the status of the customer's subscription to the product.
persistent storage	A long-term data storage solution. Options within AWS are: Amazon S3, Amazon EBS, and Amazon SimpleDB.
physical name	A unique label AWS CloudFormation assigns to each resource when creating a stack (p. 158). Some AWS CloudFormation commands accept the physical name as a value with the `--physical-name` parameter.
Pig	An open-source Apache library that runs on top of Hadoop. The library takes SQL-like commands written in a language called Pig Latin and converts those commands into MapReduce job flows.
policy	A policy is the formal description of the permissions for a resource. The access policy language distinguishes between a *policy* and a *statement*. A policy is the complete document that can contain many different permissions for a given resource. A statement is the description of an individual permission. Therefore a policy can contain multiple statements. For example, a policy could specify that Jane can use John's queue (one statement), and Bob cannot use John's queue (another statement). Auto Scaling: An object that stores the information needed to launch or terminate instances for an Auto Scaling group. Executing the policy causes instances to be launched or terminated. You can configure an alarm (p. 130) to invoke an Auto Scaling policy.
postpaid credit instrument	A payment instrument (p. 149) that is like a credit card used to make incremental purchases on your website. As purchases are made, the accumulated debt on the postpaid credit instrument increases until a credit limit is reached, or until you have arranged to make a settlement of the debt, such as a monthly payment.
postpaid payment token	A payment token used whenever a buyer (p. 134) wants to make a purchase using a postpaid credit instrument (p. 150).
Postpaid Token API	The Co-Branded service (p. 136) API that creates a postpaid payment token (p. 150) that a sender uses as a payment instrument (p. 149) and then funds, much like a credit card.
pre-signed URL	A URL that uses query string authentication (p. 152).
prefix	See job prefix.
Premium Support	A one-on-one, fast-response support channel that AWS customers can subscribe to for support for AWS infrastructure services. See Also http://aws.amazon.com/premiumsupport/.
prepaid instrument	A payment instrument (p. 149) that is like a gift card with an associated prepaid balance. A buyer (p. 134) can purchase a prepaid instrument on your website and use it to make incremental payments over a period of time, in line with whatever constraints that you or the buyer previously set up.

prepaid payment token	A payment token used whenever a buyer (p. 134) wants to make a purchase using a prepaid instrument (p. 150).
Prepaid Token API	The Co-Branded service (p. 136) API that creates a prepaid payment token (p. 151) that a sender (p. 156) funds and then uses as a payment instrument (p. 149).
principal	The principal is the person or persons who receive the permission in the policy (p. 150). The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (that is, you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address. The concept of principals doesn't apply to a IAM policy, because these policies are attached to users or groups.
private IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through Network Address Translation (NAT): a private address (following RFC 1918) and a public address. *Exception:* Instances launched in Amazon VPC are assigned only a private IP address.
private subnet	A VPC subnet whose instances cannot be reached from the Internet.
product activation	See activation.
product code	The product code is an eight-character string that identifies your registered product to AWS.
product identification token	See product token.
product token	The product token is a long encoded string that identifies the product to AWS. You might also see the product token referred to as the *product identification token*.
properties	See resource property.
property rule	A JSON (p. 145)-compliant markup standard for declaring properties, mappings, and output values in an AWS CloudFormation template.
provisioned IOPS	A storage option designed to deliver fast, predictable, and consistent I/O performance. When you specify an IOPS rate while creating a DB Instance, Amazon RDS provisions that IOPS rate for the lifetime of the DB Instance.
pseudo parameter	A predefined setting, such as `AWS:StackName` that can be used in AWS CloudFormation templates without having to declare them. You can use pseudo parameters anywhere you can use a regular parameter.
public AMI	An Amazon Machine Image (p. 131) that all AWS accounts have permission to launch.
public data set	A large set of public data that can be seamlessly integrated into AWS cloud-based applications. Amazon stores public data sets at no charge to the community and, like all AWS services, users pay only for the compute and storage they use for their own applications. These data sets currently include data from the Human Genome Project, the U.S. Census, Wikipedia, and other sources. See Also http://aws.amazon.com/publicdatasets.
public IP address	All EC2 instances are assigned two IP addresses at launch, which are directly mapped to each other through Network Address Translation (NAT): a private

address (following RFC 1918) and a public address. *Exception:* Instances launched in Amazon VPC are assigned only a private IP address.

public subnet — A subnet whose instances can be reached from the Internet.

purchase URL — The URL your customers use to purchase your product. When you advertise your product, you provide the purchase URL as the sign-up link for customers to use.

Q

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

Qualification — A property associated with a worker (p. 163) that represents that worker's skill, ability, or reputation. A Requester (p. 154) can use Qualifications to control which workers can perform HITs. Each worker can have multiple Qualifications.

Qualification requirements — A Human Intelligence Task (p. 143) can have Qualification requirements that a worker's Qualifications (q.v.) must meet before the worker can accept that HIT.

Qualification test — A form, similar to a HIT, containing a set of questions that the worker must complete successfully to receive a particular Qualification (p. 152).

Qualification type — Just as each worker (p. 163) has one or more Qualification (p. 152), each Human Intelligence Task (p. 143) has one or more Qualification type. These types specify what Qualifications the worker must have.

Query — A type of HTTP-based request interface that generally uses only the GET or POST HTTP method and a query string with parameters.
See Also REST, REST-Query.

query string authentication — An AWS feature that lets you place the authentication information in the HTTP request query string instead of in the Authorization header. For example: with Amazon DevPay, query string authentication enables your product to give anyone easy, URL-based access to objects in the customer's bucket.

queue — A sequence of messages or jobs held in temporary storage awaiting transmission or processing.

queue URL — A URL that uniquely identifies a queue.

quota — Amazon RDS: The maximum number of DB instance (p. 137)s and available storage you can use.

ElastiCache: The maximum number of the following items:

- The number of cache clusters for each AWS account
- The number of cache nodes per cache cluster
- The total number of cache nodes per AWS account across all cache clusters created by that AWS account

R

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

range GET	A range GET specifies a byte range of data to get for a download. If an object is large, you can break up a download into smaller units by sending multiple range GET requests that each specify a different byte range to GET.
raw email	A type of *sendmail* request that allows you to specify the email headers and MIME types.
RDS	See Amazon Relational Database Service.
read replica	An active copy of another DB instance. Any updates to the data on the source DB instance are replicated to the read replica DB instance using the built-in replication feature of MySQL 5.1.
receipt handle	An identifier you get when you receive a message from the queue. This identifier is required to delete a message from the queue or when changing a message's visibility timeout.
receiver	The entity that consists of the network systems, software, and policies that manage email delivery for a recipient (p. 153).
recipient	Amazon FPS: A seller who receives a payment from a buyer (sender) in exchange for a service or product. Amazon SES: The person or entity receiving an email message. For example, a person named in the "To" field of a message.
recipient token	A general term to cover all types of payment tokens associated with recipients (for example, single-use token, multi-use token, and recurring-use token).
recurring payment	An Amazon FPS payment processed with a recurring payment token. Payments can occur periodically using the same payment token. The token is valid until it expires.
recurring-use token	A type of multi-use token (p. 148) that is restricted to a predetermined fixed amount and a regular payment interval.
Recurring-Use Token API	The Co-Branded service (p. 136) API that creates a recurring-use token (p. 153).
redirect URL	The page on your own website that you want customers to see at the end of the purchase process for your product. You provide the URL when you register the product with Amazon DevPay.
reducer	An executable in the MapReduce process that uses the intermediate results from the mapper and processes them into the final output.
reference	A means of inserting a property from one AWS resource into another. For example, you could insert an Amazon EC2 security group property into an Amazon RDS resource.
region	A named set of AWS resources in the same geographical area. A region comprises at least two Availability Zones.

reply path	The email address to which an email reply is sent. This is different from the return path (p. 155).
reputation	1. An Amazon SES metric, based on factors that might include bounces, complaints, and other metrics, regarding whether or not a customer is sending high-quality emails. 2. A measure of confidence, as judged by an Internet Service Provider (p. 144) or other entity that an IP address that they are receiving emails from is not the source of spam (p. 158).
requester	A requester is a person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Can A do B to C where D applies?" In this question, the requester is A. See Also Requester.
Requester	(Note capitalization) A company, organization, or person that creates and submits tasks (a Human Intelligence Task (p. 143)) to Mechanical Turk; for workers (p. 163) to perform.
Requester Pays	An Amazon S3 feature that allows a bucket owner (p. 134) to specify that anyone who requests access to objects in a particular bucket must pay the data transfer and request costs.
reservation	A collection of EC2 instances started as part of the same launch request. Not to be confused with a Reserved Instance (p. 154).
reserve	The amount that is held in reserve against a credit card, but not charged. Later, the transaction is settled, that is charged (typically after the product is actually shipped).
Reserved Instance	A pricing option that lets you make a low, one-time payment for each instance to reserve and receive a significant discount on the hourly usage charge for that instance.
Reserved Instance Marketplace	Matches sellers who have reserved capacity that they no longer need with buyers who are looking to purchase additional capacity. Reserved Instances that you purchase from third-party sellers will have less than a full standard term remaining and can be sold at different upfront prices. The usage or reoccurring fees will remain the same as the fees set when the Reserved Instances were originally purchased. Full standard terms for Reserved Instances available from AWS run for one year or three years.
resource	1. The objects you work with on AWS. This includes buckets, domains, instances, queues, and so on. 2. Tools, code, and documents that AWS provides to support users. 3. An object that the principal (p. 151) requests access to. The resource is C in the statement "A has permission to do B to C where D applies." 4. A required element of an AWS CloudFormation stack (p. 158). Each stack contains at least one resource, such as an Auto Scaling LaunchConfiguration. All resources in a stack must be created successfully for the stack to be created.
resource property	A value required when including an AWS resource in an AWS CloudFormation stack (p. 158). Each resource may have one or more properties associated with it. For example, an `AWS::EC2::Instance` resource may have a `UserData`

property. In an AWS CloudFormation template, resources must declare a properties section, even if the resource has no properties.

resource record — Also called *resource record set*. Standard DNS terminology. See Also http://en.wikipedia.org/wiki/Domain_Name_System.

REST — A type of HTTP-based request interface that generally uses only the GET or POST HTTP method and a query string with parameters. Sometimes known as Query. In some implementations of a REST interface, other HTTP verbs besides GET and POST are used.

REST-Query — Also known as Query or HTTP Query. This is a type of HTTP request that generally uses only the GET or POST HTTP method and a query string with parameters. Compare this with REST, which is a type of HTTP request that uses any HTTP method (GET, DELETE, POST, etc.), a resource, HTTP headers, and possibly a query string with parameters.

return enabled — An Amazon CloudSearch index field option that enables the field's values to be returned in the search results.

return path — The email address to which bounced emails are returned. The return path is specified in the header of the original email. This is different from the reply path (p. 154).

reward — The money a Requester (p. 154) pays a worker (p. 163) for satisfactory work done on the Requester's Human Intelligence Task (p. 143)s.

rollback — A return to a previous state that follows the failure to create an object, such as AWS CloudFormation stack (p. 158). All resources associated with the failure are deleted during the rollback. For AWS CloudFormation, you can override this behavior using the `--disable-rollback` option on the command line.

root device volume — Contains the image used to boot the instance. If you launched the instance from an AMI backed by instance store, this is an instance store volume created from a template stored in Amazon S3. If you launched the instance from an AMI backed by Amazon EBS, this is an Amazon EBS volume created from an Amazon EBS snapshot.

route table — A set of routing rules that controls the traffic leaving any subnet that is associated with the route table. You can associate multiple subnets with a single route table, but a subnet can be associated with only one route table at a time.

S

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

sampling period — A defined duration of time, such as one minute, over which CloudWatch computes a statistic (p. 158).

sandbox — A testing location where you can test the functionality of your application without affecting production, incurring charges, or purchasing products.

Amazon SES: An Amazon SES environment that is designed for developers to test and evaluate the service. In the sandbox, you have full access to the Amazon SES API, but you can only send messages to verified email addresses and the mailbox simulator. To get out of the sandbox, you need to apply for production

	access. Accounts in the sandbox also have lower sending limits (p. 157) than production accounts.
scaling activity	A process that changes the size, configuration, or makeup of an Auto Scaling group (p. 133) by launching or terminating instances. For more information, see Auto Scaling Concepts in the Auto Scaling Developer Guide.
search API	The Amazon CloudSearch API that you use to submit search requests to a search domain.
search domain	Encapsulates your searchable data and the search instances that handle your search requests. You typically set up a separate Amazon CloudSearch domain for each different collection of data that you want to search.
search domain configuration	An Amazon CloudSearch domain's indexing options, analysis schemes, expressions, suggesters, access policies, and scaling and availability options.
search enabled	An Amazon CloudSearch index field option that enables the field data to be searched.
search endpoint	The URL that you connect to when sending search requests to a search domain. Each Amazon CloudSearch domain has a unique search endpoint that remains the same for the life of the domain.
search index	A representation of your searchable data that facilitates fast and accurate data retrieval.
search instance	A compute resource that indexes your data and processes search requests. An Amazon CloudSearch domain has one or more search instances, each with a finite amount of RAM and CPU resources. As your data volume grows, more search instances or larger search instances are deployed to contain your indexed data. When necessary, your index is automatically partitioned across multiple search instances. As your request volume or complexity increases, each search partition is automatically replicated to provide additional processing capacity.
search request	A request that is sent to an Amazon CloudSearch domain's search endpoint to retrieve documents from the index that match particular search criteria.
search result	A document that matches a search request. Also referred to as a *search hit*.
secret access key	A key that is used in conjunction with the access key ID (p. 129) to cryptographically sign programmatic AWS requests. Signing a request identifies the sender and prevents the request from being altered. You can generate secret access keys for your AWS account, individual IAM users, and temporary sessions.
security group	A named set of allowed inbound network connections for an instance. (Security groups in Amazon VPC also include support for outbound connections.) Each security group consists of a list of protocols, ports, and IP address ranges. A security group can apply to multiple instances, and multiple groups can regulate a single instance.
seller	Amazon FPS, Amazon Simple Pay: A seller receives money from a buyer in exchange for a service or product. Amazon Simple Pay: Individual who receives a payment from a buyer using an Amazon Simple Pay button. The seller receives money from a buyer in exchange for a service or product.
sender	Amazon FPS: A sender (also known as the buyer) pays a recipient for a product or service.

	Amazon SES: The person or entity sending an email message.
Sender ID	A Microsoft-controlled version of SPF. An email authentication and anti-spoofing system. For more information about Sender ID, go to http://wikipedia.org/wiki/Sender_ID.
sender token	A general term to cover all types of payment tokens that can be associated with a sender (for example, single-use token, recurring use token, postpaid token, and so on).
sending limits	The sending quota (p. 157) and maximum send rate (p. 147) that are associated with every Amazon SES account.
sending quota	The maximum number of emails that you can send using Amazon SES in a 24-hour period.
service endpoint	See endpoint.
server-side signature verification	Method used to validate Instant Payment Notification (IPN) and Return URL responses.
service health dashboard	A web page showing up-to-the-minute information about AWS service availability. The dashboard is located at http://status.aws.amazon.com.
settle	To complete a transaction that has been reserved. If you don't charge the sender (p. 156) immediately when the purchase is initiated (and instead reserve the amount against the sender's credit card), you settle the transaction later, typically after you ship the product to the sender. In FPS, *Settling* moves the reserved amount from the sender to the recipient. Amazon Simple Pay does not support settling purchases. You must use the Amazon Simple Pay API `Settle` to implement that functionality.
settlement token	A payment token used to settle the debt accumulated with a postpaid credit instrument (p. 150).
SHA	Secure Hash Algorithm. SHA1 is an earlier version of the algorithm, which AWS has deprecated in favor of SHA256.
shared AMI	An Amazon Machine Image (p. 131) that a developer builds and makes available for others to use.
shutdown action	A predefined bootstrap action that launches a script that executes a series of commands in parallel before terminating the job flow.
signature	Refers to a *digital signature*, which is a mathematical way to confirm the authenticity of a digital message. AWS uses signatures to authenticate the requests you send to our web services. For more information, to http://aws.amazon.com/security.
SIGNATURE file	A file you copy to the root directory of your storage device. The file contains a job ID, manifest file, and a signature.
Simple Mail Transfer Protocol	See SMTP.
Single-AZ DB Instance	A standard (non-Multi-AZ) DB instance (p. 137) that is deployed in one Availability Zone (p. 133), without a standby replica in another Availability Zone. See Also Multi-AZ deployment.
single-use token	A payment instrument (p. 149) that allows the caller (p. 135) to charge the sender (p. 156) only once (compare to a multi-use token (p. 148)).

Single-Use Token API	The Co-Branded service (p. 136) API that creates a single-use token (p. 157).
single-valued attribute	An attribute with one value.
sloppy phrase search	A search for a phrase that specifies how close the terms must be to one another to be considered a match.
SMTP	Simple Mail Transfer Protocol. The standard that is used to exchange email messages between internet hosts for the purpose of routing and delivery.
snapshot	Amazon Elastic Block Store (p. 131) creates *snapshots* or backups of your volumes and stores them in Amazon S3. You can use these snapshots as the starting point for new Amazon EBS volumes or to protect your data for long-term durability.
soft bounce	A temporary email delivery failure such as "mailbox full."
software VPN	A software appliance-based VPN connection over the Internet.
sort enabled	An Amazon CloudSearch index field option that enables a field to be used to sort the search results.
source/destination checking	A security measure to verify that an EC2 instance is the origin of all traffic that it sends and the ultimate destination of all traffic that it receives, that is, that the instance is not relaying traffic. Source/destination checking is enabled by default. For instances that function as gateways, such as VPC NAT instances, source/destination checking must be disabled.
spam	Unsolicited bulk email.
spamtrap	An email address that is set up by an anti-spam (p. 158) entity, not for correspondence, but to monitor unsolicited email. This is also called a *honeypot*.
SPF	Sender Policy Framework. A standard for authenticating email. See Also http://www.openspf.org.
Spot Instance	A type of EC2 instance (p. 139) that you can bid on to take advantage of unused Amazon EC2 capacity.
Spot Price	The price for a Spot Instance (p. 158) at any given time. If your maximum price exceeds the current price and your restrictions are met, Amazon EC2 launches instances on your behalf.
stack	AWS CloudFormation: A collection of AWS resources you create and delete as a single unit. AWS OpsWorks: A set of instances you manage collectively, typically because they have a common purpose such as serving PHP applications. A stack serves as a container and handles tasks that apply to the group of instances as a whole, such as managing applications and cookbooks.
Standard button	An HTML-coded button to offer Amazon Simple Pay as a standalone payment method for one-time purchases.
station	A place at an AWS facility where we transfer your AWS Import/Export data on to, or off of, your storage device.
statistic	One of five functions of the values submitted for a given sampling period (p. 155). These functions are "Maximum", "Minimum," "Sum," "Average," and "SampleCount."
stem	The common root or substring shared by a set of related words.

stemming	The process of mapping related words to a common stem. This enables matching on variants of a word. For example, a search for "horse" could return matches for horses, horseback, and horsing, as well as horse. Amazon CloudSearch supports both dictionary based and algorithmic stemming.
step	A single function applied to the data in a job flow (p. 144). The sum of all steps comprises a job flow.
step type	The type of work done in a step. There are a limited number of step types, such as moving data from Amazon S3 to Amazon EC2 or from Amazon EC2 to Amazon S3.
sticky session	A feature of the load balancer that binds a user's session to a specific application instance so that all requests coming from the user during the session are sent to the same application instance. By contrast, a load balancer defaults to route each request independently to the application instance with the smallest load.
stopping	The process of filtering stop words from an index or search request.
stopword	A word that is not indexed and is automatically filtered out of search requests because it is either insignificant or so common that including it would result in too many matches to be useful. Stop words are language-specific.
streaming	Amazon EMR: A utility that comes with Hadoop that enables you to develop MapReduce executables in languages other than Java. CloudFront: The ability to use a media file in real time—as it is transmitted in a steady stream from a server.
streaming distribution	A special kind of distribution (p. 138) that serves streamed media files using a Real Time Messaging Protocol (RTMP) connection.
string-to-sign	Before you calculate an HMAC signature, you first assemble the required components in a canonical order. The pre-encrypted string is the string-to-sign.
structured query	Search criteria specified using the Amazon CloudSearch structured query language. You use the structured query language to construct compound queries that use advanced search options and combine multiple search criteria using Boolean operators.
subnet	A segment of the IP address range of a VPC (p. 162) that EC2 instances can be attached to. You can create subnets to group instances according to security and operational needs.
Subscription button	An HTML-coded button that enables an easy way to charge customers a recurring fee.
suggester	Specifies an Amazon CloudSearch index field you want to use to get autocomplete suggestions and options that can enable fuzzy matches and control how suggestions are sorted.
suggestions	Documents that contain a match for the partial search string in the field designated by the suggester. Amazon CloudSearch suggestions include the document IDs and field values for each matching document. To be a match, the string must match the contents of the field starting from the beginning of the field.
supported AMI	An Amazon Machine Image (p. 131) similar to a paid AMI (p. 149), except that the owner charges for additional software or a service that customers use with their own AMIs.

synchronous bounce — A type of bounce (p. 134) that occurs while the email servers of the sender (p. 156) and receiver (p. 153) are actively communicating.

synonym — A word that is the same or nearly the same as an indexed word and that should produce the same results when specified in a search request. For example, a search for "Rocky Four" or "Rocky 4" should return the fourth *Rocky* movie. This can be done by designating that `four` and `4` are synonyms for `IV`. Synonyms are language-specific.

system Qualifications — The set of Qualifications (p. 152) that represent a worker's (p. 163) history and reputation. The Mechanical Turk system assigns these Qualifications to each worker, and continuously updates the values as they use the system.

T

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

tag — Metadata (consisting of up to 10 key/value pairs) that you can define and assign to Amazon EC2 resources.

tagging — Also called *labeling*. A way to format return path (p. 155) email addresses so that you can specify a different return path for each recipient of a message. Tagging enables you to support VERP (p. 162). For example, if Andrew manages a mailing list, he can use the return paths andrew+recipient1@example.net and andrew+recipient2@example.net so that he can determine which email bounced.

task node — An EC2 instance (p. 139) that runs Hadoop map and reduce tasks, but does not store data. Task nodes are managed by the master node (p. 147), which assigns Hadoop tasks to nodes and monitors their status. While a job flow is running you can increase and decrease the number of task nodes. Because they don't store data and can be added and removed from a job flow, you can use task nodes to manage the EC2 instance capacity your job flow uses, increasing capacity to handle peak loads and decreasing it later.

Task nodes only run a TaskTracker Hadoop daemon.

tebibyte — A contraction of tera binary byte, a tebibyte is 2^40 bytes or 1,099,511,627,776 bytes. A terabyte is 10^12 or 1,000,000,000,000 bytes.

template format version — The version of an AWS CloudFormation template design that determines the available features. If you omit the `AWSTemplateFormatVersion` section from your template, AWS CloudFormation assumes the most recent format version.

template validation — The process of confirming the use of JSON (p. 145) code in an AWS CloudFormation template. You can validate any AWS CloudFormation template using the `cfn-validate-template` command.

throttling — The means by which Amazon SES rejects your attempts to send email because you have exceeded your sending limits (p. 157).

time series data — Data provided as part of a metric. The time value is assumed to be when the value occurred. A metric is the fundamental concept for CloudWatch and represents a time-ordered set of data points. You publish metric data points into CloudWatch and later retrieve statistics about those data points as a time-series ordered data set.

time stamp	A date/time string in ISO 8601 format.
TLS	See Transport Layer Security.
tokenization	The process of splitting a stream of text into separate tokens on detectable boundaries such as whitespace and hyphens.
topic	A communication channel to send messages and subscribe to notifications. It provides an access point for publishers and subscribers to communicate with each other.
Transport Layer Security	A cryptographic protocol that provides security for communication over the Internet. Its predecessor is Secure Sockets Layer (SSL).
trusted signers	AWS accounts that the CloudFront distribution owner has given permission to create signed URLs for a distribution's content.
tuning	Selecting the number and type of AMIs (p. 131) to run a Hadoop job flow most efficiently.
tunnel	A route for transmission of private network traffic that uses the Internet to connect nodes in the private network. The tunnel uses encryption and secure protocols such as PPTP to prevent the traffic from being intercepted as it passes through public routing nodes.

U

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

unbounded	The number of potential occurrences is not limited by a set number. This value is often used when defining a data type that is a list (for example, `maxOccurs="unbounded"`), in Web Services Description Language (p. 163).
unit	Standard measurement for the values submitted to CloudWatch as metric data. Units include Seconds, Percent, Bytes, Bits, Count, Bytes/Second, Bits/Second, Count/Second, and None.
usage report	An AWS report giving details of your usage of a particular AWS service. You can generate and download usage reports from http://aws.amazon.com/usage-reports.
user	A person or application under an account (p. 129) that needs to make API calls to AWS products. Each user has a unique name within the AWS account, and a set of security credentials not shared with other users. These credentials are separate from the AWS account's security credentials. Each user is associated with one and only one AWS account.
user token	A customer credential returned to your product during product activation (p. 130). The user token is a long, encoded string that AWS uses to identify the customer. Your product provides the customer's user token in each request for Amazon S3 the product makes on behalf of the customer. Every user token generated for a particular customer differs from the others because the creation time is one of the items making up the token value.

V

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

validation — See template validation.

value — Instances of attributes (p. 133) for an item, such as cells in a spreadsheet. An attribute might have multiple values.

value-add — The amount you charge each customer on top of the cost of the AWS services they used.

Variable Envelope Return Path — See VERP.

verification — The process of confirming that you own an email address or a domain so that you can send emails from or to it.

VERP — Variable Envelope Return Path. A way in which email sending applications can match bounced emails with the undeliverable address that caused the bounce by using a different return path (p. 155) for each recipient. VERP is typically used for mailing lists. With VERP, the recipient's email address is embedded in the address of the return path, which is where bounced emails are returned. This makes it possible to automate the processing of bounced emails without having to open the bounce messages, which may vary in content.

versioning — Every object in Amazon S3 has a key and a version ID. Objects with the same key, but different version IDs can be stored in the same bucket. Versioning is enabled at the bucket layer using PUT Bucket versioning.

virtual private cloud — See VPC.

virtual private gateway — See VPG.

visibility timeout — The period of time that a message is invisible to the rest of your application after an application component gets it from the queue. During the visibility timeout, the component that received the message usually processes it, and then deletes it from the queue. This prevents multiple components from processing the same message.

VPC — Virtual private cloud. An elastic network populated by infrastructure, platform, and application services that share common security and interconnection.

VPG — Virtual private gateway. The Amazon side of a VPN connection that maintains connectivity. The internal interfaces of the virtual private gateway connect to your VPC via the VPN attachment and the external interfaces connect to the VPN connection, which leads to the customer gateway.

VPN CloudHub — See AWS VPN CloudHub.

VPN connection — Although VPN connection is a general term, we specifically mean the IPsec connection between a VPC (p. 162) and some other network, such as a corporate data center, home network, or co-location facility.

W

Numbers and Symbols (p. 129) | A (p. 129) | B (p. 134) | C (p. 135) | D (p. 137) | E (p. 139) | F (p. 141) | G (p. 142) | H (p. 142) | I (p. 143) | J (p. 144) | K (p. 145) | L (p. 145) | M (p. 146) | N (p. 148) | O (p. 148) | P (p. 149) | Q (p. 152) | R (p. 153) | S (p. 155) | T (p. 160) | U (p. 161) | V (p. 162) | W (p. 163) | X, Y, Z (p. 163)

Web Services Description Language	A language used to describe the actions that a web service can perform, along with the syntax of action requests and responses. Your SOAP or other toolkit interprets a WSDL file to provide your application access to the actions provided by the web service. For most toolkits, your application calls a service action using routines and classes provided or generated by the toolkit.
website owner	A developer who uses Amazon FPS, such as by creating an Amazon Simple Pay button.
worker	A person who performs the tasks specified by a Requester (p. 154) in a Human Intelligence Task (p. 143).

X, Y, Z

No entries