AWS Step Functions

**Developer Guide**



**AWS Step Functions: Developer Guide**

Copyright © 2019 Amazon Web Services, Inc. and/or its aﬃliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be aﬃliated with, connected to, or sponsored by Amazon.

Table of Contents

[What Is AWS Step Functions?](#_bookmark0) [1](#_bookmark0)

[Overview of Step Functions](#_bookmark1) [1](#_bookmark1)

[Supported Regions](#_bookmark2) [1](#_bookmark2)

[About Amazon Web Services](#_bookmark3) [2](#_bookmark3)

[Setting Up](#_bookmark4) [3](#_bookmark4)

[Prerequisites](#_bookmark5) [3](#_bookmark5)

[Create an AWS Account](#_bookmark6) [3](#_bookmark6)

[Create an IAM User](#_bookmark7) [3](#_bookmark7)

[Step 3: Get Your Access Key ID and Secret Access Key](#_bookmark8) [4](#_bookmark8)

[Setting Up Step Functions Local (Downloadable Version)](#_bookmark9) [5](#_bookmark9)

[Step Functions (Downloadable Version) on Your Computer](#_bookmark10) [6](#_bookmark10)

[Step Functions (Downloadable Version) and Docker](#_bookmark11) [7](#_bookmark11)

[Step Functions Local Conﬁguration Options](#_bookmark12) [7](#_bookmark12)

[Step Functions and Lambda Local](#_bookmark14) [8](#_bookmark14)

[Getting Started](#_bookmark21) [12](#_bookmark21)

[Step 1: Creating a State Machine](#_bookmark22) [12](#_bookmark22)

[To create the state machine](#_bookmark23) [12](#_bookmark23)

[Step 2: Starting a New Execution](#_bookmark24) [13](#_bookmark24)

[To start a new execution](#_bookmark25) [13](#_bookmark25)

[Step 3: (Optional) Update a State Machine](#_bookmark26) [14](#_bookmark26)

[To update a state machine](#_bookmark27) [14](#_bookmark27)

[Next Steps](#_bookmark28) [15](#_bookmark28)

[Tutorials](#_bookmark29) [16](#_bookmark29)

[Development Options](#_bookmark30) [16](#_bookmark30)

[Step Functions Console](#_bookmark31) [16](#_bookmark31)

[AWS SDKs](#_bookmark32) [16](#_bookmark32)

[HTTPS Service API](#_bookmark33) [17](#_bookmark33)

[Development Environments](#_bookmark34) [17](#_bookmark34)

[Endpoints](#_bookmark35) [17](#_bookmark35)

[AWS CLI](#_bookmark36) [17](#_bookmark36)

[Run Step Functions Locally](#_bookmark37) [17](#_bookmark37)

[Creating a Lambda State Machine](#_bookmark38) [18](#_bookmark38)

[Step 1: Creating an IAM Role for Lambda](#_bookmark39) [18](#_bookmark39)

[Step 2: Creating a Lambda Function](#_bookmark41) [19](#_bookmark41)

[Step 3: Testing the Lambda Function](#_bookmark43) [19](#_bookmark43)

[Step 4: Creating a State Machine](#_bookmark44) [20](#_bookmark44)

[Step 5: Starting a New Execution](#_bookmark45) [21](#_bookmark45)

[Creating a Lambda State Machine Using AWS CloudFormation](#_bookmark47) [22](#_bookmark47)

[Step 1: Setting Up Your AWS CloudFormation Template](#_bookmark48) [22](#_bookmark48)

[Step 2: Using the AWS CloudFormation Template to Create a Lambda State Machine](#_bookmark49) [26](#_bookmark49)

[Step 3: Starting a State Machine Execution](#_bookmark50) [29](#_bookmark50)

[Creating an Activity State Machine](#_bookmark51) [30](#_bookmark51)

[Step 1: Creating a New Activity](#_bookmark52) [30](#_bookmark52)

[Step 2: Creating a State Machine](#_bookmark54) [30](#_bookmark54)

[Step 3: Implementing a Worker](#_bookmark55) [32](#_bookmark55)

[Step 4: Starting an Execution](#_bookmark56) [33](#_bookmark56)

[Step 5: Running and Stopping the Worker](#_bookmark57) [34](#_bookmark57)

[Handling Error Conditions Using a State Machine](#_bookmark58) [34](#_bookmark58)

[Step 1: Creating an IAM Role for Lambda](#_bookmark59) [35](#_bookmark59)

[Step 2: Creating a Lambda Function That Fails](#_bookmark61) [35](#_bookmark61)

[Step 3: Testing the Lambda Function](#_bookmark63) [36](#_bookmark63)

[Step 4: Creating a State Machine with a Catch Field](#_bookmark64) [36](#_bookmark64)

[Step 5: Starting a New Execution](#_bookmark65) [38](#_bookmark65)

[Periodically Start a State Machine Execution Using CloudWatch Events](#_bookmark66) [39](#_bookmark66)

[Step 1: Creating a State Machine](#_bookmark67) [40](#_bookmark67)

[Step 2: Creating a CloudWatch Events Rule](#_bookmark68) [40](#_bookmark68)

[Starting a State Machine Execution in Response to Amazon S3 Events](#_bookmark69) [42](#_bookmark69)

[Prerequisite: Create a State Machine](#_bookmark70) [42](#_bookmark70)

[Step 1: Create a Bucket in Amazon S3](#_bookmark71) [43](#_bookmark71)

[Step 2: Create a Trail in AWS CloudTrail](#_bookmark72) [43](#_bookmark72)

[Step 3: Create a CloudWatch Events Rule](#_bookmark73) [43](#_bookmark73)

[Step 4: Test the CloudWatch Rule](#_bookmark74) [45](#_bookmark74)

[Example of Execution Input](#_bookmark75) [45](#_bookmark75)

[Creating a Step Functions API Using API Gateway](#_bookmark76) [47](#_bookmark76)

[Step 1: Creating an IAM Role for API Gateway](#_bookmark77) [47](#_bookmark77)

[Step 2: Creating your API Gateway API](#_bookmark79) [48](#_bookmark79)

[Step 3: Testing and Deploying the API Gateway API](#_bookmark80) [50](#_bookmark80)

[Iterating a Loop Using Lambda](#_bookmark81) [51](#_bookmark81)

[Step 1: Create a Lambda Function to Iterate a Count](#_bookmark82) [51](#_bookmark82)

[Step 2: Test the Lambda Function](#_bookmark84) [52](#_bookmark84)

[Step 3: Create a State Machine](#_bookmark85) [53](#_bookmark85)

[Step 4: Start a New Execution](#_bookmark86) [56](#_bookmark86)

[Continue as a New Execution](#_bookmark87) [58](#_bookmark87)

[Prerequisites](#_bookmark88) [59](#_bookmark88)

[Step 1: Create an Iterate Lambda Function to Iterate a Count](#_bookmark89) [59](#_bookmark89)

[Step 2: Create a Restart Lambda Function to Start a New Step Functions Execution](#_bookmark90) [61](#_bookmark90)

[Step 3: Create a State Machine](#_bookmark91) [62](#_bookmark91)

[Step 4: Update the IAM Policy](#_bookmark92) [64](#_bookmark92)

[Step 5: Run an Execution](#_bookmark93) [64](#_bookmark93)

[Using Code Snippets](#_bookmark94) [66](#_bookmark94)

[Step 1: Generate a Code Snippet](#_bookmark95) [66](#_bookmark95)

[Step 2: Update Your State Machine Deﬁnition](#_bookmark96) [68](#_bookmark96)

[Step 3: Start an Execution](#_bookmark97) [70](#_bookmark97)

[How Step Functions Works](#_bookmark98) [71](#_bookmark98)

[States](#_bookmark99) [71](#_bookmark99)

[Tasks](#_bookmark100) [72](#_bookmark100)

[Activities](#_bookmark101) [72](#_bookmark101)

[Service Integrations](#_bookmark107) [80](#_bookmark107)

[Transitions](#_bookmark123) [92](#_bookmark123)

[State Machine Data](#_bookmark124) [92](#_bookmark124)

[Data Format](#_bookmark125) [93](#_bookmark125)

[State Machine Input/Output](#_bookmark126) [93](#_bookmark126)

[State Input/Output](#_bookmark127) [93](#_bookmark127)

[Input and Output Processing](#_bookmark128) [94](#_bookmark128)

[InputPath and Parameters](#_bookmark129) [95](#_bookmark129)

[ResultPath](#_bookmark130) [97](#_bookmark130)

[OutputPath](#_bookmark135) [102](#_bookmark135)

[InputPath, ResultPath and OutputPath Example](#_bookmark136) [102](#_bookmark136)

[Executions](#_bookmark137) [104](#_bookmark137)

[Error Handling](#_bookmark138) [104](#_bookmark138)

[Error Names](#_bookmark139) [105](#_bookmark139)

[Retrying After an Error](#_bookmark140) [105](#_bookmark140)

[Fallback States](#_bookmark141) [107](#_bookmark141)

[Examples Using Retry and Using Catch](#_bookmark142) [108](#_bookmark142)

[Read Consistency](#_bookmark143) [111](#_bookmark143)

[Templates](#_bookmark144) [111](#_bookmark144)

[Tagging](#_bookmark145) [112](#_bookmark145)

[Tagging for Cost Allocation](#_bookmark146) [112](#_bookmark146)

[Viewing and Managing](#_bookmark147) [112](#_bookmark147)

[Tagging API](#_bookmark148) [113](#_bookmark148)

[Sample Projects](#_bookmark149) [114](#_bookmark149)

[Manage a Batch Job (Batch, SNS)](#_bookmark150) [114](#_bookmark150)

[To start a new execution](#_bookmark151) [115](#_bookmark151)

[Example State machine code](#_bookmark152) [115](#_bookmark152)

[IAM Example](#_bookmark153) [116](#_bookmark153)

[Manage a Container Task (ECS, SNS)](#_bookmark154) [117](#_bookmark154)

[To start a new execution](#_bookmark155) [117](#_bookmark155)

[Example State machine code](#_bookmark156) [118](#_bookmark156)

[IAM Example](#_bookmark157) [119](#_bookmark157)

[Transfer Data Records (Lambda, DynamoDB, SQS)](#_bookmark158) [120](#_bookmark158)

[To start a new execution](#_bookmark159) [121](#_bookmark159)

[Example State machine code](#_bookmark160) [121](#_bookmark160)

[IAM Example](#_bookmark161) [122](#_bookmark161)

[Poll for Job Status (Lambda, Batch),](#_bookmark162) [123](#_bookmark162)

[Starting an Execution](#_bookmark163) [124](#_bookmark163)

[Task Timer](#_bookmark164) [126](#_bookmark164)

[Amazon States Language](#_bookmark165) [128](#_bookmark165)

[Example Amazon States Language Speciﬁcation](#_bookmark166) [128](#_bookmark166)

[State Machine Structure](#_bookmark167) [129](#_bookmark167)

[States](#_bookmark168) [130](#_bookmark168)

[Common State Fields](#_bookmark169) [131](#_bookmark169)

[Pass](#_bookmark170) [131](#_bookmark170)

[Task](#_bookmark171) [132](#_bookmark171)

[Choice](#_bookmark175) [135](#_bookmark175)

[Wait](#_bookmark177) [139](#_bookmark177)

[Succeed](#_bookmark178) [140](#_bookmark178)

[Fail](#_bookmark179) [140](#_bookmark179)

[Parallel](#_bookmark180) [140](#_bookmark180)

[Input and Output Processing](#_bookmark182) [143](#_bookmark182)

[Paths](#_bookmark183) [143](#_bookmark183)

[Reference Paths](#_bookmark184) [143](#_bookmark184)

[Errors](#_bookmark185) [146](#_bookmark185)

[Error Representation](#_bookmark186) [146](#_bookmark186)

[Retrying After an Error](#_bookmark187) [146](#_bookmark187)

[Fallback States](#_bookmark188) [148](#_bookmark188)

[Best Practices](#_bookmark189) [150](#_bookmark189)

[Use Timeouts to Avoid Stuck Executions](#_bookmark190) [150](#_bookmark190)

[Use ARNs Instead of Passing Large Payloads](#_bookmark191) [150](#_bookmark191)

[Avoid Reaching the History Limit](#_bookmark192) [150](#_bookmark192)

[Handle Lambda Service Exceptions](#_bookmark193) [151](#_bookmark193)

[Avoid Latency When Polling for Activity Tasks](#_bookmark194) [151](#_bookmark194)

[Limits](#_bookmark195) [152](#_bookmark195)

[General Limits](#_bookmark196) [152](#_bookmark196)

[Limits Related to Accounts](#_bookmark197) [153](#_bookmark197)

[Limits Related to State Machine Executions](#_bookmark198) [153](#_bookmark198)

[Limits Related to Task Executions](#_bookmark199) [153](#_bookmark199)

[Limits Related to API Action Throttling](#_bookmark200) [154](#_bookmark200)

[Limits Related to State Throttling](#_bookmark201) [155](#_bookmark201)

[Restrictions Related to Tagging](#_bookmark202) [155](#_bookmark202)

[Requesting a Limit Increase](#_bookmark203) [156](#_bookmark203)

[Monitoring and Logging](#_bookmark204) [157](#_bookmark204)

[Monitoring Step Functions Using CloudWatch](#_bookmark205) [157](#_bookmark205)

[Metrics that Report a Time Interval](#_bookmark206) [158](#_bookmark206)

[Metrics that Report a Count](#_bookmark207) [158](#_bookmark207)

[State Machine Metrics](#_bookmark208) [158](#_bookmark208)

[Viewing Metrics for Step Functions](#_bookmark209) [161](#_bookmark209)

[Setting Alarms for Step Functions](#_bookmark210) [162](#_bookmark210)

[Logging Step Functions using AWS CloudTrail](#_bookmark211) [164](#_bookmark211)

[Step Functions Information in CloudTrail](#_bookmark212) [164](#_bookmark212)

[Example: Step Functions Log File Entries](#_bookmark213) [165](#_bookmark213)

[Security](#_bookmark214) [170](#_bookmark214)

[Authentication](#_bookmark215) [170](#_bookmark215)

[Creating IAM Roles for AWS Step Functions](#_bookmark216) [171](#_bookmark216)

[To create a role for Step Functions](#_bookmark217) [171](#_bookmark217)

[Attach an Inline Policy](#_bookmark218) [171](#_bookmark218)

[Creating Granular IAM Permissions for Non-Admin Users](#_bookmark219) [172](#_bookmark219)

[Service-Level Permissions](#_bookmark220) [172](#_bookmark220)

[State Machine-Level Permissions](#_bookmark221) [173](#_bookmark221)

[Execution-Level Permissions](#_bookmark222) [173](#_bookmark222)

[Activity-Level Permissions](#_bookmark223) [173](#_bookmark223)

[IAM Policies for Integrated Services](#_bookmark224) [174](#_bookmark224)

[Dynamic vs. Static Resources](#_bookmark225) [174](#_bookmark225)

[Synchronous vs. Asynchronous IAM Policies](#_bookmark226) [174](#_bookmark226)

[Lambda](#_bookmark227) [175](#_bookmark227)

[AWS Batch](#_bookmark228) [175](#_bookmark228)

[DynamoDB](#_bookmark229) [176](#_bookmark229)

[Amazon ECS/Fargate](#_bookmark230) [177](#_bookmark230)

[Amazon SNS](#_bookmark231) [179](#_bookmark231)

[Amazon SQS](#_bookmark232) [180](#_bookmark232)

[AWS Glue](#_bookmark233) [180](#_bookmark233)

[Amazon SageMaker](#_bookmark234) [181](#_bookmark234)

[Activities](#_bookmark237) [188](#_bookmark237)

[Related Step Functions Resources](#_bookmark238) [189](#_bookmark238)

[Document History](#_bookmark239) [190](#_bookmark239)

[AWS Glossary](#_bookmark240) [198](#_bookmark240)

Overview of Step Functions

# What Is AWS Step Functions?

AWS Step Functions is a web service that enables you to coordinate the components of distributed applications and microservices using visual workﬂows. You build applications from individual components that each perform a discrete function, or *task*, allowing you to scale and change applications quickly. Step Functions provides a reliable way to coordinate components and step through the functions of your application. Step Functions provides a graphical console to visualize the components of your application as a series of steps. It automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected, every time. Step Functions logs the state of each step, so when things do go wrong, you can diagnose and debug problems quickly.

Step Functions manages the operations and underlying infrastructure for you to ensure your application is available at any scale.

You can run your tasks on the AWS Cloud, on your own servers, or on any system that has access to AWS. Step Functions can be accessed and used with the Step Functions console, the AWS SDKs, or an HTTP API. This guide shows you how to develop, test, and troubleshoot your own state machine using these methods.

## Overview of Step Functions

Here are some of the key features of AWS Step Functions:

* Step Functions is based on the concepts of [tasks (p. 72)](#_bookmark100) and [state machines (p. 71)](#_bookmark99).
* You deﬁne state machines using the JSON-based [Amazon States Language (p. 128)](#_bookmark165).
* The Step Functions console displays a graphical view of your state machine's structure, which provides you with a way to visually check your state machine's logic and monitor executions.

## Supported Regions

Currently, Step Functions is supported only in the following regions:

* US East (Ohio)
* US East (N. Virginia)
* US West (Oregon)
* US West (N. California)
* Asia Paciﬁc (Mumbai)
* Asia Paciﬁc (Sydney)
* Asia Paciﬁc (Tokyo)
* Asia Paciﬁc (Seoul)
* Asia Paciﬁc (Singapore)
* EU (Frankfurt)
* EU (Ireland)
* EU (London)
* EU (Paris)
* EU (Stockholm)
* Canada (Central)
* South America (São Paulo)
* AWS GovCloud (US-West)

### About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model: you are charged only for the services that you—or your applications—use. For new AWS users, a free usage tier is available. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier, see [Use the AWS Free Tier](https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-free-tier.html). To obtain an AWS account, visit the [AWS home page](https://aws.amazon.com/) and choose **Create a Free Account**.

# Setting Up Step Functions

Topics

* + [Prerequisites For Setting Up Step Functions (p. 3)](#_bookmark5)
  + [Setting Up Step Functions Local (Downloadable Version) (p. 5)](#_bookmark9)

Prerequisites For Setting Up Step Functions

Before you can get started using AWS Step Functions, you must create these AWS resources:

### Create an AWS Account

To access any AWS service, you ﬁrst need to create an [AWS account](https://aws.amazon.com/), an Amazon.com account that can use AWS products. You can use your AWS account to view your activity and usage reports and to manage authentication and access.

To avoid using your AWS account root user for Step Functions actions, it is a best practice to create an IAM user for each person who needs administrative access to Step Functions.

To set up a new account

1. Open <https://aws.amazon.com/>, and then choose **Create an AWS Account**.

**Note**

If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a diﬀerent account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

1. Follow the online instructions.

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

### Create an IAM User

###### To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the [*AWS account root user*](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html) to the IAM console at <https://console.aws.amazon.com/iam/>.

###### Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few [account and service management tasks](https://docs.aws.amazon.com/general/latest/gr/aws_tasks-that-require-root.html).

1. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
2. For **User name**, type **Administrator**.
3. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
4. Choose **Next: Permissions**.
5. On the **Set permissions** page, choose **Add user to group**.
6. Choose **Create group**.
7. In the **Create group** dialog box, for **Group name** type **Administrators**.
8. For **Filter policies**, select the check box for **AWS managed - job function**.
9. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
10. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
11. Choose **Next: Tags** to add metadata to the user by attaching tags as key-value pairs.
12. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to speciﬁc AWS resources, go to [Access Management](https://docs.aws.amazon.com/IAM/latest/UserGuide/access.html) and [Example Policies](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_examples.html).

### Step 3: Get Your Access Key ID and Secret Access Key

To use Step Functions actions (for example, using Java or through the AWS Command Line Interface), you need an access key ID and a secret access key.

###### Note

The access key ID and secret access key are speciﬁc to AWS Identity and Access Management. Don't confuse them with credentials for other AWS services, such as Amazon EC2 key pairs.

###### To get the access key ID and secret access key for an IAM user

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 from the AWS Management Console. We recommend that you use IAM access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see [Permissions Required to](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_permissions-required.html) [Access IAM Resources](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_permissions-required.html) in the *IAM User Guide*.

1. Open the [IAM console](https://console.aws.amazon.com/iam/home?&amp;home).
2. In the navigation pane of the console, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Choose the **Security credentials** tab and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials will look something like this:
   * Access key ID: AKIAIOSFODNN7EXAMPLE
   * Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRﬁCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv ﬁle**. Store the keys in a secure location.

Keep the keys conﬁdential in order to protect your AWS account, and never email them. Do not share them 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?](https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html) in the *IAM User Guide*
* [AWS Security Credentials](https://docs.aws.amazon.com/general/latest/gr/aws-security-credentials.html) in *AWS General Reference*

## Setting Up Step Functions Local (Downloadable Version)

The downloadable version of AWS Step Functions is provided as an executable .jar ﬁle, and as a Docker image. The Java application runs on Windows, Linux, macOS X, and other platforms that support Java. In addition to Java, you need to install the AWS Command Line Interface. For information on installing and conﬁguring the AWS CLI, see the [AWS Command Line Interface User Guide](https://docs.aws.amazon.com/cli/latest/userguide/).

###### Warning

The downloadable version of AWS Step Functions is only intended to be used for testing and shouldn't be used to process sensitive information.

Follow these steps to set up and run Step Functions on your computer:

1. Download Step Functions using the following link:

|  |  |
| --- | --- |
| **Download Links** | **Checksum** |
| [.tar.gz](https://s3.amazonaws.com/stepfunctionslocal/StepFunctionsLocal.tar.gz) | [.zip](https://s3.amazonaws.com/stepfunctionslocal/StepFunctionsLocal.zip) | [tar.gz.md5](https://s3.amazonaws.com/stepfunctionslocal/StepFunctionsLocal.tar.gz.md5) | [zip.md5](https://s3.amazonaws.com/stepfunctionslocal/StepFunctionsLocal.zip.md5) |

1. Extract the zip ﬁle.
2. Test the download and view version information.

$ java -jar StepFunctionsLocal.jar -v Step Function Local

Version: 1.0.0

Build: 2019-01-21

1. (Optional) View a listing of available commands:

$ java -jar StepFunctionsLocal.jar -h

1. To start Step Functions on your computer, open a command prompt window, navigate to the directory where you extracted StepFunctionsLocal.jar and type the following command:

java -jar StepFunctionsLocal.jar

1. To access Step Functions running locally, use the --endpoint-url parameter. For example, using the AWS Command Line Interface, you would specify Step Functions commands as:

aws stepfunctions --endpoint http://localhost:8083 *command*

###### Note

By default Step Functions Local uses a fake account and credentials, and the region is set to US East (N. Virginia). To use Step Functions Local with AWS Lambda, or other supported services, you must conﬁgure your credentials and region.

To conﬁgure and run Step Functions Local to work with AWS Lambda, Lambda Local, or other supported services, see the following topics.

Topics

* + [Step Functions (Downloadable Version) on Your Computer (p. 6)](#_bookmark10)
  + [Step Functions (Downloadable Version) and Docker (p. 7)](#_bookmark11)
  + [Step Functions Local Conﬁguration Options (p. 7)](#_bookmark12)
  + [Step Functions and Lambda Local (p. 8)](#_bookmark14)

### Step Functions (Downloadable Version) on Your Computer

#### Run a HeloWorld State Machine Locally

Once you have run Step Functions locally with the AWS CLI, you can start a state machine execution.

1. Create a state machine from the AWS CLI by escaping the state machine deﬁnition.

aws stepfunctions --endpoint http://localhost:8083 create-state-machine --definition "{\

\"Comment\": \"A Hello World example of the Amazon States Language using a Pass state

\",\

\"StartAt\": \"HelloWorld\",\

\"States\": {\

\"HelloWorld\": {\

\"Type\": \"Pass\",\

\"End\": true\

}\

}}" --name "HelloWorld" --role-arn "arn:aws:iam::012345678901:role/DummyRole"

###### Note

The role-arn is not used for Step Functions Local, but you must have it included with the proper syntax. You can use the ARN from the above example.

If you successfully create the state machine, Step Functions will respond with the creation date and the state machine ARN:

{

"creationDate": 1548454198.202,

"stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"

}

1. Start an execution using the ARN of the state machine you created.

aws stepfunctions --endpoint http://localhost:8083 start-execution --state-machine-arn arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld

#### Step Functions Local with Lambda Local

You can use the local version of Step Functions along with a local version of AWS Lambda. To conﬁgure this, you must install and conﬁgure the AWS Serverless Application Model (AWS SAM).

For information on conﬁguring and running AWS SAM, see:

* [Set Up AWS SAM](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-quick-start.html)
* [Start Lambda Local](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-cli-command-reference-sam-local-start-lambda.html)

Once Lambda is running on your local system, you can start Step Functions Local. From the directory where you extracted your Step Functions local jar ﬁles, start Step Functions Local, conﬁguring the local Lambda endpoint:

java -jar StepFunctionsLocal.jar --lambda-endpoint http://127.0.0.1:3001 *command*

For more information is running Step Functions Local with AWS Lambda, see [Step Functions and Lambda](#_bookmark14) [Local (p. 8)](#_bookmark14)

### Step Functions (Downloadable Version) and Docker

The Step Functions Local Docker image enables you to get started with Step Functions Local quickly by using a docker image with all the needed dependencies. The Docker image enables you to include Step Functions local in your containerized builds, and as part of your continuous integration testing.

To get the Docker image for Step Functions Local, visit [https://hub.docker.com/r/amazon/aws-](https://hub.docker.com/r/amazon/aws-stepfunctions-local) [stepfunctions-local](https://hub.docker.com/r/amazon/aws-stepfunctions-local), or type the Docker pull command:

docker pull amazon/aws-stepfunctions-local

To start the downloadable version of Step Functions on Docker, run:

docker run -p 8083:8083 amazon/aws-stepfunctions-local

In order to interact with AWS Lambda or other supported services you need to conﬁgure your credentials and other conﬁguration options ﬁrst. See:

* [Step Functions Local Conﬁguration Options (p. 7)](#_bookmark12)
* [Credentials and Conﬁguration for Docker (p. 8)](#_bookmark13)

### Step Functions Local Conﬁguration Options

To use Step Functions Local by starting the jar ﬁle, you can set conﬁguration options by either setting them with the AWS CLI, or by including them in the system environment. For Docker, you must specify these options in a ﬁle that you reference when starting Step Functions Local.

###### Conﬁguration Options

|  |  |  |
| --- | --- | --- |
| **Option** | **Command Line** | **Environment** |
| Account | -account,--aws-account | AWS\_ACCOUNT\_ID |
| Region | -region,--aws-region | AWS\_DEFAULT\_REGION |
| Wait Time Scale | -waitTimeScale,--wait-time- scale | WAIT\_TIME\_SCALE |
| Lambda Endpoint | -lambdaEndpoint,--lambda- endpoint | LAMBDA\_ENDPOINT |
| Batch Endpoint | -batchEndpoint,--batch- endpoint | BATCH\_ENDPOINT |
| DynamoDB Endpoint | -dynamoDBEndpoint,-- dynamodb-endpoint | DYNAMODB\_ENDPOINT |

|  |  |  |
| --- | --- | --- |
| **Option** | **Command Line** | **Environment** |
| ECS Endpoint | -ecsEndpoint,--ecs-endpoint | ECS\_ENDPOINT |
| Glue Endpoint | -glueEndpoint,--glue-endpoint | GLUE\_ENDPOINT |
| SageMaker Endpoint | -sageMakerEndpoint,-- sagemaker-endpoint | SAGE\_MAKER\_ENDPOINT |
| SQS Endpoint | -sqsEndpoint,--sqs-endpoint | SQS\_ENDPOINT |
| SNS Endpoint | -snsEndpoint,--sns-endpoint | SNS\_ENDPOINT |

Credentials and Conﬁguration for Docker

To conﬁgure Step Functions Local for Docker, create a ﬁle: aws-stepfunctions-local- credentials.txt.

This ﬁle contains your credentials and other conﬁguration options, such as:

AWS\_DEFAULT\_REGION*=AWS\_REGION\_OF\_YOUR\_AWS\_RESOURCES* AWS\_ACCESS\_KEY\_ID=*YOUR\_AWS\_ACCESS\_KEY* AWS\_SECRET\_ACCESS\_KEY=*YOUR\_AWS\_SECRET\_KEY* WAIT\_TIME\_SCALE=*VALUE*

LAMBDA\_ENDPOINT=*VALUE* BATCH\_ENDPOINT=*VALUE* DYNAMODB\_ENDPOINT=*VALUE* ECS\_ENDPOINT=*VALUE* GLUE\_ENDPOINT=*VALUE* SAGE\_MAKER\_ENDPOINT=*VALUE* SQS\_ENDPOINT=*VALUE* SNS\_ENDPOINT=*VALUE*

Once you have conﬁgured your credentials and conﬁguration options in aws-stepfunctions-local- credentials.txt, start Step Functions with the following command:

docker run -p 8083:8083 --env-file aws-stepfunctions-local-credentials.txt amazon/aws- stepfunctions-local

### Step Functions and Lambda Local

With both Step Functions and Lambda running on your local machine, you can test your state machine and Lambda functions without deploying your code to AWS.

For more information, see:

* [Setting Up Step Functions Local (Downloadable Version) (p. 5)](#_bookmark9)
* [Set Up AWS SAM](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-quick-start.html)

Topics

* + [Step 1: Set Up The AWS Serverless Application Model (p. 9)](#_bookmark15)
  + [Step 2: Test Lambda Local (p. 9)](#_bookmark16)
  + [Step 3: Start Lambda Local (p. 9)](#_bookmark17)
  + [Step 4: Start Step Functions Local (p. 10)](#_bookmark18)
  + [Step 5: Create a State Machine That References Your Lambda Local Function (p. 10)](#_bookmark19)
  + [Step 6: Start an Execution of Your Local State Machine (p. 11)](#_bookmark20)

Step 1: Set Up The AWS Serverless Application Model

Lambda Local requires the AWS Command Line Interface, the AWS Serverless Application Model, and Docker to be installed.

1. Install the AWS SAM CLI.

For more information see: [Installing the AWS SAM CLI](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html).

###### Note

Before installing the AWS SAM CLI, you will need to install the AWS CLI and Docker. See the [Prerequisites](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html) for installing the AWS SAM CLI.

1. Go through the [AWS SAM Quick Start](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-quick-start.html) documentation. Be sure to follow the steps to:
   1. [Initialize the Application](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-quick-start.html#gs-ex1-setup-local-app)
   2. [Test the Application Locally](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-quick-start.html#gs-ex1-test-locally)

This will create a sam-app directory, and will build an environment that includes a Python-bases hello world Lambda function.

#### Step 2: Test Lambda Local

Now that you have installed AWS SAM and created the Hello World Lambda function, test that it works. In the sam-app directory, type:

sam local start-api

This launches a local instance of your Lambda function.

2019-01-31 16:40:27 Found credentials in shared credentials file: ~/.aws/credentials 2019-01-31 16:40:27 Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET] 2019-01-31 16:40:27 You can now browse to the above endpoints to invoke your functions.

You do not need to restart/reload SAM CLI while working on your functions changes will be reflected instantly/automatically. You only need to restart SAM CLI if you update your AWS SAM template

2019-01-31 16:40:27 \* Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)

Open a browser and enter:

http://127.0.0.1:3000/hello

This will show output from your function:

{"message": "hello world", "location": "72.21.198.66"}

Enter **CTRL+C** to end the Lambda API.

#### Step 3: Start Lambda Local

Now that you've tested that the function works, start Lambda Local. In the sam-app directory, type:

sam local start-lambda

This will start Lambda Local and will provide the endpoint to use:

2019-01-29 15:33:32 Found credentials in shared credentials file: ~/.aws/credentials 2019-01-29 15:33:32 Starting the Local Lambda Service. You can now invoke your Lambda

Functions defined in your template through the endpoint.

2019-01-29 15:33:32 \* Running on http://127.0.0.1:3001/ (Press CTRL+C to quit)

#### Step 4: Start Step Functions Local

##### Jar File

If using the .jar ﬁle version of Step Functions Local, start step functions specifying the Lambda endpoint. In the directory where you extracted the .jar ﬁles, type:

java -jar StepFunctionsLocal.jar --lambda-endpoint http://localhost:3001

When Step Functions Local starts, it will check the environment, and then the credentials conﬁgured in your ~/.aws/credentials ﬁle. By default, it will start using a fake user ID, and will be listed as region us-east-1:

2019-01-29 15:38:06.324: Failed to load credentials from environment because Unable to load AWS credentials from environment variables (AWS\_ACCESS\_KEY\_ID (or AWS\_ACCESS\_KEY) and AWS\_SECRET\_KEY (or AWS\_SECRET\_ACCESS\_KEY))

2019-01-29 15:38:06.326: Loaded credentials from profile: default

2019-01-29 15:38:06.326: Starting server on port 8083 with account 123456789012, region us- east-1

##### Docker

If using the Docker version of Step Functions Local, launch Step Functions with the following command.

docker run -p 8083:8083 amazon/aws-stepfunctions-local

For information on installing the Docker version of Step Functions, see [Step Functions (Downloadable](#_bookmark11) [Version) and Docker (p. 7)](#_bookmark11).

###### Note

You can specify the endpoint through the command line or by setting environment variables if you launch Step Functions from the .jar ﬁle. For the Docker version, you must specify the endpoints and credentials in a text ﬁle. See, [Step Functions Local Conﬁguration](#_bookmark12)

[Options (p. 7)](#_bookmark12).

#### Step 5: Create a State Machine That References Your Lambda Local Function

Once Step Functions Local is running, create a state machine that references the HelloWorldFunction

that you initialized in [Step 1: Set Up The AWS Serverless Application Model (p. 9)](#_bookmark15).

aws stepfunctions --endpoint http://localhost:8083 create-state-machine --definition "{\

\"Comment\": \"A Hello World example of the Amazon States Language using an AWS Lambda Local function\",\

\"StartAt\": \"HelloWorld\",\

\"States\": {\

\"HelloWorld\": {\

\"Type\": \"Task\",\

\"Resource\": \"arn:aws:lambda:us-east-1:123456789012:function:HelloWorldFunction\",\

\"End\": true\

}\

}\

}\

}}" --name "HelloWorld" --role-arn "arn:aws:iam::012345678901:role/DummyRole"

This will create a state machine and provide an ARN that you can use to start an execution:

{

"creationDate": 1548805711.403,

"stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"

}

#### Step 6: Start an Execution of Your Local State Machine

Once you have created a state machine, start an execution referencing the endpoint and state machine ARN.

aws stepfunctions --endpoint http://localhost:8083 start-execution --state-machine arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld --name test

This will start an execution of your HelloWorld state machine and give it the name test.

{

"startDate": 1548810641.52,

"executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test"

}

Now that Step Functions is running locally, you can interact with it using the AWS CLI. For instance, to get information about this execution:

aws stepfunctions --endpoint http://localhost:8083 describe-execution --execution-arn arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test

Calling describe-execution for an execution provides more complete details. For example:

{

"status": "SUCCEEDED", "startDate": 1549056334.073,

"name": "test",

"executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test", "stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld", "stopDate": 1549056351.276,

"output": "{\"statusCode\": 200, \"body\": \"{\\\"message\\\": \\\"hello world\\\", \\

\"location\\\": \\\"72.21.198.64\\\"}\"}", "input": "{}"

}

# Getting Started

This tutorial introduces you to the basics of working with AWS Step Functions. You'll create a simple, independently running state machine using a Pass state. The Pass state represents a *no-op* (an instruction with no operation).

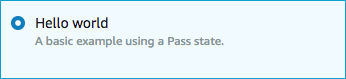
Topics

* + [Step 1: Creating a State Machine (p. 12)](#_bookmark22)
  + [Step 2: Starting a New Execution (p. 13)](#_bookmark24)
  + [Step 3: (Optional) Update a State Machine (p. 14)](#_bookmark26)
  + [Next Steps (p. 15)](#_bookmark28)

Step 1: Creating a State Machine

Step Functions oﬀers various predeﬁned state machines as *templates*. Create your ﬁrst state machine using the **Hello World** template.

### To create the state machine

1. Sign in to the [Step Functions console](https://console.aws.amazon.com/states/home), and then choose **Get Started**.
2. On the **Deﬁne state machine** page, review the **State machine deﬁnition** and the visual workﬂow.

Step Functions ﬁlls in the name of the state machine automatically. It also populates the **Code** pane with the Amazon States Language description of the state machine.

{

"Comment": "A Hello World example of the Amazon States Language using a Pass state", "StartAt": "HelloWorld",

"States": { "HelloWorld": {

"Type": "Pass",

"Result": "Hello World!", "End": true

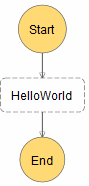
}

}

}

This JSON text deﬁnes a Pass state named HelloWorld. For more information, see [State Machine](#_bookmark167) [Structure (p. 129)](#_bookmark167).

1. Use the graph in the **Visual Workﬂow** pane to check that your Amazon States Language code describes your state machine correctly.



If you don't see the graph, choose in the **Visual Workﬂow** pane.

1. Choose **Next**.
2. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Create state machine**.

## Step 2: Starting a New Execution

After you create your state machine, you can start an execution.

### To start a new execution

1. On the ***Helloworld*** page, choose **New execution**, or **Start execution** if you have started an execution before.

The **New execution** window is displayed.

1. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

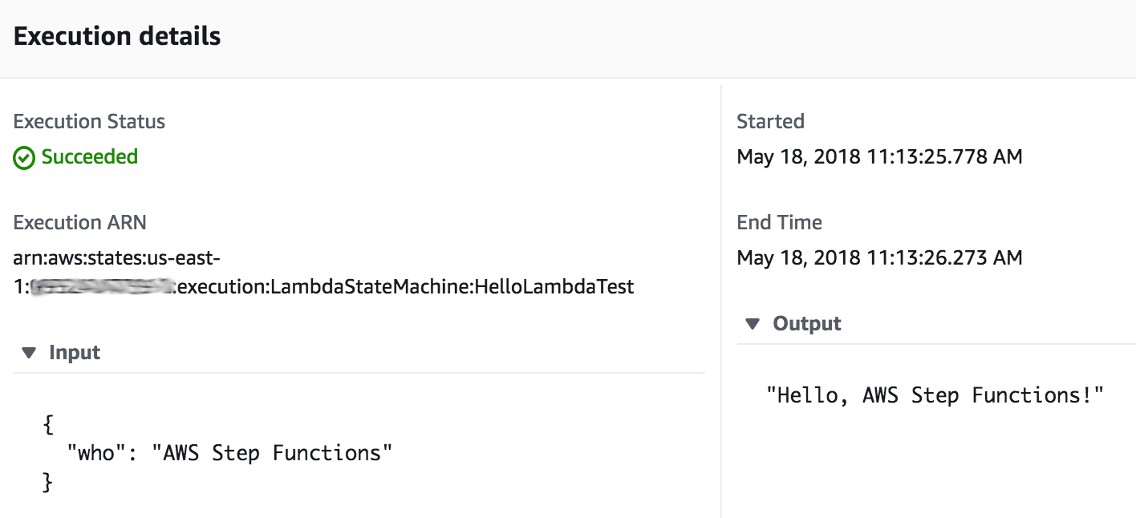
###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Choose **Start execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. (Optional) In the **Execution Details** section, choose the **Info** tab to view the **Execution Status** and the **Started** and **Closed** timestamps.
2. To view the results of your execution, choose the **Output** tab.



## Step 3: (Optional) Update a State Machine

You can update your state machine for future executions.

###### Note

State machine updates in Step Functions are *eventually consistent*. All executions within a few seconds will use the updated deﬁnition and roleArn. Executions started immediately after updating a state machine may use the previous state machine deﬁnition and roleArn.

### To update a state machine

1. On the ***Helloworld*** page, choose **Edit**.

The **Edit** page is displayed.

1. In the **Code** pane, edit the Amazon States Language description of the state machine. Update the

Result to read Hello World has been updated!

{

"Comment": "A Hello World example of the Amazon States Language using a Pass state", "StartAt": "HelloWorld",

"States": { "HelloWorld": {

"Type": "Pass",

"Result": "Hello World *has been updated*!", "End": true

}

}

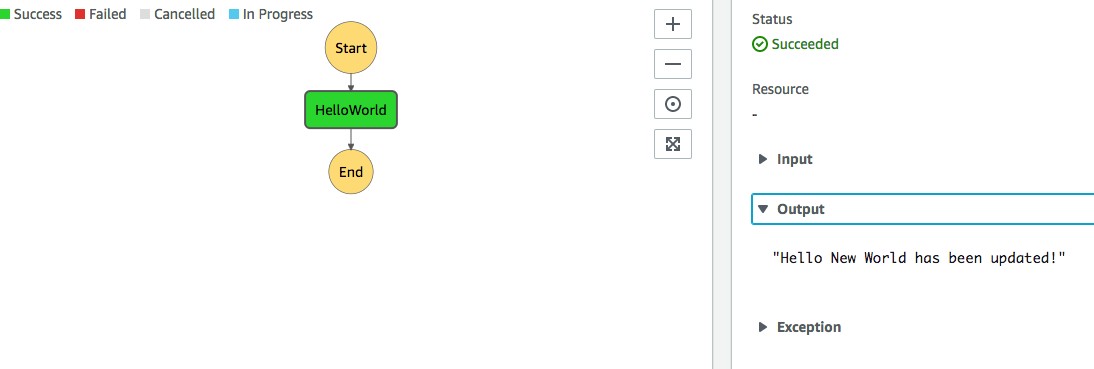
}

1. (Optional) Select a new IAM role from the **IAM role for executions** list.

###### Note

You can also select **Create new role** to create a new IAM role. For more information, see [Creating IAM Roles for AWS Step Functions (p. 171)](#_bookmark216).

1. Choose **Save** and then **Start execution**.
2. On the **New execution** page choose **Start Execution**.
3. To view the results of your execution, select the **HelloWorld** state in the **Visual workﬂow** and expand the **Output** section under **Step details**.



###### Note

The output text matches your newly updated state machine.

## Next Steps

Now that you've created a simple state machine using a Pass state, try the following:

* [Create a Lambda state machine (p. 18)](#_bookmark38)
* [Create a Lambda state machine using AWS CloudFormation (p. 22)](#_bookmark47)
* [Create an activity state machine (p. 30)](#_bookmark51)
* [Handle error conditions using a state machine (p. 34)](#_bookmark58)
* [Start a state machine using Amazon CloudWatch Events (p. 39)](#_bookmark66)
* [Create a Step Functions API using Amazon API Gateway (p. 47)](#_bookmark76)

# Tutorials

The following tutorials will help you get started working with AWS Step Functions. To complete these tutorials, you'll need an AWS account. If you don't have an AWS account, navigate to [http://](https://aws.amazon.com/) [aws.amazon.com/](https://aws.amazon.com/) and choose **Sign In to the Console**.

Topics

* + [Development Options (p. 16)](#_bookmark30)
  + [Creating a Lambda State Machine (p. 18)](#_bookmark38)
  + [Creating a Lambda State Machine Using AWS CloudFormation (p. 22)](#_bookmark47)
  + [Creating an Activity State Machine (p. 30)](#_bookmark51)
  + [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58)
  + [Periodically Start a State Machine Execution Using CloudWatch Events (p. 39)](#_bookmark66)
  + [Starting a State Machine Execution in Response to Amazon S3 Events (p. 42)](#_bookmark69)
  + [Creating a Step Functions API Using API Gateway (p. 47)](#_bookmark76)
  + [Iterating a Loop Using Lambda (p. 51)](#_bookmark81)
  + [Continue as a New Execution (p. 58)](#_bookmark87)
  + [Using Code Snippets Create a State to Send an Amazon SNS message (p. 66)](#_bookmark94)

Development Options

You can implement your Step Functions state machines in a number of ways.

### Step Functions Console

You can deﬁne a state machine using the Step Functions console. You can write complex state machines in the cloud without using a local development enviroment by taking advantage of Lambda to supply code for your tasks and the Step Functions console to deﬁne your state machine using Amazon States Language.

The [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial uses this technique to create a simple state machine, execute it, and view its results.

### AWS SDKs

Step Functions is supported by SDKs for Java, .NET, Ruby, PHP, Python (boto 3), JavaScript, Go, and C

++, providing a convenient way to use the Step Functions HTTPS API actions in various programming languages.

You can develop state machines, activities, or state machine starters using the API actions exposed by these libraries. You can also access visibility operations using these libraries to develop your own Step Functions monitoring and reporting tools.

To use Step Functions with other AWS services, see the reference documentation for the current AWS SDKs and [Tools for Amazon Web Services](http://aws.amazon.com/tools/).

###### Note

Step Functions supports only an HTTPS endpoint.

### HTTPS Service API

Step Functions provides service operations accessible through HTTPS requests. You can use these operations to communicate directly with Step Functions and to develop your own libraries in any language that can communicate with Step Functions through HTTPS.

You can develop state machines, workers, or state machine starters using the service API actions. You can also access visibility operations through the API actions to develop your own monitoring and reporting tools. For detailed information on API actions, see the [*AWS Step Functions API Reference*](https://docs.aws.amazon.com/step-functions/latest/apireference/).

### Development Environments

You must set up a development environment appropriate to the programming language that you plan to use. For example, if you intend to develop for Step Functions with Java, you should install a Java development environment (such as the SDK for Java) on each of your development workstations. If you use Eclipse IDE for Java Development, you should also install the Toolkit for Eclipse. This Eclipse plug-in adds features useful for AWS development.

If your programming language requires a run-time environment, you must set up the environment on each computer where these processes run.

### Endpoints

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

Each endpoint in Step Functions is completely independent: A state machine or activity exists only within the region where it was created. Any state machines and activities that you create in one region don't share any data or attributes with those created in another region. For example, you can register a state machine named STATES-Flows-1 in two diﬀerent regions, but the two state machines won't share data or attributes with each other, being completely independent from each other.

For a list of Step Functions endpoints, see [Regions and Endpoints: AWS Step Functions](https://docs.aws.amazon.com/general/latest/gr/rande.html#step-functions_region) in the *Amazon Web Services General Reference*.

### AWS CLI

You can access many Step Functions features from the AWS CLI. The AWS CLI provides an alternative to using the Step Functions console or, in some cases, to program using the AWS Step Functions API actions. For example, you can use the AWS CLI to create a new state machine and then list your state machines.

The Step Functions commands in AWS CLI allow you to start and manage executions, poll for activities, record task heartbeats, and so on. For a complete list of Step Functions commands and the descriptions of the available arguments and examples showing their use, see the *AWS CLI Command Reference*.

The AWS CLI commands follow the Amazon States Language closely, so you can use the AWS CLI to learn about the Step Functions API actions. You can also use your existing API knowledge to prototype code or perform Step Functions actions from the command line.

### Run Step Functions Locally

For testing and development purposes, you can install and run Step Functions on your local machine. With a local version of Step Functions you can start an execution on any machine. The local version of

Step Functions can invoke Lambda functions, both in AWS and running locally. You can also coordinate other [supported AWS services (p. 80)](#_bookmark107). For more information, see [Setting Up Step Functions Local](#_bookmark9) [(Downloadable Version) (p. 5)](#_bookmark9).

## Creating a Lambda State Machine

In this tutorial you'll create an AWS Step Functions state machine that uses a AWS Lambda function to implement a Task state. A Task state performs a single unit of work.

Lambda is well-suited for implementing Task states, because Lambda functions are *stateless* (they have a predictable input-output relationship), easy to write, and don't require deploying code to a server instance. You can write code in the AWS Management Console or your favorite editor, and AWS handles the details of providing a computing environment for your function and running it.

Topics

* + [Step 1: Creating an IAM Role for Lambda (p. 18)](#_bookmark39)
  + [Step 2: Creating a Lambda Function (p. 19)](#_bookmark41)
  + [Step 3: Testing the Lambda Function (p. 19)](#_bookmark43)
  + [Step 4: Creating a State Machine (p. 20)](#_bookmark44)
  + [Step 5: Starting a New Execution (p. 21)](#_bookmark45)

### Step 1: Creating an IAM Role for Lambda

Both Lambda and Step Functions can execute code and access AWS resources (for example, data stored in Amazon S3 buckets). To maintain security, you must grant Lambda and Step Functions access to these resources.

Lambda requires you to assign an IAM role when you create a Lambda function in the same way Step Functions requires you to assign an IAM role when you create a state machine.

#### To create a role for Lambda

You can use the IAM console to create a service-linked role.

###### To create a role (console)

1. Sign in to the AWS Management Console and open the IAM console at [https:// console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).
2. In the navigation pane of the IAM console, choose **Roles**. Then choose **Create role**.
3. Choose the **AWS Service** role type, and then choose **Lambda**.
4. Choose the **Lambda** use case. Use cases are deﬁned by the service to include the trust policy required by the service. Then choose **Next: Permissions**.
5. Choose one or more permissions policies to attach to the role (for instance,

AWSLambdaBasicExecutionRole). See [AWS Lambda Permissions Model](https://docs.aws.amazon.com/lambda/latest/dg/intro-permission-model.html).

Select the box next to the policy that assigns the permissions that you want the role to have, and then choose **Next: Review**.

1. Enter a **Role name**.
2. (Optional) For **Role description**, edit the description for the new service-linked role.
3. Review the role and then choose **Create role**.

### Step 2: Creating a Lambda Function

Your Lambda function receives input (a name) and returns a greeting that includes the input value.

#### To create the Lambda function

###### Important

Ensure that your Lambda function is under the same AWS account and region as your state machine.

1. Log in to the [Lambda console](https://console.aws.amazon.com/lambda/home) and choose **Create a function**.
2. In the **Blueprints** section, choose **Author from scratch**.
3. In the **Basic information** section, conﬁgure your Lambda function:
   1. For **Name**, type HelloFunction.
   2. For **Role**, select **Choose an existing role**.
   3. For **Existing role**, select [the Lambda role that you created earlier (p. 18)](#_bookmark40).

###### Note

If the IAM role that you created doesn't appear in the list, the role might still need a few minutes to propagate to Lambda.

* 1. Choose **Create function**.

When your Lambda function is created, note its Amazon Resource Name (ARN) in the upper- right corner of the page. For example:

arn:aws:lambda:us-east-1:123456789012:function:HelloFunction

1. Copy the following code for the Lambda function into the **Function code** section of the

***HelloFunction*** page:

exports.handler = (event, context, callback) => { callback(null, "Hello, " + event.who + "!");

};

This code assembles a greeting using the who ﬁeld of the input data, which is provided by the event object passed into your function. You will add input data for this function later, when you [start a new execution (p. 21)](#_bookmark46). The callback method returns the assembled greeting from your function.

1. Choose **Save**.

### Step 3: Testing the Lambda Function

Test your Lambda function to see it in operation.

#### To test your Lambda function

1. On the **Select a test event** drop-down, choose **Conﬁgure test event** and type HelloFunction for

Event name.

1. Replace the example data with the following:

{

"who": "AWS Step Functions"

}

The "who" entry corresponds to the event.who ﬁeld in your Lambda function, completing the greeting. You will use the same input data when running the function as a Step Functions task.

1. Choose **Create**.
2. On the ***HelloFunction*** page, **Test** your Lambda function using the new data.

The results of the test are displayed at the top of the page. Expand **Details** to see the output.

### Step 4: Creating a State Machine

Use the Step Functions console to create a state machine with a Task state. Add a reference to your Lambda function in the Task state. The Lambda function is invoked when an execution of the state machine reaches the Task state.

#### To create the state machine

1. Log in to the [Step Functions console](https://console.aws.amazon.com/states/home) and choose **Create a state machine**.
2. On the **Deﬁne state machine** page, select **Author with code snippets** and enter a **Name for your state machine**, for example *LambdaStateMachine*.

###### Note

State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:

* + Whitespace
  + Wildcard characters (? \*)
  + Bracket characters (< > { } [ ])
  + Special characters (: ; , \ | ^ ~ $ # % & ` ")
  + Control characters (\\u0000 - \\u001f or \\u007f - \\u009f).

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. In the **State machine deﬁnition** pane, add the following state machine deﬁnition using the ARN of [the Lambda function that you created earlier (p. 19)](#_bookmark42), for example:

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:HelloFunction*", "End": true

}

}

}

This is a description of your state machine using the Amazon States Language. It deﬁnes a single

Task state named HelloWorld. For more information, see [State Machine Structure (p. 129)](#_bookmark167).

###### Note

You can also set up a Retry for Task states. As a best practice, ensure production code can handle Lambda service exceptions (Lambda.ServiceException and Lambda.SdkclientException). For more information see:

* + [Handle Lambda Service Exceptions (p. 151)](#_bookmark193).
  + [Retrying After an Error (p. 105)](#_bookmark140).

Choose **Next**.

1. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Next**.

### Step 5: Starting a New Execution

After you create your state machine, you can start an execution.

#### To start a new execution

1. On the ***LambdaStateMachine*** page, choose **Start execution**. The **New execution** page is displayed.
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. In the execution input area, replace the example data with the following:

{

"who" : "AWS Step Functions"

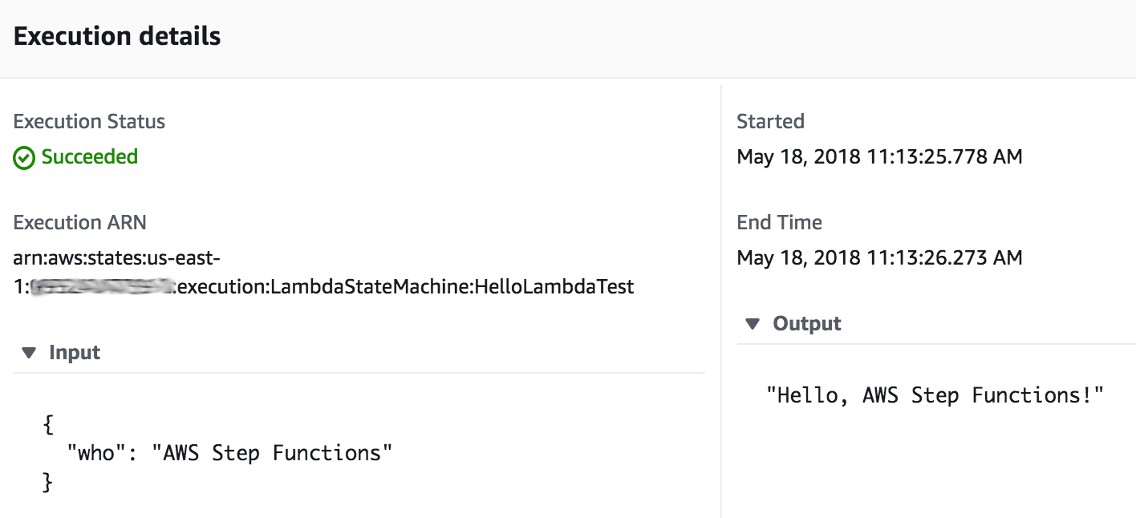
}

"who" is the key name that your Lambda function uses to get the name of the person to greet.

1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. To view the results of your execution, expand the **Output** section under **Execution details**.

Using AWS CloudFormation

## Creating a Lambda State Machine Using AWS CloudFormation

This tutorial shows you how to create a basic AWS Lambda function using AWS CloudFormation. You will use the AWS CloudFormation console and a YAML *template* to create the *stack* (IAM roles, the Lambda function, and the state machine). You will then use the AWS Step Functions console to start the state machine execution. For more information, see [Working with CloudFormation Templates](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-guide.html) and the [AWS::StepFunctions::StateMachine](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-stepfunctions-statemachine.html) resource in the *AWS CloudFormation User Guide*.

Topics

* [Step 1: Setting Up Your AWS CloudFormation Template (p. 22)](#_bookmark48)
* [Step 2: Using the AWS CloudFormation Template to Create a Lambda State Machine (p. 26)](#_bookmark49)
* [Step 3: Starting a State Machine Execution (p. 29)](#_bookmark50)

### Step 1: Setting Up Your AWS CloudFormation Template

Before you use the [example YAML template (p. 26)](#_bookmark49), you should understand its separate parts.

#### To create an IAM role for Lambda

Deﬁne the trust policy associated with the IAM role for the Lambda function. YAML

LambdaExecutionRole: Type: "AWS::IAM::Role"

Properties: AssumeRolePolicyDocument:

Version: "2012-10-17" Statement:

- Effect: Allow Principal:

Service: lambda.amazonaws.com Action: "sts:AssumeRole"

JSON

"LambdaExecutionRole": { "Type": "AWS::IAM::Role",

"Properties": { "AssumeRolePolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Principal": {

"Service": "lambda.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

}

#### To create a Lambda function

Deﬁne the following properties of the Lambda function which prints the message Hello World.

###### Important

Ensure that your Lambda function is under the same AWS account and region as your state machine.

YAML

MyLambdaFunction:

Type: "AWS::Lambda::Function" Properties:

Handler: "index.handler"

Role: !GetAtt [ LambdaExecutionRole, Arn ] Code:

ZipFile: |

exports.handler = (event, context, callback) => { callback(null, "Hello World!");

};

Runtime: "nodejs8.10" Timeout: "25"

JSON

"MyLambdaFunction": {

"Type": "AWS::Lambda::Function", "Properties": {

"Handler": "index.handler", "Role": {

"Fn::GetAtt": [ "LambdaExecutionRole", "Arn"

]

},

"Code": {

"ZipFile": "exports.handler = (event, context, callback) => {\n callback(null, \"Hello World!\");\n};\n"

},

"Runtime": "nodejs8.10",

"Timeout": "25"

}

},

#### To create an IAM role for the state machine execution

Deﬁne the trust policy associated with the IAM role for the state machine execution.

YAML

StatesExecutionRole: Type: "AWS::IAM::Role"

Properties: AssumeRolePolicyDocument:

Version: "2012-10-17" Statement:

- Effect: "Allow" Principal:

Service:

- !Sub states.${AWS::Region}.amazonaws.com Action: "sts:AssumeRole"

Path: "/" Policies:

- PolicyName: StatesExecutionPolicy PolicyDocument:

Version: "2012-10-17" Statement:

- Effect: Allow Action:

- "lambda:InvokeFunction" Resource: "\*"

JSON

"StatesExecutionRole": { "Type": "AWS::IAM::Role",

"Properties": { "AssumeRolePolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Principal": {

"Service": [

{

"Fn::Sub": "states.

${AWS::Region}.amazonaws.com"

}

]

},

"Action": "sts:AssumeRole"

}

]

},

"Path": "/", "Policies": [

{

"PolicyName": "StatesExecutionPolicy", "PolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"lambda:InvokeFunction"

],

"Resource": "\*"

}

]

}

}

]

}

},

#### To create a Lambda state machine

Deﬁne the Lambda state machine. YAML

MyStateMachine:

Type: "AWS::StepFunctions::StateMachine" Properties:

DefinitionString:

!Sub

- |-

{

"Comment": "A Hello World AWL example using an AWS Lambda function", "StartAt": "HelloWorld",

"States": { "HelloWorld": {

"Type": "Task",

"Resource": "${lambdaArn}", "End": true

}

}

}

- {lambdaArn: !GetAtt [ MyLambdaFunction, Arn ]} RoleArn: !GetAtt [ StatesExecutionRole, Arn ]

JSON

}

}

"lambdaArn": {

"Fn::GetAtt": [ "MyLambdaFunction", "Arn"

]

}

}

]

},

"RoleArn": {

"Fn::GetAtt": [ "StatesExecutionRole", "Arn"

]

}

\"HelloWorld

\"Resource\": \"${lambdaArn}\",\n \"End\":

\"Type\": \"Task\",\n

}\n }\n}",

{

\": {\n

true\n

"MyStateMachine": {

"Type": "AWS::StepFunctions::StateMachine", "Properties": {

"DefinitionString": { "Fn::Sub": [

"{\n \"Comment\": \"A Hello World AWL example using an AWS Lambda function\",\n \"StartAt\": \"HelloWorld\",\n \"States\": {\n

### Step 2: Using the AWS CloudFormation Template to Create a Lambda State Machine

After you understand the diﬀerent parts of the AWS CloudFormation template, you can put them together and use the template to create a AWS CloudFormation stack.

#### To create the Lambda state machine

1. Copy the following example data to a ﬁle named MyStateMachine.yaml for the YAML example, or

MyStateMachine.json for JSON. YAML

AWSTemplateFormatVersion: "2010-09-09"

Description: "An example template with an IAM role for a Lambda state machine." Resources:

LambdaExecutionRole: Type: "AWS::IAM::Role"

Properties: AssumeRolePolicyDocument:

Version: "2012-10-17" Statement:

- Effect: Allow Principal:

Service: lambda.amazonaws.com Action: "sts:AssumeRole"

MyLambdaFunction:

Type: "AWS::Lambda::Function" Properties:

Handler: "index.handler"

Role: !GetAtt [ LambdaExecutionRole, Arn ] Code:

ZipFile: |

exports.handler = (event, context, callback) => { callback(null, "Hello World!");

};

Runtime: "nodejs8.10" Timeout: "25"

StatesExecutionRole: Type: "AWS::IAM::Role"

Properties: AssumeRolePolicyDocument:

Version: "2012-10-17" Statement:

- Effect: "Allow" Principal:

Service:

- !Sub states.${AWS::Region}.amazonaws.com Action: "sts:AssumeRole"

Path: "/" Policies:

- PolicyName: StatesExecutionPolicy PolicyDocument:

Version: "2012-10-17" Statement:

- Effect: Allow Action:

- "lambda:InvokeFunction" Resource: "\*"

MyStateMachine:

Type: "AWS::StepFunctions::StateMachine" Properties:

DefinitionString:

!Sub

- |-

{

"Comment": "A Hello World AWL example using an AWS Lambda function", "StartAt": "HelloWorld",

"States": { "HelloWorld": {

"Type": "Task",

"Resource": "${lambdaArn}", "End": true

}

}

}

- {lambdaArn: !GetAtt [ MyLambdaFunction, Arn ]} RoleArn: !GetAtt [ StatesExecutionRole, Arn ]

JSON

{

"AWSTemplateFormatVersion": "2010-09-09",

"Description": "An example template with an IAM role for a Lambda state machine.",

"Resources": { "LambdaExecutionRole": {

"Type": "AWS::IAM::Role",

"Properties": { "AssumeRolePolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Principal": {

"Service": "lambda.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

}

},

"MyLambdaFunction": {

"Type": "AWS::Lambda::Function", "Properties": {

"Handler": "index.handler", "Role": {

"Fn::GetAtt": [ "LambdaExecutionRole", "Arn"

]

},

"Code": {

"ZipFile": "exports.handler = (event, context, callback) => {\n callback(null, \"Hello World!\");\n};\n"

},

"Runtime": "nodejs8.10",

"Timeout": "25"

}

},

"StatesExecutionRole": {

"Type": "AWS::IAM::Role",

"Properties": { "AssumeRolePolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Principal": {

"Service": [

{

${AWS::Region}.amazonaws.com"

]

},

"Fn::Sub": "states.

}

"Action": "sts:AssumeRole"

}

]

},

"Path": "/", "Policies": [

{

"PolicyName": "StatesExecutionPolicy", "PolicyDocument": {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"lambda:InvokeFunction"

],

"Resource": "\*"

}

]

}

}

]

}

},

"MyStateMachine": {

"Type": "AWS::StepFunctions::StateMachine", "Properties": {

"DefinitionString": { "Fn::Sub": [

"{\n \"Comment\": \"A Hello World AWL example using an AWS Lambda function\",\n \"StartAt\": \"HelloWorld\",\n \"States\":

{\n \"HelloWorld\": {\n \"Type\": \"Task\",\n \"Resource\":

\"${lambdaArn}\",\n \"End\": true\n }\n }\n}",

{

"lambdaArn": { "Fn::GetAtt": [

"MyLambdaFunction", "Arn"

]

}

}

]

},

"RoleArn": {

"Fn::GetAtt": [ "StatesExecutionRole", "Arn"

]

}

}

}

}

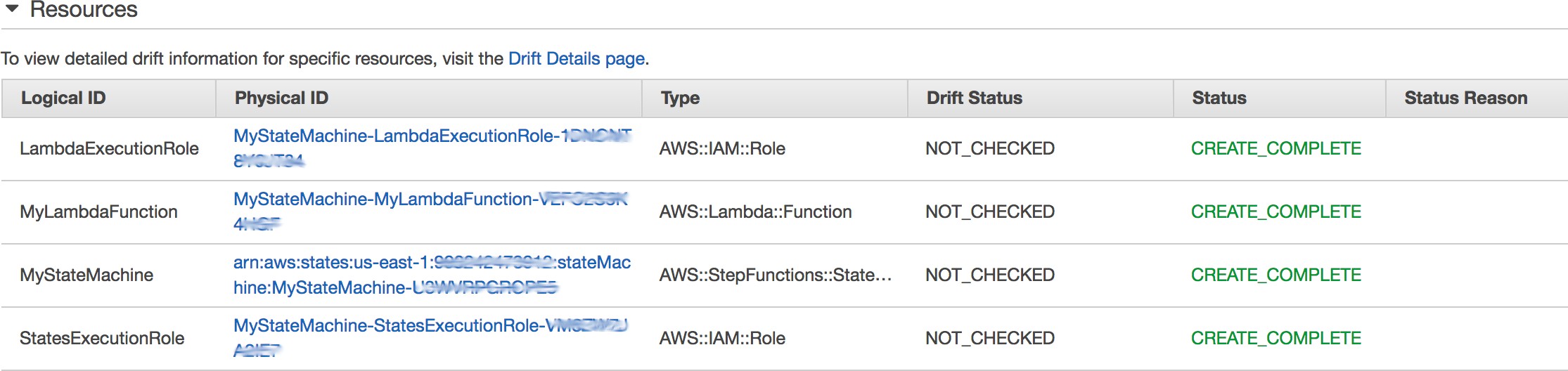
}

1. Log in to the [AWS CloudFormation console](https://console.aws.amazon.com/cloudformation/home) and choose **Create Stack**.
2. On the **Select Template** page, select **Upload a template to Amazon S3**. Choose your

MyStateMachine ﬁle, and then choose **Next**.

1. On the **Specify Details** page, for **Stack name**, type MyStateMachine, and then choose **Next**.
2. On the **Options** page, choose **Next**.
3. On the **Review** page, choose **I acknowledge that AWS CloudFormation might create IAM resources.** and then choose **Create**.

AWS CloudFormation begins to create the MyStateMachine stack and displays the **CREATE\_IN\_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.

1. (Optional) To display the resources in your stack, select the stack and choose the **Resources** tab.

### Step 3: Starting a State Machine Execution

After you create your Lambda state machine, you can start an execution.

#### To start the state machine execution

1. Log in to the [Step Functions console](https://console.aws.amazon.com/states/home) and choose the name of the state machine that you created using AWS CloudFormation.
2. On the ***MyStateMachine-ABCDEFGHIJ1K*** page, choose **New execution**.

The **New execution** page is displayed.

1. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. (Optional) In the **Execution Details** review the **Execution Status** and the **Started** and **Closed**

timestamps.

1. To view the results of your execution, choose **Output**.

## Creating an Activity State Machine

Activities allow you to control worker code that runs somewhere else in your state machine. For an overview, see the [Activities (p. 72)](#_bookmark101) topic in the [How Step Functions Works (p. 71)](#_bookmark98) section. This

tutorial introduces you to creating an activity-based state machine using Java and AWS Step Functions. To complete this tutorial you'll need the following:

* The [SDK for Java](https://aws.amazon.com/sdk-for-java/). The example activity in this tutorial is a Java application that uses the AWS SDK for

Java to communicate with AWS.

* AWS credentials in the environment or in the standard AWS conﬁguration ﬁle. For more information, see [Set up Your AWS credentials](https://docs.aws.amazon.com/AWSSdkDocsJava/latest/DeveloperGuide/set-up-creds.html) in the *AWS SDK for Java Developer Guide*.

Topics

* + [Step 1: Creating a New Activity (p. 30)](#_bookmark52)
  + [Step 2: Creating a State Machine (p. 30)](#_bookmark54)
  + [Step 3: Implementing a Worker (p. 32)](#_bookmark55)
  + [Step 4: Starting an Execution (p. 33)](#_bookmark56)
  + [Step 5: Running and Stopping the Worker (p. 34)](#_bookmark57)

### Step 1: Creating a New Activity

You must make Step Functions aware of the *activity* whose *worker* (a program) you want to create. Step Functions responds with an ARN that establishes an identity for the activity. Use this identity to coordinate the information passed between your state machine and worker.

###### Important

Ensure that your activity task is under the same AWS account as your state machine.

To create the new activity task

1. In the [Step Functions console](https://console.aws.amazon.com/states/home), choose **Activities** in the left navigation panel.
2. Choose **Create activity**.
3. Type an **Activity Name**. For example *get-greeting*, and choose **Create Activity**.
4. When your activity task is created, note its Amazon Resource Name (ARN), for example:

arn:aws:states:us-east-1:123456789012:activity:get-greeting

### Step 2: Creating a State Machine

Create a state machine that will determine when your activity is invoked and when your worker should perform its primary work, collect its results, and return them.

#### To create the state machine

1. In the [Step Functions console](https://console.aws.amazon.com/states/home), choose **State machines** in the left navigation panel.
2. On the **State machines** page, choose **Create state machine**, select **Author with code snippets**, and enter a name under **Details** (for example *ActivityStateMachine*).

###### Note

State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:

* + Whitespace
  + Wildcard characters (? \*)
  + Bracket characters (< > { } [ ])
  + Special characters (: ; , \ | ^ ~ $ # % & ` ")
  + Control characters (\\u0000 - \\u001f or \\u007f - \\u009f).

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

Under **State machine deﬁnition**, enter the following code, and include the ARN of [the activity task](#_bookmark53) [that you created earlier (p. 30)](#_bookmark53) in the Resource ﬁeld, for example:

{

"Comment": "An example using a Task state.", "StartAt": "getGreeting",

"Version": "1.0",

"TimeoutSeconds": 300, "States":

{

"getGreeting": { "Type": "Task",

"Resource": "*arn:aws:states:us-east-1:123456789012:activity:get-greeting*", "End": true

}

}

}

This is a description of your state machine using the Amazon States Language. It deﬁnes a single

Task state named getGreeting. For more information, see [State Machine Structure (p. 129)](#_bookmark167).

1. Use the graph in the **Visual Workﬂow** pane to check that your Amazon States Language code describes your state machine correctly.

If you don't see the graph, choose in the **Visual Workﬂow** pane.

1. Choose **Next**.
2. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Create state machine**.

### Step 3: Implementing a Worker

Create a *worker*, a program which is responsible for the following:

* Polling Step Functions for activities using the GetActivityTask API action.
* Performing the work of the activity using your code, (for example, the getGreeting() method in the code below).
* Returning the results using the SendTaskSuccess, SendTaskFailure, and SendTaskHeartbeat

API actions.

###### Note

For a more complete example of an activity worker, see [Example Activity Worker in](#_bookmark106)

[Ruby (p. 74)](#_bookmark106). This example provides an implementation based on best practices, that can be used as a reference for your activity worker. The code implements a consumer-producer pattern with a conﬁgurable number of threads for pollers and activity workers.

To implement the worker

1. Create a new ﬁle named GreeterActivities.java.
2. Add the following code to it:

import com.amazonaws.ClientConfiguration;

import com.amazonaws.auth.EnvironmentVariableCredentialsProvider; import com.amazonaws.regions.Regions;

import com.amazonaws.services.stepfunctions.AWSStepFunctions;

import com.amazonaws.services.stepfunctions.AWSStepFunctionsClientBuilder; import com.amazonaws.services.stepfunctions.model.GetActivityTaskRequest; import com.amazonaws.services.stepfunctions.model.GetActivityTaskResult; import com.amazonaws.services.stepfunctions.model.SendTaskFailureRequest; import com.amazonaws.services.stepfunctions.model.SendTaskSuccessRequest; import com.amazonaws.util.json.Jackson;

import com.fasterxml.jackson.databind.JsonNode; import java.util.concurrent.TimeUnit;

public class GreeterActivities {

public String getGreeting(String who) throws Exception { return "{\"Hello\": \"" + who + "\"}";

}

public static void main(final String[] args) throws Exception { GreeterActivities greeterActivities = new GreeterActivities(); ClientConfiguration clientConfiguration = new ClientConfiguration(); clientConfiguration.setSocketTimeout((int)TimeUnit.SECONDS.toMillis(70));

AWSStepFunctions client = AWSStepFunctionsClientBuilder.standard()

.withRegion(Regions.US\_EAST\_1)

.withCredentials(new EnvironmentVariableCredentialsProvider())

.withClientConfiguration(clientConfiguration)

.build();

while (true) {

GetActivityTaskResult getActivityTaskResult = client.getActivityTask(

new GetActivityTaskRequest().withActivityArn(ACTIVITY\_ARN));

if (getActivityTaskResult.getTaskToken() != null) { try {

JsonNode json = Jackson.jsonNodeOf(getActivityTaskResult.getInput());

String greetingResult =

greeterActivities.getGreeting(json.get("who").textValue()); client.sendTaskSuccess(

new SendTaskSuccessRequest().withOutput(

greetingResult).withTaskToken(getActivityTaskResult.getTaskToken()));

} catch (Exception e) {

client.sendTaskFailure(new SendTaskFailureRequest().withTaskToken( getActivityTaskResult.getTaskToken()));

}

} else {

Thread.sleep(1000);

}

}

}

}

###### Note

The EnvironmentVariableCredentialsProvider class in this example assumes that the AWS\_ACCESS\_KEY\_ID (or AWS\_ACCESS\_KEY) and AWS\_SECRET\_KEY (or AWS\_SECRET\_ACCESS\_KEY) environment variables are set. For more information about providing the required credentials to the factory, see [AWSCredentialsProvider](https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/AWSCredentialsProvider.html) in the *AWS*

*SDK for Java API Reference* and [Set up AWS Credentials and Region for Development](https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-credentials.html) in the

*AWS SDK for Java Developer Guide*.

To give Step Functions suﬃcient time to process the request, setSocketTimeout is set to 70 seconds.

1. In the parameter list of the GetActivityTaskRequest().withActivityArn() constructor, replace the ACTIVITY\_ARN value with the ARN of [the activity task that you created](#_bookmark53)

[earlier (p. 30)](#_bookmark53).

### Step 4: Starting an Execution

When you start the execution of the state machine, your worker polls Step Functions for activities, performs its work (using the input that you provide), and returns its results.

#### To start the execution

1. On the ***ActivityStateMachine*** page, choose **Start execution**. The **New execution** page is displayed.
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. In the execution input area, replace the example data with the following:

{

"who" : "AWS Step Functions"

}

1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. In the **Execution Details** section, choose **Info** to view the **Execution Status** and the **Started** and

**Closed** timestamps.

1. In the **Execution Details** section, expand the **Output** section to view the output of your workﬂow.

### Step 5: Running and Stopping the Worker

To have the worker poll your state machine for activities, you must run the worker.

###### Note

After the execution completes, you should stop your worker. If you don't stop the worker, it will continue to run and poll for activities. When the execution is stopped, your worker has no source of tasks and generates a SocketTimeoutException during each poll.

To run and stop the worker

1. On the command line, navigate to the directory in which you created GreeterActivities.java.
2. To use the AWS SDK, add the full path of the lib and third-party directories to the dependencies of your build ﬁle and to your Java CLASSPATH. For more information, see [Downloading and Extracting the SDK](https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-install.html#download-and-extract-sdk) in the *AWS SDK for Java Developer Guide*.
3. Compile the ﬁle:

$ javac GreeterActivities.java

1. Run the ﬁle:

$ java GreeterActivities

1. In the Step Functions console, navigate to the **Execution Details** page.
2. When the execution completes, choose **Output** to see the results of your execution.
3. Stop the worker.

## Handling Error Conditions Using a State Machine

In this tutorial, you create an AWS Step Functions state machine with a Catch ﬁeld which uses an AWS Lambda function to respond with conditional logic based on error message type, a method called

*function error handling*. For more information, see [Function Error Handling](https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-mode-exceptions.html#nodejs-prog-model-custom-exceptions) in the *AWS Lambda Developer Guide*.

###### Note

You can also create state machines that Retry on timeouts or those that use Catch to transition to a speciﬁc state when an error or timeout occurs. For examples of these error handling techniques, see [Examples Using Retry and Using Catch (p. 108)](#_bookmark142).

Topics

* [Step 1: Creating an IAM Role for Lambda (p. 35)](#_bookmark59)
* [Step 2: Creating a Lambda Function That Fails (p. 35)](#_bookmark61)
* [Step 3: Testing the Lambda Function (p. 36)](#_bookmark63)
* [Step 4: Creating a State Machine with a Catch Field (p. 36)](#_bookmark64)
* [Step 5: Starting a New Execution (p. 38)](#_bookmark65)

### Step 1: Creating an IAM Role for Lambda

Both Lambda and Step Functions can execute code and access AWS resources (for example, data stored in Amazon S3 buckets). To maintain security, you must grant Lambda and Step Functions access to these resources.

Lambda requires you to assign an IAM role when you create a Lambda function in the same way Step Functions requires you to assign an IAM role when you create a state machine.

#### To create a role for Lambda

1. Sign in to the [IAM console](https://console.aws.amazon.com/iam/home) and choose **Roles**, **Create role**.
2. On the **Select type of trusted entity** page, under **AWS service**, select **Lambda** from the list, and then choose **Next: Permissions**.

###### Note

The role is automatically provided with a trust relationship that allows Lambda to use the role.

1. On the **Attach permissions policy** page, choose **Next: Review**.
2. On the **Review** page, type MyLambdaRole for **Role Name**, and then choose **Create role**.

The IAM role appears in the list of roles.

### Step 2: Creating a Lambda Function That Fails

Use a Lambda function to simulate an error condition.

###### Important

Ensure that your Lambda function is under the same AWS account and region as your state machine.

To create a Lambda function that fails

1. Log in to the [Lambda console](https://console.aws.amazon.com/lambda/home) and choose **Create a function**.
2. In the **Blueprints** section, type step-functions into the ﬁlter, and then choose the **step- functions-error** blueprint.
3. In the **Basic information** section, conﬁgure your Lambda function:
   1. For **Name**, type FailFunction.
   2. For **Role**, select **Choose an existing role**.
   3. For **Existing role**, select [the Lambda role that you created earlier (p. 35)](#_bookmark60).

###### Note

If the IAM role that you created doesn't appear in the list, the role might still need a few minutes to propagate to Lambda.

1. The following code is displayed in the **Lambda function code** pane:

'use strict';

exports.handler = (event, context, callback) => { function CustomError(message) {

this.name = 'CustomError'; this.message = message;

}

CustomError.prototype = new Error();

const error = new CustomError('This is a custom error!'); callback(error);

};

The context object returns the error message This is a custom error!.

1. Choose **Create function**.

When your Lambda function is created, note its Amazon Resource Name (ARN) in the upper-right corner of the page. For example:

arn:aws:lambda:us-east-1:123456789012:function:FailFunction

### Step 3: Testing the Lambda Function

Test your Lambda function to see it in operation.

#### To test your Lambda function

1. On the ***FailFunction*** page, choose **Test**.
2. On the **Conﬁgure test event** dialog box, type FailFunction for **Event name**, and then choose

Create.

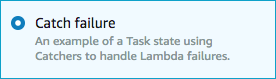
1. On the ***FailFunction*** page, **Test** your Lambda function.

The results of the test (the simulated error) are displayed at the bottom of the page.

### Step 4: Creating a State Machine with a Catch Field

Use the Step Functions console to create a state machine that uses a Task state with a Catch ﬁeld. Add a reference to your Lambda function in the Task state. The Lambda function is invoked and fails during execution. Step Functions retries the function twice using exponential backoﬀ between retries.

#### To create the state machine

1. Log in to the [Step Functions console](https://console.aws.amazon.com/states/home) and choose **Create state machine**.
2. On the **Create a state machine** page, select **Templates** and choose **Catch failure**.
3. **Name your state machine**, for example *Catchfailure*.

###### Note

State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:

* + Whitespace
  + Wildcard characters (? \*)
  + Bracket characters (< > { } [ ])
  + Special characters (: ; , \ | ^ ~ $ # % & ` ")
  + Control characters (\\u0000 - \\u001f or \\u007f - \\u009f).

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. In the **Code** pane, add the ARN of [the Lambda function that you created earlier (p. 35)](#_bookmark62) to the

Resource ﬁeld, for example:

{

"Comment": "A Catch example of the Amazon States Language using an AWS Lambda

function",

"StartAt": "*CreateAccount*", "States": {

"CreateAccount": { "Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:FailFunction*", "Catch": [ {

"ErrorEquals": ["CustomError"], "Next": "CustomErrorFallback"

}, {

"ErrorEquals": ["States.TaskFailed"], "Next": "ReservedTypeFallback"

}, {

"ErrorEquals": ["States.ALL"], "Next": "CatchAllFallback"

} ],

"End": true

},

"CustomErrorFallback": { "Type": "Pass",

"Result": "This is a fallback from a custom Lambda function exception", "End": true

},

"ReservedTypeFallback": { "Type": "Pass",

"Result": "This is a fallback from a reserved error code", "End": true

},

"CatchAllFallback": { "Type": "Pass",

"Result": "This is a fallback from any error code", "End": true

}

}

}

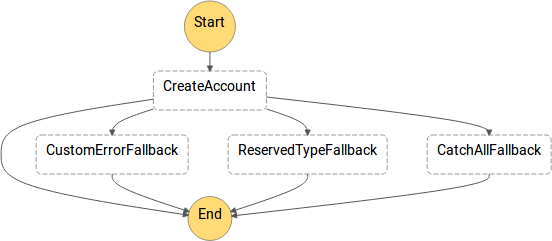
This is a description of your state machine using the Amazon States Language. It deﬁnes a single

Task state named CreateAccount. For more information, see [State Machine Structure (p. 129)](#_bookmark167).

For more information about the syntax of the Retry ﬁeld, see [Retrying After an Error (p. 146)](#_bookmark187).

###### Note

Unhandled errors in Lambda are reported as Lambda.Unknown in the error output. These include out-of-memory errors, function timeouts, and hitting the concurrent Lambda invoke limit. You can match on Lambda.Unknown, States.ALL, or States.TaskFailed to handle these errors. For more information about Lambda Handled and Unhandled errors, see FunctionError in the [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html#API_Invoke_ResponseSyntax).

1. Use the graph in the **Visual Workﬂow** pane to check that your Amazon States Language code describes your state machine correctly.

If you don't see the graph, choose in the **Visual Workﬂow** pane.

1. Choose **Next**.
2. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Create state machine**.

### Step 5: Starting a New Execution

After you create your state machine, you can start an execution.

#### To start a new execution

1. On the ***CatchStateMachine*** page, choose **New execution**.

Execution Using CloudWatch Events

The **New execution** page is displayed.

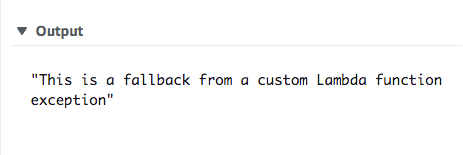
1. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

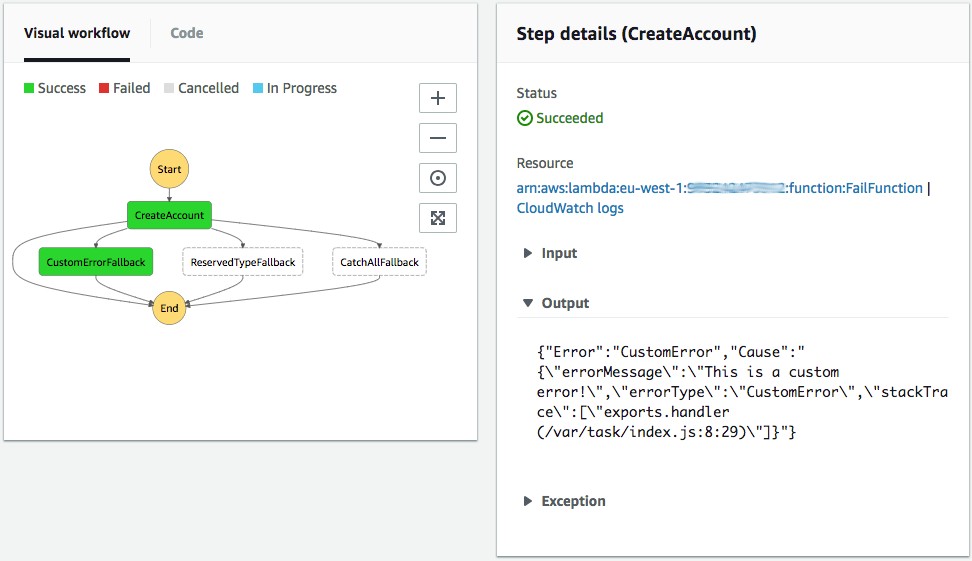
###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. In the **Execution Details** section, expand the **Output** section to view the output of your workﬂow.
2. To view your custom error message, select CreateAccount in the **Visual workﬂow** and expand the

**Output** section.

###### Note

You can preserve the state input along with the error by using ResultPath. See [Use](#_bookmark134)

[ResultPath to Include Both Error and Input in a Catch (p. 102)](#_bookmark134)

## Periodically Start a State Machine Execution Using CloudWatch Events

You can execute a Step Functions state machine in response to an event pattern or on a schedule using Amazon CloudWatch Events. This tutorial shows how to set a state machine as a target for a CloudWatch Events rule that starts the execution of a state machine every 5 minutes.

For more information about setting a Step Functions state machine as a target using the PutTarget

Amazon CloudWatch Events API action, see [Add a Step Functions state machine as a target](https://docs.aws.amazon.com/AmazonCloudWatchEvents/latest/APIReference/API_PutTargets.html#API_PutTargets_Example_Adds_a_Step_Functions_state_machine_as_a_target).

Topics

* [Step 1: Creating a State Machine (p. 40)](#_bookmark67)
* [Step 2: Creating a CloudWatch Events Rule (p. 40)](#_bookmark68)

### Step 1: Creating a State Machine

Before you can set a CloudWatch Events target, you must create a state machine.

* To create a basic state machine, use the [Getting Started (p. 12)](#_bookmark21) tutorial.
* If you already have a state machine, proceed to the next step.

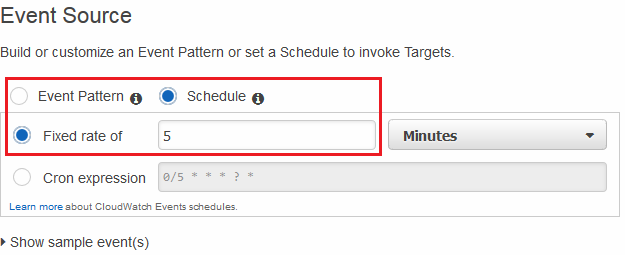
### Step 2: Creating a CloudWatch Events Rule

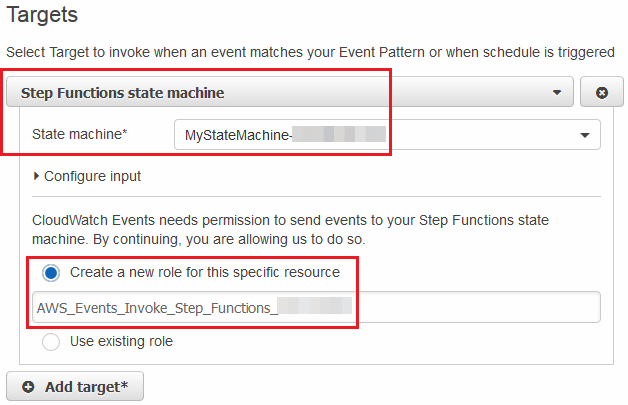
After you create your state machine, you can create your CloudWatch Events rule.

#### To create the rule

1. Navigate to the [CloudWatch Events console](https://console.aws.amazon.com/cloudwatch/), choose **Events**, and then choose **Create Rule**.

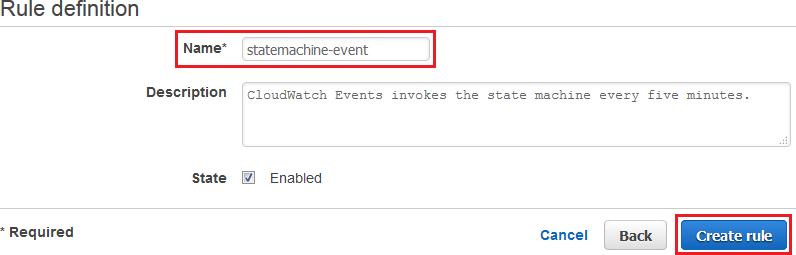
The **Step 1: Create rule** page is displayed.

1. In the **Event source** section, select **Schedule** and type 5 for **Fixed rate of**.
2. In the **Targets** section, choose **Add target** and from the list choose **Step Functions state machine**.

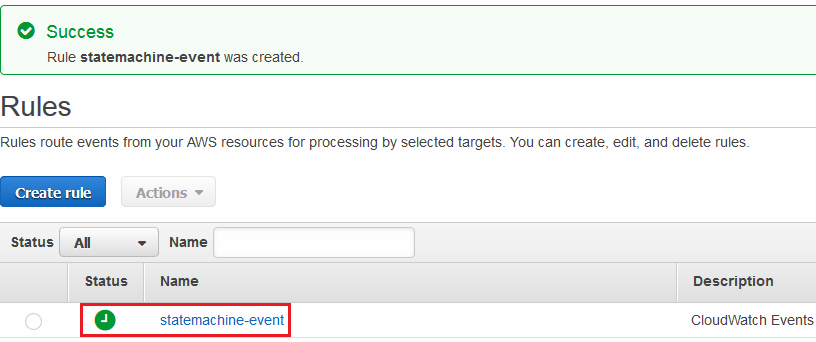


1. CloudWatch Events can create the IAM role needed for your event to run:
   * To create an IAM role automatically, select **Create a new role for this speciﬁc resource**.
   * To use an IAM role that you created before, choose **Use existing role**.
2. Choose **Conﬁgure details**.

The **Step 2: Conﬁgure rule details** page is displayed.

1. Type a **Name** for your rule (for example, statemachine-event), choose **Enabled** for **State**, and then choose **Create rule**.

The rule is created and the **Rules** page is displayed, listing all your CloudWatch Events rules.

in Response to Amazon S3 Events

A new execution of your state machine starts every 5 minutes.

## Starting a State Machine Execution in Response to Amazon S3 Events

You can use Amazon CloudWatch Events to execute a Step Functions state machine in response to an event or on a schedule.

This tutorial shows how to conﬁgure a state machine as a target for a CloudWatch Events rule. This will start an execution when ﬁles are added to an Amazon S3 bucket.

For a practical application, you could launch a state machine that performs operations on ﬁles that you add to the bucket, such as creating thumbnails or running Amazon Rekognition analysis on image and video ﬁles.

For this tutorial you start an execution of a simple Helloworld state machine by adding a ﬁle to an Amazon S3 bucket. Then we review example input of that execution to show what information is included in the input from CloudTrail.

Topics

* + [Prerequisite: Create a State Machine (p. 42)](#_bookmark70)
  + [Step 1: Create a Bucket in Amazon S3 (p. 43)](#_bookmark71)
  + [Step 2: Create a Trail in AWS CloudTrail (p. 43)](#_bookmark72)
  + [Step 3: Create a CloudWatch Events Rule (p. 43)](#_bookmark73)
  + [Step 4: Test the CloudWatch Rule (p. 45)](#_bookmark74)
  + [Example of Execution Input (p. 45)](#_bookmark75)

### Prerequisite: Create a State Machine

Before you can conﬁgure a CloudWatch Events target, you must create a state machine.

* To create a basic state machine, use the [Getting Started (p. 12)](#_bookmark21) tutorial.
* If you already have a Helloworld state machine, proceed to the next step.

### Step 1: Create a Bucket in Amazon S3

Now that you have a Helloworld state machine, you need an Amazon S3 bucket. In Step 3 of this tutorial, you set up a rule so that when a ﬁle is added to this bucket, CloudWatch Events triggers an execution of the state machine.

1. Navigate to the [Amazon S3 console](https://console.aws.amazon.com/s3/), and then choose **Create bucket**.
2. Enter a **Bucket name**, such as *username*-sfn-tutorial.

###### Note

Bucket names must be unique across all existing bucket names in all AWS Regions in Amazon S3. Use your own *username* to make this name unique. You need to create all resources in the same AWS Region.

1. Choose **Create**.

### Step 2: Create a Trail in AWS CloudTrail

Once you have created an Amazon S3 bucket, create a trail in CloudTrail.

For API events in Amazon S3 to match your CloudWatch Events rule, you must conﬁgure a trail in CloudTrail to receive those events.

1. Navigate to the [AWS CloudTrail console](https://console.aws.amazon.com/cloudtrail/), choose **View trails**, and then choose **Create trail**.
2. For **Trail name**, enter S3Event.
3. On the **S3** tab, select **Add S3 bucket**.
4. For **Bucket name**, enter the name of the Amazon S3 bucket you created earlier: *username*-sfn- tutorial ([Step 1: Create a Bucket in Amazon S3 (p. 43)](#_bookmark71)).
5. Under **Storage location**, choose **Yes** next to **Create a new S3 bucket**.
6. For **S3 bucket**, enter a name for a new bucket to store information about the actions of the Amazon S3 bucket you created earlier.

###### Note

This bucket name must be unique across all of Amazon S3. Include your *username* in the bucket name so that the name will be unique: *username*-sfn-tutorial-storage.

1. Choose **Create**.

### Step 3: Create a CloudWatch Events Rule

Once you have a state machine, and have created the Amazon S3 bucket and a trail in AWS CloudTrail, create your Amazon CloudWatch Events rule.

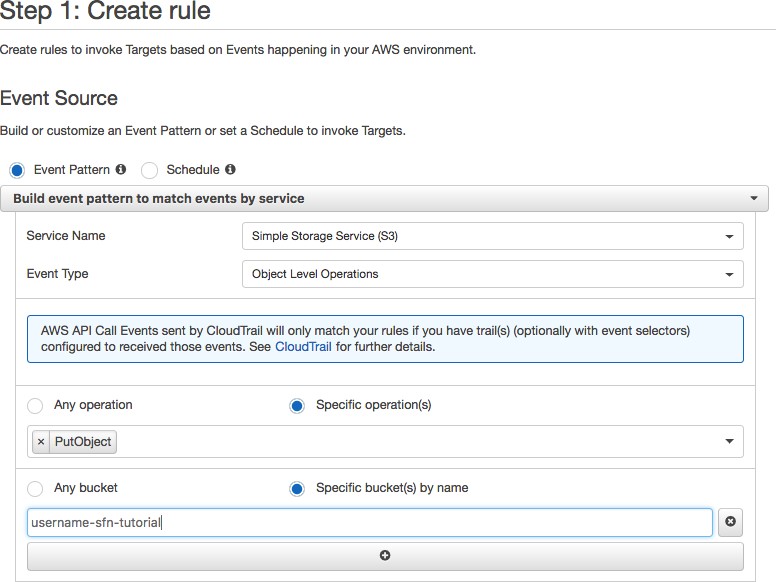
###### Note

You must conﬁgure CloudWatch Eventsin the same AWS Region as the Amazon S3 bucket.

To create the rule

1. Navigate to the [CloudWatch console](https://console.aws.amazon.com/cloudwatch/), choose **Events**, and then **Create Rule**. The **Step 1: Create rule** page is displayed.
2. In **Event source**, choose **Event Pattern**.
3. For **Service Name**, select **Simple Storage Service (S3)**.
4. For **Event Type**, select **Object Level Operations**.
5. Select **Speciﬁc operation(s)**, and then choose **PutObject**.
6. Choose **Speciﬁc bucket(s) by name** and enter the bucket name you created in Step 1 (*username*- sfn-tutorial).

The **Event Source** page should look like the following.



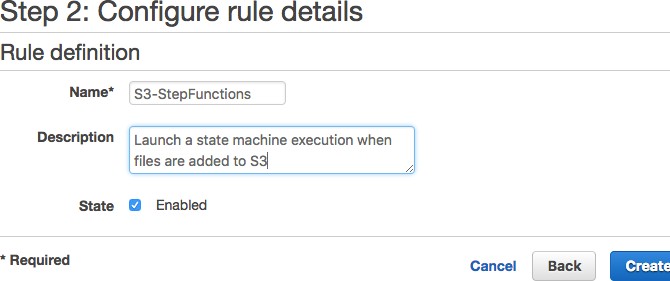
#### To create the target

1. In the **Targets** section, choose **Add target**.
2. From the list, choose **Step Functions state machine**, and in the **State machine** list, choose the state machine from Step 1 (Helloworld).
3. CloudWatch Events can create the IAM role that your event needs to run:
   * To create an IAM role automatically, choose **Create a new role for this speciﬁc resource**.
   * To use an IAM role that you created before, choose **Use existing role**.
4. Choose **Conﬁgure details**.

The **Step 2: Conﬁgure rule details** page is displayed.

1. Type a **Name** for your rule (for example, S3StepFunctions), select **Enabled** for **State**, and then choose **Create rule**.

The **Conﬁgure rule details** section should look like the following.

The rule is created and the **Rules** page is displayed, listing all your CloudWatch Events rules.

### Step 4: Test the CloudWatch Rule

Now that everything is in place, test adding a ﬁle to the Amazon S3 bucket, and then look at the input of the resulting state machine execution.

1. Add a ﬁle to your Amazon S3 bucket.

Navigate to the [Amazon S3 console](https://console.aws.amazon.com/s3/), select the bucket you created (*username*-sfn-tutorial), and then choose **Upload**.

1. Add a ﬁle (test.png in the following example), and then choose **Upload**.

This launches an execution of your state machine, passing information from AWS CloudTrail as the input.

1. Check the execution for your state machine.

Navigate to the [Step Functions console and select the state machine used in your CloudWatch](https://console.aws.amazon.com/states/) [Events rule (Helloworld)](https://console.aws.amazon.com/states/).

1. Select the most recent execution of that state machine and expand the **Input** section.

This input includes information such as the bucket name and the object name. In a real-world use case, a state machine can use this input to perform actions on that object.

### Example of Execution Input

The following example shows typical input to the state machine execution.

{

"version": "0",

"id": "8d6f9246-b781-44f8-a026-f1c1ab2c61f0",

"detail-type": "AWS API Call via CloudTrail", "source": "aws.s3",

"account": "123456789012", "time": "2018-09-12T00:25:10Z",

"region": "us-east-2",

"resources": [], "detail": {

"eventVersion": "1.05", "userIdentity": {

"type": "IAMUser",

"principalId": "AKIAIOSFODNN7EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/*username*", "accountId": "123456789012",

"accessKeyId": "AKIAI44QH8DHBEXAMPLE",

"userName": "*username*", "sessionContext": {

"attributes": {

"creationDate": "2018-09-11T20:10:38Z",

"mfaAuthenticated": "true"

}

},

"invokedBy": "signin.amazonaws.com"

},

"eventTime": "2018-09-12T00:25:10Z",

"eventSource": "s3.amazonaws.com", "eventName": "PutObject", "awsRegion": "us-east-2", "sourceIPAddress": "203.0.113.34", "userAgent": "signin.amazonaws.com", "requestParameters": {

"X-Amz-Date": "20180912T002509Z",

"bucketName": "*username*-sfn-tutorial", "X-Amz-Algorithm": "AWS4-HMAC-SHA256",

"x-amz-acl": "private",

"X-Amz-SignedHeaders": "content-type;host;x-amz-acl;x-amz-storage-class", "X-Amz-Expires": "300",

"key": "test.png",

"x-amz-storage-class": "STANDARD"

},

"responseElements": null, "additionalEventData": {

"x-amz-id-2": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km"

},

"requestID": "79104EXAMPLEB723",

"eventID": "cdc4b7ed-e171-4cef-975a-ad829d4123e8", "readOnly": false,

"resources": [

{

"type": "AWS::S3::Object",

"ARN": "arn:aws:s3:::*username*-sfn-tutorial-2/test.png"

},

{

"accountId": "123456789012",

"type": "AWS::S3::Bucket",

"ARN": "arn:aws:s3:::*username*-sfn-tutorial"

}

],

"eventType": "AwsApiCall", "recipientAccountId": "123456789012"

}

}

## Creating a Step Functions API Using API Gateway

You can use Amazon API Gateway to associate your AWS Step Functions APIs with methods in an API Gateway API. When an HTTPS request is sent to an API method, API Gateway invokes your Step Functions API actions.

This tutorial shows you how to create an API that uses one resource and the POST method to communicate with the StartExecution API action. You'll use the IAM console to create a role for API Gateway. Then, you'll use the API Gateway console to create an API Gateway API, create a resource and method, and map the method to the StartExecution API action. Finally, you'll deploy and test your API. For more information about this API action, see [StartExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html) in the *AWS Step Functions API Reference*.

###### Note

While Amazon API Gateway can start a Step Functions execution by calling [StartExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html)

you must call [DescribeExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DescribeExecution.html) to get the result.

Topics

* + [Step 1: Creating an IAM Role for API Gateway (p. 47)](#_bookmark77)
  + [Step 2: Creating your API Gateway API (p. 48)](#_bookmark79)
  + [Step 3: Testing and Deploying the API Gateway API (p. 50)](#_bookmark80)

### Step 1: Creating an IAM Role for API Gateway

Before you create your API Gateway API, you need to give API Gateway permission to call Step Functions API actions.

#### To create a role for API Gateway

1. Log in to the [IAM console](https://console.aws.amazon.com/iam/home) and choose **Roles**, **Create role**.
2. On the **Select type of trusted entity** page, under **AWS service**, select **API Gateway** from the list and then choose **Next: Permissions**.
3. On the **Attached permissions policy** page, choose **Next: Review**.
4. On the **Review** page, type APIGatewayToStepFunctions for **Role name** and then choose **Create role**.

The IAM role appears in the list of roles.

1. Choose the name of your role and note the **Role ARN**, for example:

arn:aws:iam::123456789012:role/APIGatewayToStepFunctions

#### To attach a policy to the IAM role

1. On the **Roles** page, search for your role (APIGatewayToStepFunctions) and then choose the role.
2. On the **Permissions** tab, choose **Attach Policy**.
3. On the **Attach Policy** page, search for AWSStepFunctionsFullAccess, choose the policy, and then choose **Attach Policy**.

### Step 2: Creating your API Gateway API

After you create your IAM role, you can create your custom API Gateway API.

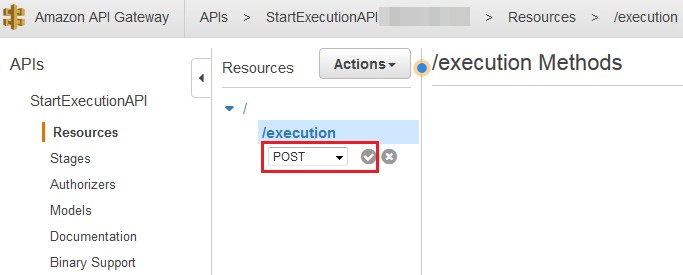
#### To create the API

1. Navigate to the [Amazon API Gateway console](https://console.aws.amazon.com/apigateway/) and choose **Get Started**.
2. On the **Create new API** page, choose **New API**.
3. In the **Settings** section, type StartExecutionAPI for the **API name**, and then choose **Create API**.

#### To create a resource

1. On the **Resources** page of ***StartExecutionAPI***, choose **Actions**, **Create Resource**.
2. On the **New Child Resource** page, type execution for **Resource Name**, and then choose **Create Resource**.

#### To create a POST Method

1. On the **/execution Methods** page, choose **Actions**, **Create Method**.
2. From the list, choose POST, and then select the checkmark.

#### To conﬁgure the method

On the **/execution - POST - Setup** page, conﬁgure the integration point for your method.

1. For **Integration Type**, choose **AWS Service**.
2. For **AWS Region**, choose a region from the list.

###### Note

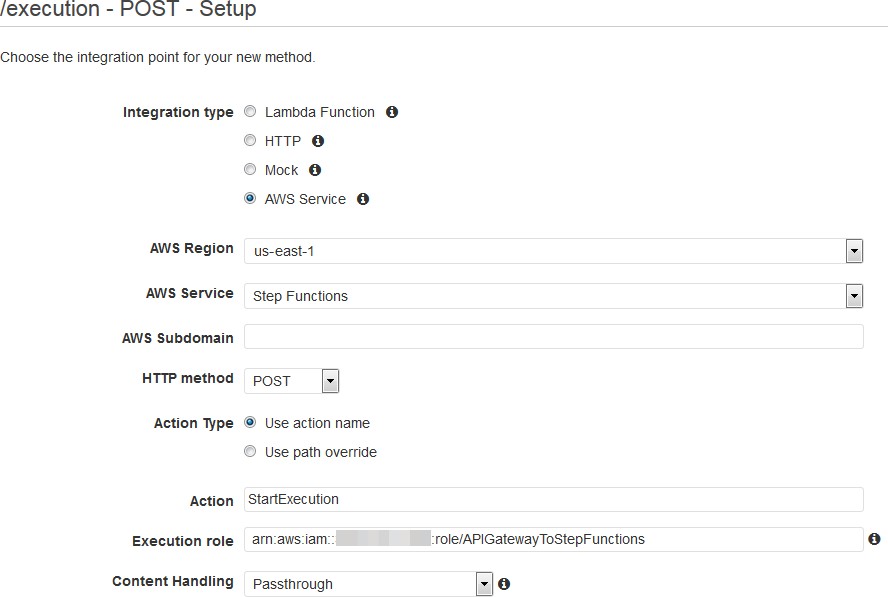
For regions that currently support Step Functions, see the [Supported Regions (p. 1)](#_bookmark2).

1. For **AWS Service**, choose **Step Functions** from the list.
2. For **HTTP Method**, choose **POST** from the list.

###### Note

All Step Functions API actions use the HTTP POST method.

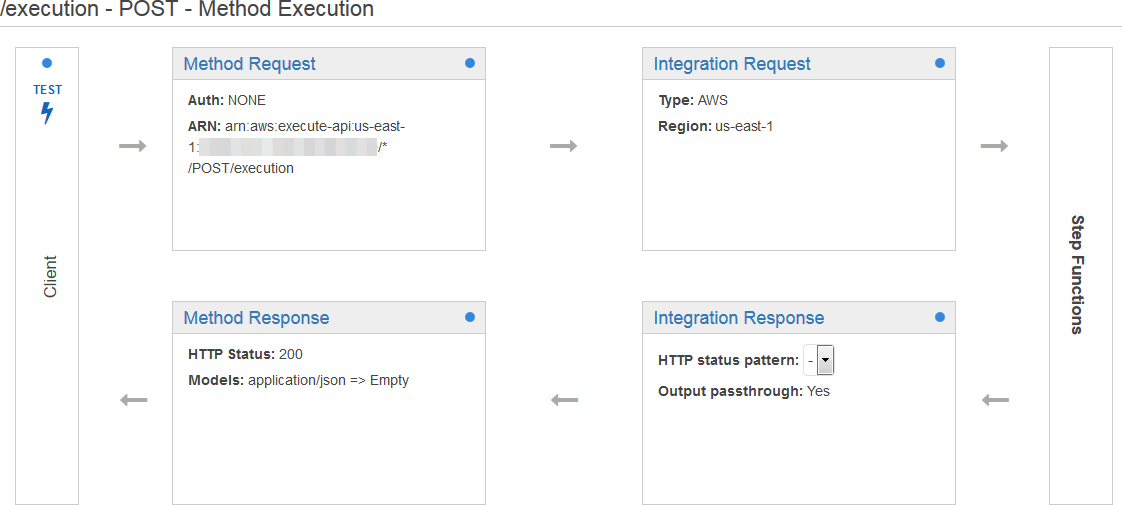
1. For **Action Type**, choose **Use action name**.
2. For **Action**, type StartExecution.
3. For **Execution Role**, type [the role ARN of the IAM role that you created earlier (p. 47)](#_bookmark78), for example:



arn:aws:iam::123456789012:role/APIGatewayToStepFunctions

1. Choose **Save**.

The visual mapping between API Gateway and Step Functions is displayed on the **/execution - POST**

**- Method Execution** page.

### Step 3: Testing and Deploying the API Gateway API

#### To test the communication between API Gateway and Step Functions

1. On the **/execution - POST - Method Execution** page, choose **Test**.
2. On the **/execution - POST - Method Test** page, copy the following request parameters into the **Request Body** section using the ARN of an existing state machine (or [create a new state machine (p. 12)](#_bookmark21)), and then choose **Test**.

{

"input": "{}",

"name": "MyExecution",

"stateMachineArn": "*arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld*"

}

###### Note

For more information, see the StartExecution [Request Syntax](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html#API_StartExecution_RequestSyntax) in the *AWS Step Functions API Reference*.

If you don't want to include the ARN of your state machine in the body of your API Gateway call, you can conﬁgure a body-mapping template, for example:

{

"input": "$util.escapeJavaScript($input.json('$'))", "stateMachineArn": "*arn:aws:states:us-*

*east-1:123456789012:stateMachine:HelloWorld*"

}

This approach allows you to have diﬀerent state machines based on your development stages (for example, dev, test, and prod). To release an update, you only need to change the stage variable, for example:

{

"input": "$util.escapeJavaScript($input.json('$'))", "stateMachineArn":

"$util.escapeJavaScript($stageVariables.get(*arn:aws:states:us- east-1:123456789012:stateMachine:HelloWorld*))"

}

1. The execution starts and the execution ARN and its epoch date are displayed under **Response Body**.

{

"executionArn": "*arn:aws:states:us-*

*east-1:123456789012:execution:HelloWorld:MyExecution*", "startDate": 1486768956.878

}

###### Note

You can view the execution by choosing your state machine on the [AWS Step Functions](https://console.aws.amazon.com/states/) [console](https://console.aws.amazon.com/states/).

#### To deploy your API

1. On the **Resources** page of ***StartExecutionAPI***, choose **Actions**, **Deploy API**.
2. In the **Deploy API** dialog box, select **[New Stage]** from the **Deployment stage** list, type alpha for

**Stage name**, and then choose **Deploy**.

#### To test your deployment

1. On the **Stages** page of ***StartExecutionAPI***, expand **alpha**, **/**, **/execution**, **POST**.
2. On the **alpha - POST - /execution** page, note the **Invoke URL**, for example:

https://a1b2c3d4e5.execute-api.us-east-1.amazonaws.com/alpha/execution

1. From the command line, run the curl command using the ARN of your state machine, and then invoke the URL of your deployment, for example:

curl -X POST -d '{"input": "{}","name": "MyExecution","stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"}' https://

a1b2c3d4e5.execute-api.us-east-1.amazonaws.com/alpha/execution

The execution ARN and its epoch date are returned, for example:

{"executionArn":"arn:aws:states:us-

east-1:123456789012:execution:HelloWorld:MyExecution","startDate":1.486772644911E9}

## Iterating a Loop Using Lambda

In this tutorial, you implement a design pattern that uses a state machine and an AWS Lambda function to iterate a loop a speciﬁc number of times.

Use this design pattern any time you need to keep track of the number of loops in a state machine. This implementation can help you break up large tasks or long-running executions into smaller chunks, or to end an execution after a speciﬁc number of events. You can use a similar implementation to periodically end and restart a long-running execution to avoid exceeding service limits for AWS Step Functions, AWS Lambda, or other AWS services.

Before you begin, go through the [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial to ensure you have created the necessary IAM role, and are familiar with using Lambda and Step Functions together.

Topics

* + [Step 1: Create a Lambda Function to Iterate a Count (p. 51)](#_bookmark82)
  + [Step 2: Test the Lambda Function (p. 52)](#_bookmark84)
  + [Step 3: Create a State Machine (p. 53)](#_bookmark85)
  + [Step 4: Start a New Execution (p. 56)](#_bookmark86)

### Step 1: Create a Lambda Function to Iterate a Count

By using a Lambda function you can track the number of iterations of a loop in your state machine. The following Lambda function receives input values for count, index, and step. It returns these values with an updated index and a Boolean named continue. The Lambda function sets continue to true if the index is less than count.

Your state machine then implements a Choice state that executes some application logic if continue is

true, or exits if it is false.

#### To create the Lambda function

1. Sign in to the [Lambda console](https://console.aws.amazon.com/lambda/home), and then choose **Create function**.
2. In the **Create function** section, choose **Author with code snippets**.
3. In the **Author with code snippets** section, conﬁgure your Lambda function, as follows:
   1. For **Name**, type Iterator.
   2. For **Runtime**, select **Node.js 6.10**.
   3. For **Role**, select **Choose an existing role**.
   4. For **Existing role**, select the Lambda role that you created in the [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial.

###### Note

If the IAM role that you created doesn't appear in the list, the role might still need a few minutes to propagate to Lambda.

* 1. Choose **Create function**.

When your Lambda function is created, make a note of its Amazon Resource Name (ARN) in the upper-right corner of the page. For example:

arn:aws:lambda:us-east-1:123456789012:function:Iterator

1. Copy the following code for the Lambda function into the **Conﬁguration** section of the ***Iterator***

page in the Lambda console.

exports.iterator = function iterator (event, context, callback) { let index = event.iterator.index

let step = event.iterator.step let count = event.iterator.count

index += step callback(null, {

index,

step, count,

continue: index < count

})

}

This code accepts input values for count, index, and step. It increments the index by the value of step and returns these values, and the Boolean continue. The value of continue is true if index is less than count.

1. Choose **Save**.

### Step 2: Test the Lambda Function

Run your Lambda function with numeric values to see it in operation. You can provide input values for your Lambda function that mimic an iteration, to see what output you get with speciﬁc input values.

#### To test your Lambda function

1. In the **Conﬁgure test event** dialog box, choose **Create new test event**, and then type

TestIterator for **Event name**.

1. Replace the example data with the following.

{

"Comment": "Test my Iterator function", "iterator": {

"count": 10,

"index": 5,

"step": 1

}

}

These values mimic what would come from your state machine during an iteration. The Lambda function will increment the index and return continue as true. Once the index is not less than the count, it will return continue as false. For this test, the index has already incremented to 5. The results should increment the index to 6 and set continue to true.

1. Choose **Create**.
2. On the ***Iterator*** page in your Lambda console, be sure **TestIterator** is listed, and then choose **Test**.

The results of the test are displayed at the top of the page. Choose **Details** and review the result.

{

"index": 6,

"step": 1,

"count": 10, "continue": true

}

###### Note

If you set index to 9 for this test, the index will increment to 10, and continue will be

false.

### Step 3: Create a State Machine

#### To create the state machine

1. Sign in to the [Step Functions console](https://console.aws.amazon.com/states/home), and then choose **Create a state machine**.

###### Important

Ensure that your state machine is under the same AWS account and region as the Lambda function you created earlier.

1. On the **Create a state machine** page, choose **Author with code snippets**. For **Give a name to your state machine**, enter IterateCount.

###### Note

State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:

* + Whitespace
  + Wildcard characters (? \*)
  + Bracket characters (< > { } [ ])
  + Special characters (: ; , \ | ^ ~ $ # % & ` ")
  + Control characters (\\u0000 - \\u001f or \\u007f - \\u009f).

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch.

To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. The following code describes a state machine with the following states:
   * ConfigureCount: Sets the default values for count, index, and step.

"ConfigureCount": { "Type": "Pass", "Result": {

"count": 10,

"index": 0,

"step": 1

},

"ResultPath": "$.iterator", "Next": "Iterator"

},

* + Iterator: References your Lambda function you created earlier, passing in the values conﬁgured in ConfigureCount.

"Iterator": {

"Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:Iterate*", "ResultPath": "$.iterator",

"Next": "IsCountReached"

},

* + IsCountReached: A choice state that will either run your sample work again or will go to Done

based on a boolean returned from your Iterator Lambda function.

"IsCountReached": { "Type": "Choice", "Choices": [

{

"Variable": "$.iterator.continue", "BooleanEquals": true,

"Next": "ExampleWork"

}

],

"Default": "Done"

},

* + ExampleWork: A stub for the work you want to accomplish in your execution. In this example it is a pass state. In an actual implementation this would be a task state. See [Tasks (p. 72)](#_bookmark100).
  + Done: The end state of your execution.

In the **Code** pane, add the following state machine deﬁnition using the Amazon Resource Name of [the Lambda function that you created earlier (p. 52)](#_bookmark83).

{

"Comment": "Iterator State Machine Example", "StartAt": "ConfigureCount",

"States": {

"ConfigureCount": { "Type": "Pass", "Result": {

"count": 10,

"index": 0,

"step": 1

},

"ResultPath": "$.iterator", "Next": "Iterator"

},

"Iterator": {

"Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:Iterate*", "ResultPath": "$.iterator",

"Next": "IsCountReached"

},

"IsCountReached": { "Type": "Choice", "Choices": [

{

"Variable": "$.iterator.continue", "BooleanEquals": true,

"Next": "ExampleWork"

}

],

"Default": "Done"

},

"ExampleWork": {

"Comment": "Your application logic, to run a specific number of times", "Type": "Pass",

"Result": { "success": true

},

"ResultPath": "$.result", "Next": "Iterator"

},

"Done": {

"Type": "Pass", "End": true

}

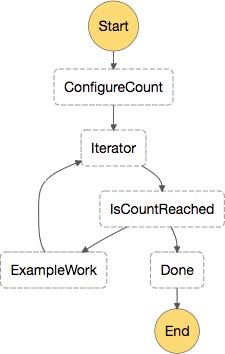
}

}

Be sure to update the Amazon Resource Name in the Iterator state above so that it references the Lambda you created earlier. For more information about the Amazon States Language, see [State](#_bookmark167) [Machine Structure (p. 129)](#_bookmark167).

1. Use the graph in the **Visual Workﬂow** pane to check that your Amazon States Language code describes your state machine correctly.

This graph shows the logic expressed in the above state machine code.



If you don't see the graph, choose in the **Visual Workﬂow** pane.

1. Choose **Next**.
2. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Create state machine**.

### Step 4: Start a New Execution

After you create your state machine, you can start an execution.

#### To start a new execution

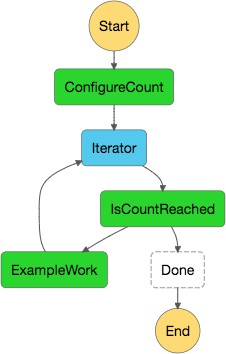
1. On the **IterateCount** page, choose **New execution**.
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

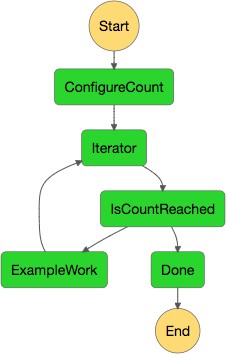
1. Choose **Start Execution**.

A new execution of your state machine starts, showing your running execution.



The execution increments in steps, tracking the count using your Lambda function. On each iteration, it performs the example work referenced in the ExampleWork state in your state machine.

1. (Optional) In the **Execution Details** section, choose the **Info** tab to view the **Execution Status** and the **Started** and **Closed** time stamps.
2. Once the count reaches the number conﬁgured in the ConfigureCount state in your state machine, the execution quits iterating and ends.

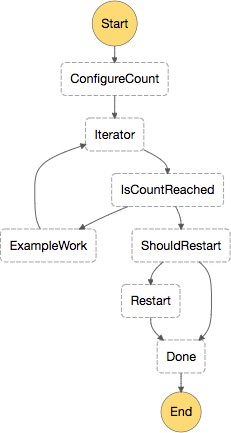


## Continue as a New Execution

AWS Step Functions is designed to run workﬂows that have a ﬁnite duration and number of steps. Executions are limited to a duration of one year, and a maximum of 25,000 events (see [Limits (p. 152)](#_bookmark195)). However, you can create a state machine that uses a Lambda function to start a new execution, before allowing the current execution to terminate. This enables you to have a state machine that can break large jobs into smaller workﬂows, or to have a state machine that runs indeﬁnitely.

This tutorial builds on the concept of using an external Lambda function to modify your workﬂow, which was demonstrated in the [Iterating a Loop Using Lambda (p. 51)](#_bookmark81) tutorial. You'll use the same Lambda function (Iterator) to iterate a loop for a speciﬁc number of times. In addition, you'll create another Lambda function to start a new execution of your workﬂow, and to decrement a count each time it starts a new execution. By setting the number of executions in the input, this state machine will end and restart an execution a speciﬁed number of times.

This tutorial shows you how to create a state machine with a Lambda function that can start a new execution, continuing your ongoing work in that new execution.



The state machine you'll create implements the following states.

|  |  |
| --- | --- |
| **State** | **Purpose** |
| ConfigureCount | A [Pass (p. 131)](#_bookmark170) state that conﬁgures the count, index, and step values that are used by the Iterator Lambda function to step through iterations of work. |
| Iterator | A [Task (p. 132)](#_bookmark171) state that references the Iterator Lambda function. |

|  |  |
| --- | --- |
| **State** | **Purpose** |
| IsCountReached | A [Choice (p. 135)](#_bookmark175) state that uses a Boolean value from the Iterator function to decide if the state machine should continue the example work, or move to the ShouldRestart choice state. |
| ExampleWork | In this example, ExampleWork is a Pass state that represents the  Task state that would perform work in an actual implementation. |
| ShouldRestart | A [Choice (p. 135)](#_bookmark175) state that uses the executionCount value to decide if it should end one execution and start another, or simply end. |
| Restart | A [Task (p. 132)](#_bookmark171) state that uses a Lambda function to start a new execution of your state machine. Like the Iterator function, this function also decrements a count. It passes that value to the input of the new execution. |

### Prerequisites

Before you begin, go through the [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial to ensure you have created an initial IAM role, and that you are familiar with using Lambda and Step Functions together.

Topics

* + [Step 1: Create an Iterate Lambda Function to Iterate a Count (p. 59)](#_bookmark89)
  + [Step 2: Create a Restart Lambda Function to Start a New Step Functions Execution (p. 61)](#_bookmark90)
  + [Step 3: Create a State Machine (p. 62)](#_bookmark91)
  + [Step 4: Update the IAM Policy (p. 64)](#_bookmark92)
  + [Step 5: Run an Execution (p. 64)](#_bookmark93)

### Step 1: Create an Iterate Lambda Function to Iterate a Count

###### Note

If you have completed the [Iterating a Loop Using Lambda (p. 51)](#_bookmark81) tutorial, you can skip this step and use that Lambda function.

This section, and the [Iterating a Loop Using Lambda (p. 51)](#_bookmark81) tutorial, shows how you can use a Lambda function to track a count so that you can track the number of iterations of a loop in your state machine.

The following Lambda function receives input values for count, index, and step. It returns these values with an updated index and a Boolean named continue. The Lambda function sets continue to true if the index is less than count.

Your state machine then implements a Choice state that executes some application logic if continue is

true, or moves on to ShouldRestart if continue is false.

To create the Iterate Lambda function

1. Sign in to the [Lambda console](https://console.aws.amazon.com/lambda/home), and then choose **Create function**.
2. In the **Create function** section, choose **Author from scratch**.
3. In the **Author with code snippets** section, conﬁgure your Lambda function, as follows:

Function to Iterate a Count

* 1. For **Name**, type Iterator.
  2. For **Runtime**, select **Node.js 6.10**.
  3. For **Role**, select **Choose an existing role**.
  4. For **Existing role**, select the Lambda role that you created in the [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial.

###### Note

If the IAM role that you created doesn't appear in the list, the role might still need a few minutes to propagate to Lambda.

* 1. Choose **Create function**.

When your Lambda function is created, make a note of its Amazon Resource Name (ARN) in the upper-right corner of the page. For example:

arn:aws:lambda:us-east-1:123456789012:function:Iterator

1. Copy the following code for the Lambda function into the **Conﬁguration** section of the ***Iterator***

page in the Lambda console.

exports.iterator = function iterator (event, context, callback) { let index = event.iterator.index

let step = event.iterator.step let count = event.iterator.count

index += step callback(null, {

index,

step, count,

continue: index < count

})

}

This code accepts input values for count, index, and step. It increments the index by the value of step and returns these values, and the Boolean continue. The value of continue is true if index is less than count.

1. Choose **Save**.

#### Test the Iterate Lambda Function

To see your Iterate function working, run it with numeric values. You can provide input values for your Lambda function that mimic an iteration to see what output you get with speciﬁc input values.

##### To test your Lambda function

1. In the **Conﬁgure test event** dialog box, choose **Create new test event**, and then type

TestIterator for **Event name**.

1. Replace the example data with the following.

{

"Comment": "Test my Iterator function",

"iterator": {

"count": 10,

"index": 5,

"step": 1

to Start a New Step Functions Execution

}

}

These values mimic what would come from your state machine during an iteration. The Lambda function increments the index and returns continue as true. Once the index is not less than the count, it returns continue as false. For this test, the index has already incremented to 5. The results should increment the index to 6 and set continue to true.

1. Choose **Create**.
2. On the ***Iterator*** page in your Lambda console, be sure **TestIterator** is listed, and then choose **Test**.

The results of the test are displayed at the top of the page. Choose **Details** and review the result.

{

"index": 6,

"step": 1,

"count": 10, "continue": true

}

###### Note

If you set index to 9 for this test, the index increments to 10, and continue is false.

### Step 2: Create a Restart Lambda Function to Start a New Step Functions Execution

1. Sign in to the [Lambda console](https://console.aws.amazon.com/lambda/home), and then choose **Create function**.
2. In the **Author with code snippets** section, conﬁgure your Lambda function, as follows:
   1. For **Name**, type Restart.
   2. For **Runtime**, select **Node.js 6.10**.
   3. For **Role**, select **Choose an existing role**.
   4. Under **Existing role**, select the role that includes the IAM policy you created previously.
   5. Choose **Create function**.

When your Lambda function is created, make a note of its Amazon Resource Name (ARN) in the upper-right corner of the page. For example:

arn:aws:lambda:us-east-1:123456789012:function:Restart

1. Copy the following code for the Lambda function into the **Conﬁguration** section of the ***Restart***

page in the Lambda console.

The following code decrements a count of the number of executions, and starts a new execution of your state machine, including the decremented value.

var aws = require('aws-sdk');

var sfn = new aws.StepFunctions();

exports.restart = function(event, context, callback) {

let StateMachineArn = event.restart.StateMachineArn; event.restart.executionCount -= 1;

event = JSON.stringify(event);

let params = {

input: event,

stateMachineArn: StateMachineArn

};

sfn.startExecution(params, function(err, data) { if (err) callback(err);

else callback(null,event);

});

}

1. Choose **Save**.

### Step 3: Create a State Machine

Now that you've created your two Lambda functions, create a state machine. In this state machine, the

ShouldRestart and Restart states are how you break your work across multiple executions.

###### Example ShouldRestart Choice state

This excerpt of your state machine shows the ShouldRestart [Choice (p. 135)](#_bookmark175) state. This state decides if you should restart the execution.

"ShouldRestart": { "Type": "Choice", "Choices": [

{

"Variable": "$.restart.executionCount", "NumericGreaterThan": 1,

"Next": "Restart"

}

],

The $.restart.executionCount value is included in the input of the initial execution. It's decremented by one each time the Restart function is called, and then placed into the input for each subsequent execution.

###### Example Restart Task state

This excerpt of your state machine shows the Restart [Task (p. 132)](#_bookmark171) state. This state uses the Lambda function you created earlier to restart the execution, and to decrement the count to track the remaining number of executions to start.

"Restart": { "Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:Restart*", "Next": "Done"

},

1. In the Step Functions console, choose **Create a state machine**.
2. Select **Author with code snippets**, and enter ContinueAsNew as your state machine name.
3. Paste the following into the Code pane.

###### Example ContinueAsNew state machine

{

"Comment": "Continue-as-new State Machine Example", "StartAt": "ConfigureCount",

"States": {

"ConfigureCount": { "Type": "Pass", "Result": {

"count": 100,

"index": -1,

"step": 1

},

"ResultPath": "$.iterator", "Next": "Iterator"

},

"Iterator": {

"Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:Iterator*", "ResultPath": "$.iterator",

"Next": "IsCountReached"

},

"IsCountReached": { "Type": "Choice", "Choices": [

{

"Variable": "$.iterator.continue", "BooleanEquals": true,

"Next": "ExampleWork"

}

],

"Default": "ShouldRestart"

},

"ExampleWork": {

"Comment": "Your application logic, to run a specific number of times", "Type": "Pass",

"Result": { "success": true

},

"ResultPath": "$.result", "Next": "Iterator"

},

"ShouldRestart": { "Type": "Choice", "Choices": [

{

"Variable": "$.restart.executionCount", "NumericGreaterThan": 0,

"Next": "Restart"

}

],

"Default": "Done"

},

"Restart": { "Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:Restart*", "Next": "Done"

},

"Done": {

"Type": "Pass", "End": true

}

}

}

1. Update the Resource string in the Restart and Iterator states to reference the respective Lambda functions you created earlier.
2. Choose **Next**.
3. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Select **Create state machine**.

###### Note

Save the Amazon Resource Name of this state machine.

### Step 4: Update the IAM Policy

To ensure your Lambda function has permissions to start a new Step Functions execution, attach an inline policy to the IAM role you use for your Restart Lambda function. For more information, see [Embedding Inline Policies](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_manage-attach-detach.html#embed-inline-policy-console) in the *IAM User Guide*.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "VisualEditor0", "Effect": "Allow", "Action": [

"states:StartExecution"

],

"Resource": "\*"

}

]

}

###### Note

You can update the "Resource": "\*" line in the previous example to reference the ARN of your ContinueAsNew state machine. This restricts the policy so that it can only start an execution of that speciﬁc state machine.

### Step 5: Run an Execution

To start an execution, provide input that includes the ARN of the state machine and an

executionCount for how many times it should start a new execution.

1. On the **ContinueAsNew** page, choose **New execution**.
2. In the **Input** section, on the **New execution** page, enter Test1 for the execution name. Then enter the following in the **Input**.

{

"restart": {

"StateMachineArn": "*arn:aws:states:us-*

*east-1:123456789012:stateMachine:ContinueAsNew*",

"executionCount": *4*

}

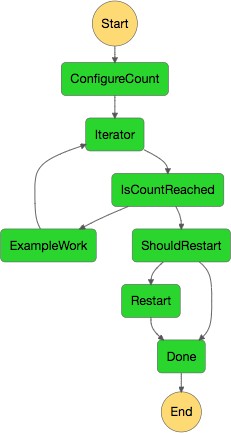
}

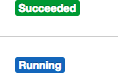
1. Update the StateMachineArn ﬁeld with the Amazon Resource Name for your ContinueAsNew

state machine.

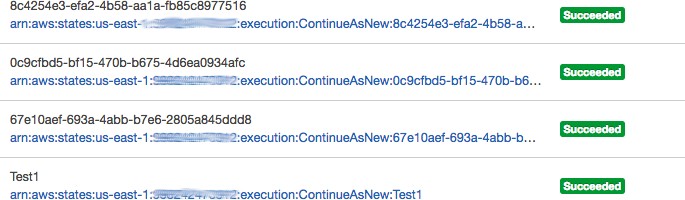
1. Choose **Start Execution**.

The **Visual Workﬂow** graph will display the ﬁrst of the four executions. Before it completes, it will pass through the Restart state and start a new execution.



With this execution complete, you can go look at the next execution that's running. Select the **ContinueAsNew** link at the top to see the list of executions. You should see both the recently closed execution, and an ongoing execution that the Restart Lambda function kicked oﬀ.

Once all the executions are complete, you should see four successful executions in the list. The ﬁrst execution started displays the name you chose, and subsequent executions have a generated name.



## Using Code Snippets Create a State to Send an Amazon SNS message

In this tutorial you will generate a code snippet that sends a text message using Amazon Simple Notiﬁcation Service (SNS). Step Functions integrates with some other AWS services, and in this tutorial you will pass parameters directly to Amazon SNS from your state machine deﬁnition.

For more information, on how Step Functions integrates with other AWS services directly from the Amazon States Language, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Code Snippets (p. 82)](#_bookmark113)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Topics

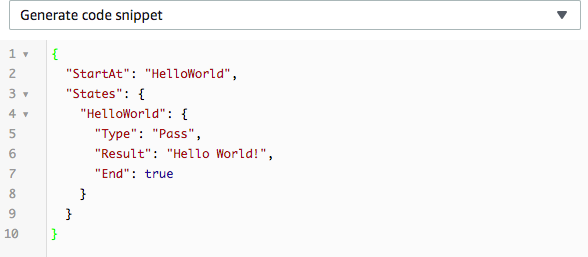
* + [Step 1: Generate a Code Snippet (p. 66)](#_bookmark95)
  + [Step 2: Update Your State Machine Deﬁnition (p. 68)](#_bookmark96)
  + [Step 3: Start an Execution (p. 70)](#_bookmark97)

### Step 1: Generate a Code Snippet

To generate a code snippet, you must start by editing a state machine deﬁnition.

1. From the Step Functions console, select **Get started** or **Create state machine**.
2. Choose **Author with code snippets** and type a name for your state machine.

The default HelloWorld state machine is displayed in the **State machine deﬁnition**.



1. Select the **Generate Code Snippet** drop-down and choose **Amazon SNS: Publish a message**.

The **Generate SNS Publish task state** window is displayed.

1. Under **Destination**, select **Enter phone number**, and enter your cellphone number.

Use the format [+][country code][subscriber number including area code]. For example: +12065550123.

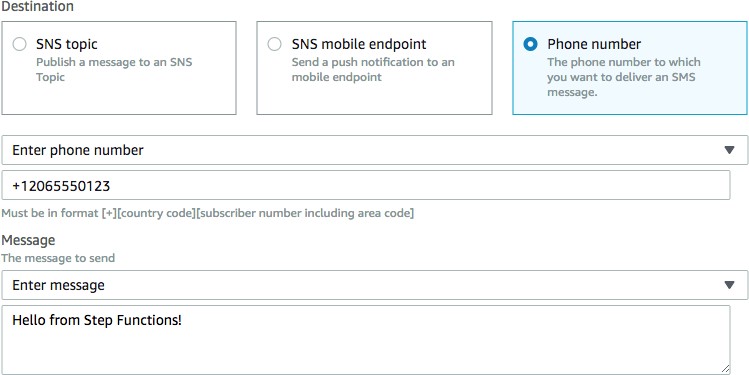
1. Under **Message**, select **Enter message** and type some text that you want to send as an SMS message.

###### Note

You can also choose **Specify message at runtime with state input**. This option will allow you to use a reference path to select a message from the input of your state machine execution. For more information see:

* + [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)
  + [Reference Paths (p. 143)](#_bookmark184)
  + [Pass State Input as Parameters Using Paths (p. 81)](#_bookmark112)

As you conﬁgure options on the **Generate SNS Publish task state** page, the **Preview** section updates with the Amazon States Language code for a task state with the necessary options. For instance, if you select these options:



With these options selected, the generated code snippet displayed in the **Preview** area is:

"Amazon SNS: Publish a message": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"Message": "Hello from Step Functions!", "PhoneNumber": "+12065550123"

},

"Next": "NEXT\_STATE"

}

###### Note

Under the **Task state options** section you can also conﬁgure Retry, Catch, and

TimeoutSeconds options. See [Error Handling (p. 104)](#_bookmark138).

### Step 2: Update Your State Machine Deﬁnition

Now that you have conﬁgured your Amazon SNS options, paste the generated code snippet into your state machine deﬁnition and update the existing Amazon States Language code.

1. Once you have entered the conﬁguration options, and have reviewed the code in the **Preview**

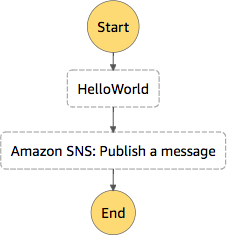
section, select **Copy to clipboard**.

1. Place your cursor after the closing bracket of the HelloWorld state in your state machine deﬁnition.



Enter a comma, press Enter to start a new line, and paste your code snippet into your state machine deﬁnition.

1. Change the last line of the Amazon SNS: Publish a message state from *"Next": "NEXT\_STATE"* to *"End": true*.
2. Change the last line of the HelloWorld state from *"End": true* to *"Next": "Amazon SNS: Publish a message"*.
3. Choose in the **Visual Workﬂow** pane. Check the visual workﬂow to ensure your new state is included.



1. (Optional) Indent the JSON to make your code easier to read. Your state machine deﬁnition should look like this.

{

"StartAt":"HelloWorld",

"States":{

"HelloWorld":{ "Type":"Pass",

"Result":"Hello World!", "Next":"Amazon SNS: Publish a message"

},

"Amazon SNS: Publish a message":{ "Type":"Task", "Resource":"arn:aws:states:::sns:publish", "Parameters":{

"Message":"Hello from Step Functions!", "PhoneNumber":"+12065550123"

},

"End":true

}

}

}

1. Choose **Next**.
2. Create or enter an IAM role.
   * To create a new IAM role for Step Functions, select **Create an IAM role for me**, and enter a **Name**

for your role.

* + If you have [previously created an IAM role (p. 171)](#_bookmark216) with the correct permissions for your state machine, select **Choose an existing IAM role**. Select a role from the drop-down, or provide an ARN for that role.

###### Note

If you delete the IAM role that Step Functions creates, Step Functions can't recreate it later. Similarly, if you modify the role (for example, by removing Step Functions from the principals in the IAM policy), Step Functions can't restore its original settings later.

1. Choose **Create state machine**.

### Step 3: Start an Execution

Once it has been created, the page from your new state machine is displayed.

1. Review the details of your state machine, including the Amazon Resource Name (ARN), the related IAM ARN, and the state machine deﬁnition.
2. Select the **Executions** tab and choose **Start execution**.
3. (Optional) Enter a name for your execution.

###### Note

If we had chosen **Specify message at runtime with state input** when creating our Amazon SNS code snippet, we would include a message in the **Input - optional**. For now you can use the default state input.

Choose **Start execution**.

If you conﬁgured a valid cell phone number in your code snippet, you should have received a text message from Amazon SNS that was triggered directly by your state machine execution.

# How Step Functions Works

To understand AWS Step Functions, you will need to be familiar with a number of important concepts. This section describes how Step Functions works.

Topics

* [States (p. 71)](#_bookmark99)
* [Tasks (p. 72)](#_bookmark100)
* [Transitions (p. 92)](#_bookmark123)
* [State Machine Data (p. 92)](#_bookmark124)
* [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)
* [Executions (p. 104)](#_bookmark137)
* [Error Handling (p. 104)](#_bookmark138)
* [Read Consistency (p. 111)](#_bookmark143)
* [Templates (p. 111)](#_bookmark144)
* [Tagging (p. 112)](#_bookmark145)

States

A ﬁnite state machine can express an algorithm as a number of states, their relationships, and their input and output. AWS Step Functions allows you to coordinate individual tasks by expressing your workﬂow as a ﬁnite state machine, written in the Amazon States Language. Individual states can make decisions based on their input, perform actions, and pass output to other states. In Step Functions you express your workﬂows in the Amazon States Language, and the Step Functions console provides a graphical representation of that state machine to help visualize your application logic.

States are elements in your state machine. A state is referred to by its *name*, which can be any string, but which must be unique within the scope of the entire state machine.

###### Note

An instance of a state exists until the end of its execution. States can perform a variety of functions in your state machine:

* Do some work in your state machine (a [Task (p. 72)](#_bookmark100) state).
* Make a choice between branches of execution (a [Choice (p. 135)](#_bookmark175) state)
* Stop an execution with a failure or success (a [Fail (p. 140)](#_bookmark179) or [Succeed (p. 140)](#_bookmark178) state)
* Simply pass its input to its output or inject some ﬁxed data (a [Pass (p. 131)](#_bookmark170) state)
* Provide a delay for a certain amount of time or until a speciﬁed time/date (a [Wait (p. 139)](#_bookmark177) state)
* Begin parallel branches of execution (a [Parallel (p. 140)](#_bookmark180) state)

For example, here is a example state named HelloWorld which performs a Lambda function:

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction", "Next": "AfterHelloWorldState",

"Comment": "Run the HelloWorld Lambda function"

}

States share a number of common features:

* Each state must have a Type ﬁeld indicating what type of state it is.
* Each state can have an optional Comment ﬁeld to hold a human-readable comment about, or description of, the state.
* Each state (except a Succeed or Fail state) requires a Next ﬁeld or, alternatively, can become a terminal state by specifying an End ﬁeld.

###### Note

A Choice state may have more than one Next but only one within each Choice Rule. A

Choice state cannot use End.

Certain state types require additional ﬁelds, or may redeﬁne common ﬁeld usage.

For more information regarding the various states that you can deﬁne using Amazon States Language, see [States (p. 130)](#_bookmark168).

Once you create a state machine and have executed it, you can access information about each state, its input and output, when it was active and for how long, by viewing the **Execution Details** page in the Step Functions console.

## Tasks

All work in your state machine is done by *tasks*. A task performs work by using an activity, a Lambda function, or by passing parameters to the API actions of other services.

* AWS Step Functions can invoke Lambda functions directly from a task state. A Lambda function is a cloud-native task that runs on AWS Lambda. You can write Lambda functions in a variety of

programming languages, using the AWS Management Console or by uploading code to Lambda. For more information see [??? (p. 18)](#_bookmark38).

* Step Functions can coordinate some AWS services directly from a task state. For more information see [Service Integrations (p. 80)](#_bookmark107).
* An activity consists of program code that waits for an operator to perform an action or to provide input. You can host activities on Amazon EC2, on Amazon ECS, or even on mobile devices. Activities poll Step Functions using the GetActivityTask and SendTaskSuccess, SendTaskFailure, and SendTaskHeartbeat API actions.

Amazon States Language represents tasks by setting a state's type to Task and by providing the task with the ARN of the activity or Lambda function. For more information about specifying task types, see [Task (p. 132)](#_bookmark171) in [Amazon States Language (p. 128)](#_bookmark165).

For examples of how diﬀerent kinds of tasks are used, see the [Tutorials (p. 16)](#_bookmark29) section.

### Activities

Activities are an AWS Step Functions feature that enables you to have a task in your state machine where the work is performed by a *worker* that can be hosted on Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Container Service (Amazon ECS), mobile devices—basically anywhere.

Topics

* + [Overview (p. 73)](#_bookmark102)
  + [Waiting For an Activity Task to Complete (p. 73)](#_bookmark103)
  + [APIs Related to Activity Tasks (p. 73)](#_bookmark104)
  + [Next Steps (p. 74)](#_bookmark105)
  + [Example Activity Worker in Ruby (p. 74)](#_bookmark106)

Overview

In AWS Step Functions, activities are a way to associate code running somewhere (known as an *activity worker*) with a speciﬁc task in a state machine. You can create an activity in the Step Functions console, or by calling [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html). This provides an Amazon Resource Name (ARN) for your task state. Use this ARN to poll the task state for work in your activity worker.

###### Note

Activities are not versioned and are expected to be backward compatible. If you must make a backward-incompatible change to an activity, create a *new* activity in Step Functions using a unique name.

An activity worker can be an application running on an Amazon EC2 instance, an AWS Lambda function, a mobile device: any application that can make an HTTP connection, hosted anywhere. When Step Functions reaches an activity task state, the workﬂow waits for an activity worker to poll for a task. An activity worker polls Step Functions by using [GetActivityTask](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html), and sending the ARN for the related activity. GetActivityTask returns a response including input (a string of JSON input for the task) and a [taskToken](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html#StepFunctions-GetActivityTask-response-taskToken) (a unique identiﬁer for the task). After the activity worker completes its work, it can provide a report of its success or failure by using [SendTaskSuccess](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskSuccess.html) or [SendTaskFailure](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskFailure.html). These two calls use the taskToken provided by GetActivityTask to associate the result with that task.

#### Waiting For an Activity Task to Complete

Conﬁgure how long a state waits by setting TimeoutSeconds in the task deﬁnition. To keep the task active and waiting, periodically send a heartbeat from your activity worker using [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html) within the time conﬁgured in TimeoutSeconds. By conﬁguring a long timeout duration and actively sending a heartbeat, an activity in Step Functions can wait up to a year for an execution to complete.

For example, if you need a workﬂow that waits for the outcome of a long process, do the following.

1. Create an activity in the console, or by using [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html). Make a note of the activity ARN.
2. Reference that ARN in an activity task state in your state machine deﬁnition and set

TimeoutSeconds.

1. Implement an activity worker that polls for work by using [GetActivityTask](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html), referencing that activity ARN.
2. Use [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html) periodically within the time you set in [HeartbeatSeconds (p. 132)](#_bookmark171) in your state machine task deﬁnition to keep the task from timing out.
3. Start an execution of your state machine.
4. Start your activity worker process.

The execution pauses at the activity task state and waits for your activity worker to poll for a task. Once a taskToken is provided to your activity worker, your workﬂow will wait for [SendTaskSuccess](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskSuccess.html) or [SendTaskFailure](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskFailure.html) to provide a status. If the execution doesn't receive either of these or a [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html) call before the time conﬁgured in TimeoutSeconds, the execution will fail and the execution history will contain an ExecutionTimedOut event.

#### APIs Related to Activity Tasks

Step Functions provides APIs for creating and listing activities, requesting a task, and for managing the ﬂow of your state machine based on the results of your worker.

The following are the Step Functions APIs that are related to activities:

* [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html)
* [GetActivityTask](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html)
* [ListActivities](https://docs.aws.amazon.com/step-functions/latest/apireference/API_ListActivities.html)
* [SendTaskFailure](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskFailure.html)
* [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html)
* [SendTaskSuccess](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskSuccess.html)

###### Note

Polling for activity tasks with GetActivityTask can cause latency in some implementations. See [Avoid Latency When Polling for Activity Tasks (p. 151)](#_bookmark194).

#### Next Steps

For a detailed look at creating a state machine that uses an activity worker, see [Creating an Activity State](#_bookmark51) [Machine (p. 30)](#_bookmark51).

#### Example Activity Worker in Ruby

The following is an example activity worker that uses the AWS SDK for Ruby to show you how to use best practices and implement your own activity worker..

The code implements a consumer-producer pattern with a conﬁgurable number of threads for pollers and activity workers. The poller threads are constantly long polling the activity task. Once an activity task is retrieved, it's passed through a bounded blocking queue for the activity thread to pick it up.

* For more information about the AWS SDK for Ruby, see the [AWS SDK for Ruby API Reference](https://docs.aws.amazon.com/sdk-for-ruby/v3/api/).
* To download this code and related resources, see [step-functions-ruby-activity-worker](https://github.com/aws-samples/step-functions-ruby-activity-worker) on GitHub.com.

The following Ruby code is the main entry point for this example Ruby activity worker.

require\_relative '*../lib/step\_functions/activity*' credentials = Aws::SharedCredentials.new

region = '*us-west-2*' activity\_arn = '*ACTIVITY\_ARN*'

activity = StepFunctions::Activity.new( credentials: credentials,

region: region, activity\_arn: activity\_arn, workers\_count: 1,

pollers\_count: 1,

heartbeat\_delay: 30

)

# The start method takes as argument the block that is the actual logic of your custom activity.

activity.start do |input|

{ result: :SUCCESS, echo: input['value'] }

The code includes defaults you can change to reference your activity, and to adapt it to your speciﬁc implementation. This code takes as input the actual implementation logic, allows you to reference your speciﬁc activity and credentials, and enables you to conﬁgure the number of threads and heartbeat delay. For more information and to download the code, see [Step Functions Ruby Activity Worker](https://github.com/aws-samples/step-functions-ruby-activity-worker).

|  |  |
| --- | --- |
| **Item** | **Description** |
| require\_relative | Relative path to the following example activity worker code. |

|  |  |
| --- | --- |
| **Item** | **Description** |
| region | AWS Region of your activity. |
| workers\_count | The number of threads for your activity worker. For most implementations, between 10 and  20 threads should be suﬃcient. The longer the activity takes to process, the more threads it might need. As an estimate, multiply the number of process activities per second by the 99th percentile activity processing latency, in seconds. |
| pollers\_count | The number of threads for your pollers. Between 10 and 20 threads should be suﬃcient for most implementations. |
| heartbeat\_delay | The delay in seconds between heartbeats. |
| input | Implementation logic of your activity. |

The following is the Ruby activity worker, referenced with ../lib/step\_functions/activity in your code.

require 'set' require 'json' require 'thread' require 'logger' require 'aws-sdk'

module Validate

def self.positive(value)

raise ArgumentError, 'Argument has to positive' if value <= 0 value

end

def self.required(value)

raise ArgumentError, 'Argument is required' if value.nil? value

end end

module StepFunctions

class RetryError < StandardError def initialize(message)

super(message) end

end

def self.with\_retries(options = {}, &block) retries = 0

base\_delay\_seconds = options[:base\_delay\_seconds] || 2 max\_retries = options[:max\_retries] || 3

begin

block.call rescue => e

puts e

if retries < max\_retries retries += 1

sleep base\_delay\_seconds\*\*retries retry

end

raise RetryError, 'All retries of operation had failed' end

end

class Activity

def initialize(options = {})

@states = Aws::States::Client.new(

credentials: Validate.required(options[:credentials]), region: Validate.required(options[:region]),

http\_read\_timeout: Validate.positive(options[:http\_read\_timeout] || 60)

)

@activity\_arn = Validate.required(options[:activity\_arn]) @heartbeat\_delay = Validate.positive(options[:heartbeat\_delay] || 60) @queue\_max = Validate.positive(options[:queue\_max] || 5) @pollers\_count = Validate.positive(options[:pollers\_count] || 1) @workers\_count = Validate.positive(options[:workers\_count] || 1) @max\_retry = Validate.positive(options[:workers\_count] || 3)

@logger = Logger.new(STDOUT) end

def start(&block)

@sink = SizedQueue.new(@queue\_max) @activities = Set.new start\_heartbeat\_worker(@activities) start\_workers(@activities, block, @sink) start\_pollers(@activities, @sink)

wait end

def queue\_size

return 0 if @sink.nil? @sink.size

end

def activities\_count

return 0 if @activities.nil? @activities.size

end private

def start\_pollers(activities, sink) @pollers = Array.new(@pollers\_count) do

PollerWorker.new( states: @states,

activity\_arn: @activity\_arn, sink: sink,

activities: activities, max\_retry: @max\_retry

)

end @pollers.each(&:start)

end

def start\_workers(activities, block, sink) @workers = Array.new(@workers\_count) do

ActivityWorker.new( states: @states, block: block, sink: sink,

activities: activities, max\_retry: @max\_retry

)

end @workers.each(&:start)

end

def start\_heartbeat\_worker(activities) @heartbeat\_worker = HeartbeatWorker.new(

states: @states, activities: activities,

heartbeat\_delay: @heartbeat\_delay, max\_retry: @max\_retry

)

@heartbeat\_worker.start end

def wait sleep

rescue Interrupt shutdown

ensure

Thread.current.exit end

def shutdown stop\_workers(@pollers) wait\_workers(@pollers) wait\_activities\_drained stop\_workers(@workers) wait\_activities\_completed shutdown\_workers(@workers)

shutdown\_worker(@heartbeat\_worker) end

def shutdown\_workers(workers) workers.each do |worker|

shutdown\_worker(worker) end

end

def shutdown\_worker(worker) worker.kill

end

def wait\_workers(workers) workers.each(&:wait)

end

def wait\_activities\_drained wait\_condition { @sink.empty? }

end

def wait\_activities\_completed wait\_condition { @activities.empty? }

end

def wait\_condition(&block) loop do

break if block.call sleep(1)

end end

def stop\_workers(workers) workers.each(&:stop)

end

class Worker

def initialize

@logger = Logger.new(STDOUT) @running = false

end

def run

raise 'Method run hasn\'t been implemented' end

def process loop do

begin

break unless @running run

rescue => e puts e

@logger.error('Unexpected error had occurred') @logger.error(e)

end end

end

def start

return unless @thread.nil? @running = true

@thread = Thread.new do process

end end

def stop

@running = false end

def kill

return if @thread.nil? @thread.kill

@thread = nil end

def wait @thread.join

end end

class PollerWorker < Worker def initialize(options = {})

@states = options[:states] @activity\_arn = options[:activity\_arn] @sink = options[:sink]

@activities = options[:activities] @max\_retry = options[:max\_retry] @logger = Logger.new(STDOUT)

end

def run

activity\_task = StepFunctions.with\_retries(max\_retry: @max\_retry) do begin

@states.get\_activity\_task(activity\_arn: @activity\_arn) rescue => e

@logger.error('Failed to retrieve activity task') @logger.error(e)

end end

return if activity\_task.nil? || activity\_task.task\_token.nil? @activities.add(activity\_task.task\_token) @sink.push(activity\_task)

end end

class ActivityWorker < Worker def initialize(options = {})

@states = options[:states] @block = options[:block] @sink = options[:sink]

@activities = options[:activities] @max\_retry = options[:max\_retry] @logger = Logger.new(STDOUT)

end

def run

activity\_task = @sink.pop

result = @block.call(JSON.parse(activity\_task.input)) send\_task\_success(activity\_task, result)

rescue => e send\_task\_failure(activity\_task, e)

ensure

@activities.delete(activity\_task.task\_token) unless activity\_task.nil? end

def send\_task\_success(activity\_task, result)

StepFunctions.with\_retries(max\_retry: @max\_retry) do begin

@states.send\_task\_success(

task\_token: activity\_task.task\_token, output: JSON.dump(result)

)

rescue => e

@logger.error('Failed to send task success') @logger.error(e)

end end

end

def send\_task\_failure(activity\_task, error)

StepFunctions.with\_retries do begin

@states.send\_task\_failure(

task\_token: activity\_task.task\_token, cause: error.message

)

rescue => e

@logger.error('Failed to send task failure') @logger.error(e)

end end

end end

class HeartbeatWorker < Worker def initialize(options = {}) @states = options[:states]

@activities = options[:activities] @heartbeat\_delay = options[:heartbeat\_delay] @max\_retry = options[:max\_retry]

@logger = Logger.new(STDOUT) end

def run sleep(@heartbeat\_delay) @activities.each do |token|

send\_heartbeat(token) end

end

def send\_heartbeat(token)

StepFunctions.with\_retries(max\_retry: @max\_retry) do begin

@states.send\_task\_heartbeat(token) rescue => e

@logger.error('Failed to send heartbeat for activity') @logger.error(e)

end end

rescue => e

@logger.error('Failed to send heartbeat for activity') @logger.error(e)

end end

end end

### AWS Service Integrations

Step Functions integrates with some AWS services, including the ability to directly call and pass parameters to the API of those services. You can use Step Functions to coordinate these services directly from the Amazon States Language. For instance, using Step Functions you can coordinate other services to:

* Invoke a Lambda function.
* Run a AWS Batch job and then perform diﬀerent actions based on the results
* Insert or get an item from DynamoDB
* Run an Amazon ECS task and wait for it to complete
* Publish a topic in Amazon SNS
* Send a message in Amazon SQS

For working examples of state machines that control other AWS services, see [Sample Projects (p. 114)](#_bookmark149). These sample projects will deploy a state machine and all related resources.

Topics

* + [Connect to Resources (p. 80)](#_bookmark108)
  + [Pass Parameters to a Service API (p. 81)](#_bookmark110)
  + [Code Snippets (p. 82)](#_bookmark113)
  + [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114)

Connect to Resources

Connections to supported services are conﬁgured with an Amazon Resource Name in the Resource ﬁeld of a Task state. For a list of integrated services that can be managed directly from a Step Functions task state, see [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114)

For instance, to publish an Amazon SNS topic:

"Resource": "arn:aws:states:::sns:publish",

The above example references the [Publish](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html) API of Amazon SNS, which will accept diﬀerent parameters. See [Call Amazon SNS With Step Functions (p. 88)](#_bookmark119) for a list of supported parameters for Amazon SNS.

##### Make Synchronous Connections to Resources

When you reference a resource that has a response, the Amazon Resource Name will end with .sync. For instance, when submitting a AWS Batch job, the Resource line in the state machine deﬁnition reads:

"Resource": "arn:aws:states:::batch:submitJob.sync",

Synchronous connections allow you to wait for a task to complete. The task will pause until the task is complete, and will then progress to the next state.

###### Note

For information on conﬁguring IAM for connected resources, see [IAM Policies for Integrated](#_bookmark224) [Services (p. 174)](#_bookmark224).

#### Pass Parameters to a Service API

Use the Parameters ﬁeld in a Task state to control what parameters are passed to a service API.

##### Pass Static JSON as Parameters

You can include a JSON object directly in your state machine deﬁnition to pass as a parameter to a resource. For example, to set the RetryStrategy parameter for the SubmitJob API for AWS Batch, you could include in your parameters:

"RetryStrategy": {

"attempts": 5

}

You can pass multiple parameters with static JSON as well. As a more complete example, here are the

Resource and Parameters of the speciﬁcation of a task that publishes an Amazon SNS topic:

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"TopicArn": "arn:aws:sns:us-east-1:123456789012:myTopic", "Message": "test message",

"MessageStructure": "json", "MessageAttributes": {

"my attribute no 1": { "DataType": "String",

"StringValue": "value of my attribute no 1"

},

"my attribute no 2": { "DataType": "String",

"StringValue": "value of my attribute no 2"

}

}

},

##### Pass State Input as Parameters Using Paths

You can pass portions of the state input into parameters by using [Paths (p. 143)](#_bookmark183). A Path is a string, beginning with $, that is used to identify components within JSON text. Step Functions paths use [JsonPath](https://github.com/json-path/JsonPath) syntax.

To specify that a parameter use a path to reference a JSON node in the input, end the name of the parameter with .$. For instance, if you have text in your state input in a node named message, you could pass that to a parameter by referencing the input JSON with a path. With the following state input:

{

"comment": "A message in the state input", "input": {

"message": "foo",

"otherInfo": "bar",

},

"data": "example"

}

You could pass the message foo as a parameter using:

"Parameters": {"Message.$": "$.input.message"},

For more information about using parameters in Step Functions, see:

* [Input and Output Processing (p. 94)](#_bookmark128)
* [InputPath and Parameters (p. 95)](#_bookmark129)

###### Note

For a list of services that can be controlled directly from the Amazon States Language, see [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114).

#### Code Snippets

Code snippets are a way to easily conﬁgure the options for a new state in your state machine deﬁnition. When you edit or create a state machine, the top of the code pane will include a **Generate code snippet** menu. Selecting an option from the **Generate code snippet** menu will open a window to conﬁgure parameters speciﬁc to that state, and will generate Amazon States Language code based on the options you choose.

For instance, if you choose the **AWS Batch: Manage a job** code snippet, you can conﬁgure:

* **Batch job name** — You can either specify the job name, or specify it at runtime using a path.
* **Batch job deﬁnition** — You can select the ARN of an existing an existing AWS Batch job in your account, enter the job deﬁnition, or choose to specify it at runtime using a path.
* **Batch job queue** — You can select the ARN of an existing an existing AWS Batch job queue in your account, enter the job queue deﬁnition, or choose to specify it at runtime using a path.
* **Run synchronously** — Selecting this option will conﬁgure Step Functions to wait until the AWS Batch job completes before continuing to the next state.

###### Note

For more information on specifying service parameters by using input to your state machine execution, see [Pass State Input as Parameters Using Paths (p. 81)](#_bookmark112).

Once you have conﬁgured your AWS Batch options you can specify error handling options for your state, such as Retry, Catch, and TimeoutSeconds. For more information, see [Errors (p. 146)](#_bookmark185) in the Amazon States Language section.

To learn more about Step Functions service integrations, see:

* [AWS Service Integrations (p. 80)](#_bookmark107)
* [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114)
* [Using Code Snippets (p. 66)](#_bookmark94)

#### Supported AWS Service Integrations for Step Functions

The following topics include the supported APIs, parameters, request/response syntax, and provide example code in the Amazon States Language for coordinating other AWS services. Integrated services that you can manage directly in the Step Functions Amazon States Language are:

Topics

* + [Invoke Lambda with Step Functions (p. 83)](#_bookmark115)
  + [Manage AWS Batch With Step Functions (p. 83)](#_bookmark116)
  + [Call DynamoDB APIs With Step Functions (p. 84)](#_bookmark117)
  + [Manage Amazon ECS/Fargate Tasks With Step Functions (p. 86)](#_bookmark118)
  + [Call Amazon SNS With Step Functions (p. 88)](#_bookmark119)
  + [Call Amazon SQS With Step Functions (p. 89)](#_bookmark120)
  + [Manage AWS Glue Jobs With Step Functions (p. 90)](#_bookmark121)
  + [Manage Amazon SageMaker Training Jobs With Step Functions (p. 90)](#_bookmark122)

Invoke Lambda with Step Functions

It is simple and convenient to use a Lambda function for implementing task states. Include the ARN of your Lambda function in the Resource ﬁeld of a task state. Step Functions will wait for the Lambda function to complete. The output of the Lambda function will be the result of the task.

###### Note

For more information on managing state input, output and results, see:

* + - [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)
    - [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)

The following includes a Task state that invokes a Lambda function.

"States": { "HelloWorld": {

"Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction", "End": true

}

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Manage AWS Batch With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported APIs:

###### Note

Parameters in Step Functions are expressed in CamelCase, even when the native service API is

pascalCase.

* [SubmitJob](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html)
  + [Request syntax](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#API_SubmitJob_RequestSyntax)
  + Supported parameters:
    - [ArrayProperties](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-arrayProperties)
    - [ContainerOverrides](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-containerOverrides)
    - [DependsOn](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-dependsOn)
    - [JobDefinition](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-jobDefinition)
    - [JobName](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-jobName)
    - [JobQueue](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-jobQueue)
    - [Parameters](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-parameters)
    - [RetryStrategy](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-retryStrategy)
    - [Timeout](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#Batch-SubmitJob-request-timeout)

###### Note

Only use Timeout with [asynchronous (p. 81)](#_bookmark109) connections. For [synchronous (p. 81)](#_bookmark109) connections, see [Use Timeouts to Avoid Stuck Executions (p. 150)](#_bookmark190).

* + [Response syntax](https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html#API_SubmitJob_ResponseSyntax)

The following includes a Task state that submits an AWS Batch job and waits for it to complete.

{

"StartAt": "BATCH\_JOB",

"States": {

"BATCH\_JOB": {

"Type": "Task",

"Resource": "arn:aws:states:::batch:submitJob.sync", "Parameters": {

"JobDefinition": "preprocessing", "JobName": "PreprocessingBatchJob", "JobQueue": "SecondaryQueue", "Parameters.$": "$.batchjob.parameters", "ContainerOverrides": {

"vcpus": 4

}

},

"End": true

}

}

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Call DynamoDB APIs With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

###### Note

There is a limit on the maximum input or result data size for a task in Step Functions. This limits you to 32,768 characters of data when you send to, or receive data from, another service. See [Limits Related to State Machine Executions (p. 153)](#_bookmark198).

Supported DynamoDB APIs and syntax:

* [GetItem](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html)
  + [Request syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#API_GetItem_RequestSyntax)
  + Supported parameters:
    - [Key](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-Key)
    - [TableName](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-TableName)
    - [AttributesToGet](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-AttributesToGet)
    - [ConsistentRead](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-ConsistentRead)
    - [ExpressionAttributeNames](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-ExpressionAttributeNames)
    - [ProjectionExpression](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-ProjectionExpression)
    - [ReturnConsumedCapacity](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#DDB-GetItem-request-ReturnConsumedCapacity)
  + [Response syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html#API_GetItem_ResponseSyntax)
* [PutItem](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html)
  + [Request syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#API_PutItem_RequestSyntax)
  + Supported parameters:
    - [Item](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-Item)
    - [TableName](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-TableName)
    - [ConditionalOperator](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ConditionalOperator)
    - [ConditionExpression](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ConditionExpression)
    - [Expected](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-Expected)
    - [ExpressionAttributeNames](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ExpressionAttributeNames)
    - [ExpressionAttributeValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ExpressionAttributeValues)
    - [ReturnConsumedCapacity](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ReturnConsumedCapacity)
    - [ReturnItemCollectionMetrics](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ReturnItemCollectionMetrics)
    - [ReturnValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#DDB-PutItem-request-ReturnValues)
  + [Response syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#API_PutItem_ResponseSyntax)
* [DeleteItem](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html)
  + [Request syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestSyntax)
  + Supported parameters:
    - [Key](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [TableName](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ConditionalOperator](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ConditionExpression](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [Expected](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ExpressionAttributeNames](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ExpressionAttributeValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ReturnConsumedCapacity](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ReturnItemCollectionMetrics](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
    - [ReturnValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_RequestParameters)
  + [Response syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html#API_DeleteItem_ResponseSyntax)
* [UpdateItem](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html)

• [Request synta](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#API_UpdateItem_RequestSyntax)x

* + Supported parameters: 85
    - [Key](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [TableName](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [AttributeUpdates](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ConditionalOperator](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ConditionExpression](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [Expected](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ExpressionAttributeNames](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ExpressionAttributeValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ReturnConsumedCapacity](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ReturnItemCollectionMetrics](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [ReturnValues](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
    - [UpdateExpression](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_RequestParameters)
  + [Response syntax](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html#API_UpdateItem_ResponseSyntax)

The following is a Task state that retrieves a message from Amazon DynamoDB.

"Read Next Message from DynamoDB": { "Type": "Task",

"Resource": "arn:aws:states:::dynamodb:getItem", "Parameters": {

"TableName": "TransferDataRecords-DDBTable-3I41R5L5EAGT", "Key": {

"MessageId": {"S.$": "$.List[0]"}

}

},

"ResultPath": "$.DynamoDB", "Next": "Send Message to SQS"

},

To see this state in a working example, see the [Transfer Data Records (Lambda, DynamoDB,](#_bookmark158) [SQS) (p. 120)](#_bookmark158) sample project.

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Manage Amazon ECS/Fargate Tasks With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported Amazon ECS APIs and syntax:

###### Note

Parameters in Step Functions are expressed in CamelCase, even when the native service API is

pascalCase.

* [RunTask](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html) starts a new task using the speciﬁed task deﬁnition.
  + [Request syntax](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#API_RunTask_RequestSyntax)
  + Supported parameters:
    - [Cluster](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-cluster)
    - [Group](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-group)
    - [LaunchType](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-launchType)
    - [NetworkConfiguration](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-networkConfiguration)
    - [Overrides](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-overrides)
    - [PlacementConstraints](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-placementConstraints)
    - [PlacementStrategy](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-placementStrategy)
    - [PlatformVersion](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-platformVersion)
    - [TaskDefinition](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#ECS-RunTask-request-taskDefinition)
  + [Response syntax](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html#API_RunTask_ResponseSyntax)

###### Note

For the Overrides parameter, Step Functions does not support executionRoleArn or

taskRoleArn as containerOverrides.

Passing data to an Amazon ECS task

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

You can use overrides to override the default command for a container, and pass input to your ECS Tasks. See, [ContainerOverride](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_ContainerOverride.html). In the example, we have used JsonPath to pass values to the Task from the input to the Task state.

The following includes a Task state that runs an Amazon ECS task and waits for it to complete.

{

"StartAt": "Run an ECS Task and wait for it to complete", "States": {

"Run an ECS Task and wait for it to complete": { "Type": "Task",

"Resource": "arn:aws:states:::ecs:runTask.sync", "Parameters": {

"Cluster": "*cluster-arn*", "TaskDefinition": "*job-id*", "Overrides": {

"ContainerOverrides": [

{

"Name": "*container-name*", "Command.$": "$.commands"

}

]

}

},

"End": true

}

}

}

For the above example, if the input to the execution is:

{

"commands": [

"test command 1",

"test command 2",

"test command 3"

]

}

The "Command.$": "$.commands" line in ContainerOverrides passes the commands from the state input to the container.

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Call Amazon SNS With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported APIs:

###### Note

There is a limit on the maximum input or result data size for a task in Step Functions. This limits you to 32,768 characters of data when you send to, or receive data from, another service. See [Limits Related to State Machine Executions (p. 153)](#_bookmark198).

* [Publish](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html)
  + [Request syntax](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_Example_1_Request)
  + Supported Parameters
    - [Message](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [MessageAttributes](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [MessageStructure](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [PhoneNumber](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [Subject](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [TargetArn](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
    - [TopicArn](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_RequestParameters)
  + [Response syntax](https://docs.aws.amazon.com/sns/latest/api/API_Publish.html#API_Publish_Example_1_Response)

The following includes a Task state that publishes an Amazon SNS topic.

{

"StartAt": "Publish to SNS", "States": {

"Publish to SNS": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"TopicArn": "arn:aws:sns:us-east-1:123456789012:myTopic", "Message.$": "$.input.message",

"MessageStructure": "json", "MessageAttributes": {

"my attribute no 1": { "DataType": "String",

"StringValue": "value of my attribute no 1"

},

"my attribute no 2": { "DataType": "String",

"StringValue": "value of my attribute no 2"

}

}

},

"End": true

}

}

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Call Amazon SQS With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported APIs:

###### Note

There is a limit on the maximum input or result data size for a task in Step Functions. This limits you to 32,768 characters of data when you send to, or receive data from, another service. See [Limits Related to State Machine Executions (p. 153)](#_bookmark198).

* [SendMessage](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html)
  + [DelaySeconds](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
  + [MessageAttribute](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
  + [MessageBody](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
  + [MessageDeduplicationId](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
  + [MessageGroupId](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
  + [QueueUrl](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_RequestParameters)
* [Response syntax](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_SendMessage.html#API_SendMessage_ResponseElements)

The following includes a Task state that sends an Amazon SQS message.

{

"StartAt": "Send to SQS", "States": {

"Send to SQS": { "Type": "Task",

"Resource": "arn:aws:states:::sqs:sendMessage", "Parameters": {

"QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/myQueue", "MessageBody.$": "$.input.message",

"MessageAttributes": { "my attribute no 1": {

"DataType": "String",

"StringValue": "value of my attribute no 1"

},

"my attribute no 2": { "DataType": "String",

"StringValue": "value of my attribute no 2"

}

}

},

"End": true

}

}

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Manage AWS Glue Jobs With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported APIs:

* [StartJobRun](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
* Supported Parameters:
  + [JobName](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [JobRunId](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [Arguments](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [AllocatedCapacity](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [Timeout](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [SecurityConfiguration](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)
  + [NotificationProperty](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-runs.html#aws-glue-api-jobs-runs-StartJobRun)

The following includes a Task state that starts an AWS Glue job.

"Glue StartJobRun": { "Type": "Task",

"Resource": "arn:aws:states:::glue:startJobRun.sync", "Parameters": {

"JobName": "GlueJob-JTrRO5l98qMG"

},

"Next": "ValidateOutput"

},

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

##### Manage Amazon SageMaker Training Jobs With Step Functions

Step Functions can control some AWS services directly from the Amazon States Language. For more information, see:

* [Service Integrations (p. 80)](#_bookmark107)
* [Pass Parameters to a Service API (p. 81)](#_bookmark110)

Supported Amazon SageMaker APIs and syntax:

* [CreateTrainingJob](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html)
  + [Request syntax](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestSyntax)
  + Supported parameters:
    - [AlgorithmSpecification](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [HyperParameters](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [InputDataConfig](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [OutputDataConfig](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [ResourceConfig](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [RoleArn](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [StoppingCondition](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [Tags](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [TrainingJobName](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
    - [VpcConfig](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_RequestParameters)
  + [Response syntax](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html#API_CreateTrainingJob_ResponseSyntax)
* [CreateTransformJob](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html)
  + [Request syntax](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestSyntax)
  + Supported parameters:
    - [BatchStrategy](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [Environment](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [MaxConcurrentTransforms](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [MaxPayloadInMB](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [ModelName](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [Tags](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [TransformInput](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [TransformJobName](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [TransformOutput](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
    - [TransformResources](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_RequestParameters)
  + [Response syntax](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTransformJob.html#API_CreateTransformJob_ResponseSyntax)

###### Note

AWS Step Functions will not automatically create a policy for CreateTransformJob when you create a state machine that integrates with Amazon SageMaker. You must attach an inline policy to the created role. For more infomation, see this example IAM policy: [CreateTrainingJob (p. 181)](#_bookmark235).

The following includes a Task state that creates a Amazon SageMaker transform job, specifying the Amazon S3 location for DataSource and TransformOutput.

{

"SageMaker CreateTransformJob": { "Type": "Task",

"Resource": "arn:aws:states:::sagemaker:createTransformJob.sync", "Parameters": {

"ModelName": "SageMakerCreateTransformJobModel-9iFBKsYti9vr", "TransformInput": {

"CompressionType": "None", "ContentType": "text/csv", "DataSource": {

"S3DataSource": { "S3DataType": "S3Prefix",

"S3Uri": "s3://my-s3bucket-example-1/TransformJobDataInput.txt"

}

}

},

"TransformOutput": {

"S3OutputPath": "s3://my-s3bucket-example-1/TransformJobOutputPath"

},

"TransformResources": { "InstanceCount": 1, "InstanceType": "ml.m4.xlarge"

},

"TransformJobName": "sfn-binary-classification-prediction"

},

"Next": "ValidateOutput"

},

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

## Transitions

When an execution of a state machine is launched, the system begins with the state referenced in the top-level StartAt ﬁeld. This ﬁeld (a string) must exactly match, including case, the name of one of the states.

After executing a state, AWS Step Functions uses the value of the Next ﬁeld to determine the next state to advance to.

Next ﬁelds also specify state names as strings, and must match the name of a state speciﬁed in the state machine description exactly (case-sensitive).

For example, the following state includes a transition to NextState:

"SomeState" : {

...,

"Next" : "NextState"

}

Most states permit only a single transition rule via the Next ﬁeld. However, certain ﬂow-control states (for example, a Choice state) allow you to specify multiple transition rules, each with its own Next ﬁeld. The [Amazon States Language (p. 128)](#_bookmark165) provides details about each of the state types you can specify, including information about how to specify transitions.

States can have multiple incoming transitions from other states.

The process repeats until it reaches a terminal state (a state with "Type": Succeed, "Type": Fail, or

"End": true), or a runtime error occurs.

The following rules apply to states within a state machine:

* States can occur in any order within the enclosing block, but the order in which they're listed doesn't aﬀect the order in which they're run, which is determined by the contents of the states themselves.
* Within a state machine, there can be only one state designated as the start state, which is designated by the value of the StartAt ﬁeld in the top-level structure.
* Depending on your state machine logic—for example, if your state machine has multiple branches of execution—you may have more than one end state.
* If your state machine consists of only one state, it can be both the start state and the end state.

## State Machine Data

State Machine data takes the following forms:

* The initial input into a state machine
* Data passed between states
* The output from a state machine

This section describes how state machine data is formatted and used in AWS Step Functions.

Topics

* + [Data Format (p. 93)](#_bookmark125)
  + [State Machine Input/Output (p. 93)](#_bookmark126)
  + [State Input/Output (p. 93)](#_bookmark127)

### Data Format

State machine data is represented by JSON text, so you can provide values using any data type supported by JSON:

###### Note

* + - Numbers in JSON text format conform to JavaScript semantics. These numbers typically correspond to double-precision [IEEE-854](https://standards.ieee.org/findstds/standard/854-1987.html) values.
    - The following is valid JSON text: stand-alone, quote-delimited strings; objects; arrays; numbers; Boolean values; and null.
    - The output of a state becomes the input into the next state. However, you can restrict states to working on a subset of the input data by using [Input and Output Processing (p. 143)](#_bookmark182).

### State Machine Input/Output

You can give AWS Step Functions initial input data by passing it to a [StartExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html) action when you start an execution, or by passing initial data using the Step Functions console. Initial data is passed to the state machine's StartAt state. If no input is provided, the default is an empty object ({}).

The output of the execution is returned by the last state (terminal). This output appears as JSON text in the execution's result. You can retrieve execution results from the execution history using external callers (for example, in the [DescribeExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DescribeExecution.html) action). You can view execution results on the Step Functions console.

### State Input/Output

Each state's input consists of JSON text from the preceding state or, for the StartAt state, the input into the execution. Certain ﬂow-control states echo their input to their output.

In the following example, the state machine adds two numbers together:

1. Deﬁne the Lambda function.

function Add(input) {

var numbers = JSON.parse(input).numbers; var total = numbers.reduce(

function(previousValue, currentValue, index, array) { return previousValue + currentValue; });

return JSON.stringify({ result: total });

}

1. Deﬁne the state machine.

{

"Comment": "An example that adds two numbers together.", "StartAt": "Add",

"Version": "1.0",

"TimeoutSeconds": 10, "States":

{

"Add": {

"Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:Add", "End": true

}

}

}

1. Start an execution with the following JSON text:

{ "numbers": [3, 4] }

The Add state receives the JSON text and passes it to the Lambda function The Lambda function returns the result of the calculation to the state.

The state returns the following value in its output:

{ "result": 7 }

Because Add is also the ﬁnal state in the state machine, this value is returned as the state machine's output.

If the ﬁnal state returns no output, then the state machine returns an empty object ({}).

For more information, see [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)

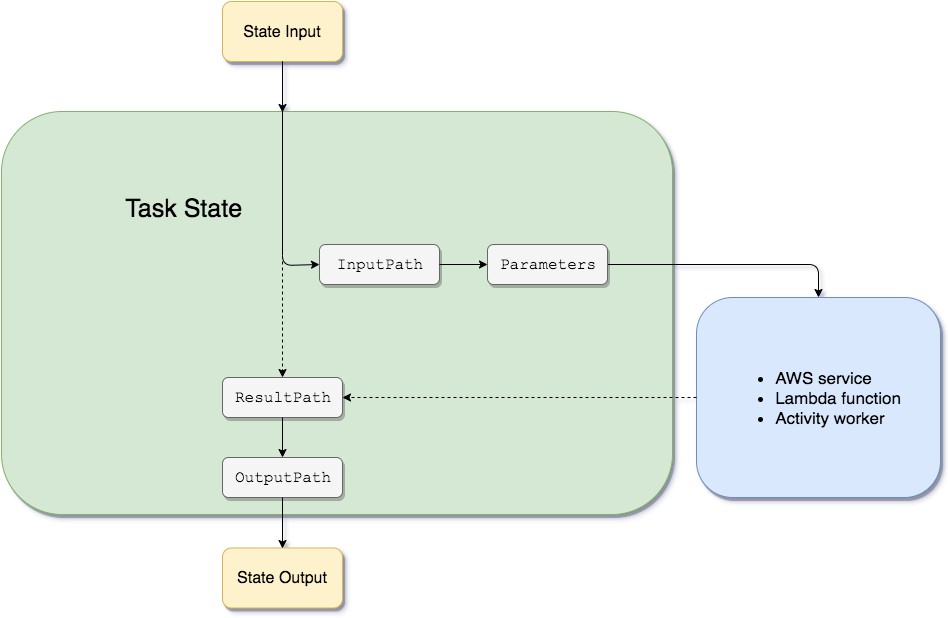
## Input and Output Processing in Step Functions

A Step Functions execution receives a JSON ﬁle as input and passes that input to the ﬁrst state in the workﬂow. Individual states receive JSON as input and usually pass JSON as output to the next state. Understanding how this information ﬂows from state to state, and learning how to ﬁlter and manipulate this data, is key to eﬀectively designing and implementing workﬂows in AWS Step Functions.

In the Amazon States Language, these ﬁelds ﬁlter and control the ﬂow of JSON from state to state:

* InputPath
* OutputPath
* ResultPath
* Parameters

The following diagram shows how JSON information moves through a task state. InputPath selects which parts of the JSON input to pass to the task of the Task state (for example, an AWS Lambda

function). ResultPath then selects what combination of the state input and the task result to pass to the output. OutputPath can ﬁlter the JSON output to further limit the information that is passed to the output.

InputPath, Parameters, ResultPath and OutputPath each manipulate JSON as it moves through each state in your workﬂow.

Each may use [paths (p. 143)](#_bookmark183) to select portions of the JSON from the input or the result. A path is a string, beginning with $, that identiﬁes nodes within JSON text. Step Functions paths use [JsonPath](https://github.com/json-path/JsonPath) syntax.

Topics

* + [InputPath and Parameters (p. 95)](#_bookmark129)
  + [ResultPath (p. 97)](#_bookmark130)
  + [OutputPath (p. 102)](#_bookmark135)
  + [InputPath, ResultPath and OutputPath Example (p. 102)](#_bookmark136)

InputPath and Parameters

Both InputPath and Parameters provide a way to manipulate JSON as it moves through your workﬂow. InputPath can limit the input that is passed by ﬁltering the JSON using a path (see,

[Paths (p. 143)](#_bookmark183)). Parameters allow you to pass a collection of key-value pairs, where the values are either static values that you deﬁne in your state machine deﬁnition, or that are selected from the input using a path.

AWS Step Functions applies InputPath ﬁrst, and then Parameters. You can ﬁrst ﬁlter your raw input to a selection you want using InputPath, and then apply Parameters to manipulate that input further, or add new values.

#### InputPath

Use InputPath to select a portion of the state input. For instance, if input to your state includes the following:

{

"comment": "Example for InputPath.", "dataset1": {

"val1": 1,

"val2": 2,

"val3": 3,

},

"dataset2": { "val1": "a",

"val2": "b",

"val3": "c"

}

}

You could apply the InputPath:

"InputPath": "$.dataset2",

With the above InputPath, the JSON that is passed as the input is:

{

"val1": "a",

"val2": "b",

"val3": "c"

}

#### Parameters

Use Parameters to create a collection of key-value pairs that will be passed as input. The values of each can either be static values that you include in your state machine deﬁnition, or selected from the input with a path. For key-value pairs where the value is selected using a path, the key name must end in \*.$. For instance, given the following input:

{

"comment": "Example for Parameters.", "product": {

"details": { "color": "blue",

"size": "small", "material": "cotton",

},

"availability": "in stock", "sku": "2317",

"cost": "$23"

}

}

To select some of the information you could specify these Parameters in your state machine deﬁnition:

"Parameters": {

"comment": "Selecting what I care about.", "MyDetails": {

"size.$": "$.product.details.size", "exists.$": "$.product.availability", "StaticValue": "foo"

}

},

Given the above previous input and speciﬁed Parameters, this is the JSON that is passed:

{

"comment": "Selecting what I care about.", "MyDetails": {

"size": "small",

"exists": "in stock", "StaticValue": "foo"

}

},

###### Note

Parameters can also pass information to connected resources. For instance, if your task state is orchestrating an AWS Batch job, you can pass the relevant API parameters directly to the API actions of that service. For more information see:

* + - [Pass Parameters to a Service API (p. 81)](#_bookmark110)
    - [Service Integrations (p. 80)](#_bookmark107)

### ResultPath

The output of a state can be a copy of its input, the result it produces (for example, output from a Task state’s Lambda function), or a combination of its input and result. Use ResultPath to control which combination of these is passed to the state output.

The following state types can generate a result and can include ResultPath:

* [Pass (p. 131)](#_bookmark170)
* [Task (p. 132)](#_bookmark171)
* [Parallel (p. 140)](#_bookmark180)

Use ResultPath to combine a task result with task input, or to select one of these. The path you provide to ResultPath controls what information passes to the output.

###### Note

ResultPath is limited to using [reference paths (p. 143)](#_bookmark184), which limit scope so that it can identify only a single node in JSON. See [Reference Paths (p. 143)](#_bookmark184) in the [Amazon States](#_bookmark165) [Language (p. 128)](#_bookmark165).

These examples are based on the state machine and Lambda function described in the [Creating a](#_bookmark38) [Lambda State Machine (p. 18)](#_bookmark38) tutorial. Work through that tutorial and test diﬀerent outputs by trying various paths in a ResultPath ﬁeld.

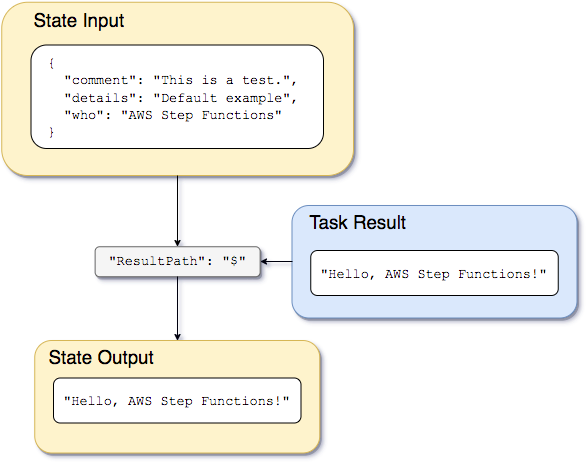
Use ResultPath to:

* + [Use ResultPath to Replace Input with Result (p. 98)](#_bookmark131)
  + [Use ResultPath to Include Result with Input (p. 99)](#_bookmark132)
  + [Use ResultPath to Update a Node in Input with the Result (p. 100)](#_bookmark133)
  + [Use ResultPath to Include Both Error and Input in a Catch (p. 102)](#_bookmark134)

Use ResultPath to Replace Input with Result

If you don't specify a ResultPath, the default behavior is as if you had speciﬁed "ResultPath": "$". Because this tells the state to replace the entire input with the result, the state input is completely replaced by the result coming from the task result.

The following diagram shows how ResultPath can completely replace the input with the result of the task.



Using the state machine and Lambda function described in [Creating a Lambda State Machine (p. 18)](#_bookmark38), if we pass the following input:

{

"comment": "This is a test of the input and output of a Task state.", "details": "Default example",

"who": "AWS Step Functions"

}

The Lambda function provides the following result:

"Hello, AWS Step Functions!"

If ResultPath isn't speciﬁed in the state, or if "ResultPath": "$" is set, the input of the state is replaced by the result of the Lambda function, and the output of the state is:

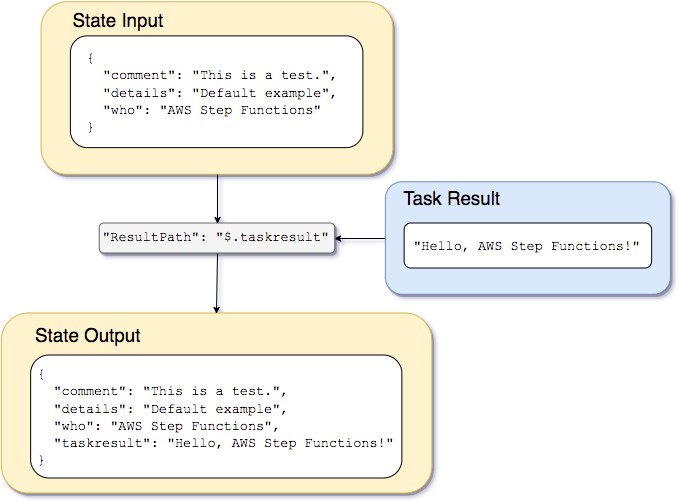
"Hello, AWS Step Functions!"

###### Note

ResultPath is used to include content from the result with the input, before passing it to the output. But, if ResultPath isn't speciﬁed, the default is to replace the entire input.

#### Use ResultPath to Include Result with Input

The following diagram shows how ResultPath can include the result with the input.



Using the state machine and Lambda function described in the [Creating a Lambda State Machine (p. 18)](#_bookmark38) tutorial, we could pass the following input:

{

"comment": "This is a test of the input and output of a Task state.", "details": "Default example",

"who": "AWS Step Functions"

}

The result of the Lambda function is:

"Hello, AWS Step Functions!"

If we want to preserve the input, insert the result of the Lambda function, and then pass the combined JSON to the next state, we could set ResultPath to:

"ResultPath": "$.taskresult"

This includes the result of the Lambda function with the original input:

{

"comment": "This is a test of input and output of a Task state.", "details": "Default behavior example",

"who": "AWS Step Functions",

"taskresult": "Hello, AWS Step Functions!"

}

The output of the Lambda function is appended to the original input as a value for taskresult. The input, including the newly inserted value, is passed to the next state.

You can also insert the result into a child node of the input. Set the ResultPath to:

"ResultPath": "$.strings.lambdaresult"

Start an execution using the following input:

{

"comment": "An input comment.", "strings": {

"string1": "foo",

"string2": "bar",

"string3": "baz"

},

"who": "AWS Step Functions"

}

The result of the Lambda function is inserted as a child of the strings node in the input:

{

"comment": "An input comment.", "strings": {

"string1": "foo",

"string2": "bar",

"string3": "baz",

"lambdaresult": "Hello, AWS Step Functions!"

},

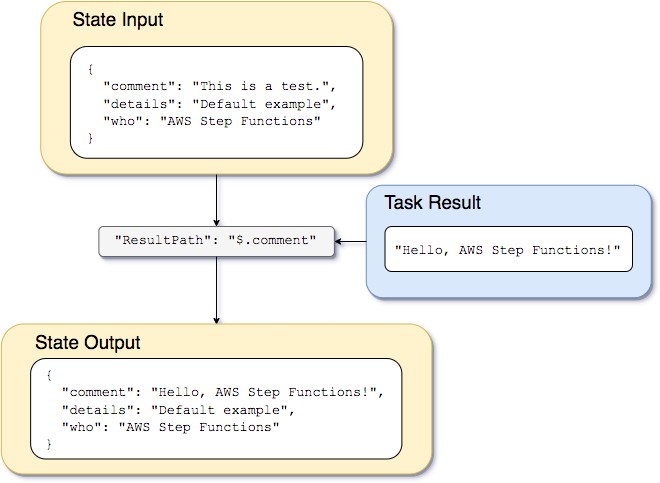
"who": "AWS Step Functions"

}

The state output now includes the original input JSON with the result as a child node.

#### Use ResultPath to Update a Node in Input with the Result

The following diagram shows how ResultPath can update the value of existing JSON nodes in the input with values from the task result.



Using the example of the state machine and Lambda function described in the [Creating a Lambda State](#_bookmark38) [Machine (p. 18)](#_bookmark38) tutorial, we could pass the following input:

{

"comment": "This is a test of the input and output of a Task state.", "details": "Default example",

"who": "AWS Step Functions"

}

The result of the Lambda function is:

Hello, AWS Step Functions!

Instead of preserving the input and inserting the result as a new node in the JSON, we can overwrite an existing node. For example, just as omitting or setting "ResultPath": "$" overwrites the entire node, you can specify an individual node to overwrite with the result:

"ResultPath": "$.comment"

Because the comment node already exists in the state input, setting ResultPath to "$.comment" replaces that node in the input with the result of the Lambda function. Without further ﬁltering by OutputPath, the following is passed to the output:

{

"comment": "Hello, AWS Step Functions!", "details": "Default behavior example", "who": "AWS Step Functions",

}

OutputPath

The value for the comment node, "This is a test of the input and output of a Task state.", is replaced by the result of the Lambda function: "Hello, AWS Step Functions!" in the state output.

#### Use ResultPath to Include Both Error and Input in a Catch

The [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) tutorial shows how to use a state machine to catch an error. In some cases, you might want to preserve the original input with the error. Use ResultPath in a Catch to include the error with the original input, rather than replace it:

"Catch": [{

"ErrorEquals": ["States.ALL"], "Next": "NextTask", "ResultPath": "$.error"

}]

If the previous Catch statement catches an error, it includes the result in an error node within the state input. For example, with the following input:

{"foo": "bar"}

The state output when catching an error is:

{

"foo": "bar", "error": {

"Error": "*Error here*"

}

}

For more information about error handling, see:

* [Error Handling (p. 104)](#_bookmark138)
* [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58)

### OutputPath

OutputPath allows you to select a portion of the state output to pass to the next state. This allows you to ﬁlter out unwanted information, and pass only the portion of JSON that you care about.

If you do not specify an OutputPath the default value is $, which passes the entire JSON node (determined by the state input, the task result, and ResultPath) to the next state.

For more information, see:

* [Paths in the Amazon States Language (p. 143)](#_bookmark183)
* [InputPath, ResultPath and OutputPath Example (p. 102)](#_bookmark136)
* [Pass Static JSON as Parameters (p. 81)](#_bookmark111)
* [Input and Output Processing in Step Functions (p. 94)](#_bookmark128)

### InputPath, ResultPath and OutputPath Example

Any state other than a Fail state can include InputPath, ResultPath or OutputPath. These allow you to use a path to ﬁlter the JSON as it moves through your workﬂow.

InputPath, ResultPath and OutputPath Example

For example, start with the AWS Lambda function and state machine described in the [Creating a Lambda](#_bookmark38) [State Machine (p. 18)](#_bookmark38) tutorial. Modify the state machine so that it includes the following InputPath, ResultPath, and OutputPath:

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "*arn:aws:lambda:us-east-1:123456789012:function:HelloFunction*", "InputPath": *"$.lambda",*

"ResultPath": *"$.data.lambdaresult",*

"OutputPath": *"$.data",*

"End": true

}

}

}

Start an execution using the following input:

{

"comment": "An input comment.", "data": {

"val1": 23,

"val2": 17

},

"extra": "foo", "lambda": {

"who": "AWS Step Functions"

}

}

Assume that the comment and extra nodes can be discarded, but that we want to include the output of the Lambda function, as well as preserve the information in the data node.

In the updated state machine, the Task state is altered to process the input to the task:

"InputPath": "$.lambda",

This line in the state machine deﬁnition limits the task input to only the lambda node from the state input. The Lambda function receives only the JSON object {"who": "AWS Step Functions"} as input.

"ResultPath": "$.data.lambdaresult",

This ResultPathtells the state machine to insert the result of the Lambda function into a node named lambdaresult, as a child of the data node in the original state machine input. Without further processing with OutputPath, the input of the state now includes the result of the Lambda function with the original input:

{

"comment": "An input comment.",

"data": {

"val1": 23,

"val2": 17,

"lambdaresult": "Hello, AWS Step Functions!"

Executions

},

"extra": "foo", "lambda": {

"who": "AWS Step Functions"

}

}

But, our goal was to preserve only the data node, and include the result of the Lambda function.

OutputPath ﬁlters this combined JSON before passing it to the state output:

"OutputPath": "$.data",

This selects only the data node from the original input (including the lambdaresult child inserted by

ResultPath) to be passed to the output. The state output is ﬁltered to:

{

"val1": 23,

"val2": 17,

"lambdaresult": "Hello, AWS Step Functions!"

}

In this Task state:

1. InputPath sends only the lambda node from the input to the Lambda function.
2. ResultPath inserts the result as a child of the data node in the original input.
3. OutputPath ﬁlters the state input (which now includes the result of the Lambda function) so that it passes only the data node to the state output.

For more information, see: [Input and Output Processing in Step Functions (p. 94)](#_bookmark128).

## Executions

A state machine *execution* occurs when a Step Functions state machine runs and performs its tasks. Each Step Functions state machine can have multiple simultaneous executions which you can initiate from the Step Functions console, or using the AWS SDKs, the Step Functions API actions, or the AWS CLI. An execution receives JSON input and produces JSON output.

For more information about the diﬀerent ways of working with Step Functions, see [Development](#_bookmark30) [Options (p. 16)](#_bookmark30). For more information about initiating an execution from the Step Functions console, see [To start a new execution (p. 13)](#_bookmark25).

## Error Handling

Any state can encounter runtime errors. Errors can happen for various reasons:

* State machine deﬁnition issues (for example, no matching rule in a Choice state).
* Task failures (for example, an exception in a Lambda function).
* Transient issues (for example, network partition events).

By default, when a state reports an error, Step Functions causes the execution to fail entirely.

### Error Names

Step Functions identiﬁes errors in Amazon States Language using case-sensitive strings, known as *error names*. Amazon States Language deﬁnes a set of built-in strings that name well-known errors, all beginning with the States. preﬁx.

States.ALL

A wildcard that matches any known error name.

States.Timeout

A Task state either ran longer than the TimeoutSeconds value, or failed to send a heartbeat for a period longer than the HeartbeatSeconds value.

States.TaskFailed

A Task state failed during the execution.

States.Permissions

A Task state failed because it had insuﬃcient privileges to execute the speciﬁed code.

States can report errors with other names. However, these must not begin with the States. preﬁx.

As a best practice, ensure production code can handle Lambda service exceptions (Lambda.ServiceException and Lambda.SdkclientException). For more information see [Handle](#_bookmark193) [Lambda Service Exceptions (p. 151)](#_bookmark193).

**Note**

Unhandled errors in Lambda are reported as Lambda.Unknown in the error output. These include out-of-memory errors, function timeouts, and hitting the concurrent Lambda invoke limit. You can match on Lambda.Unknown, States.ALL, or States.TaskFailed to handle these errors. For more information about Lambda Handled and Unhandled errors, see FunctionError in the [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html#API_Invoke_ResponseSyntax).

### Retrying After an Error

Task and Parallel states can have a ﬁeld named Retry, whose value must be an array of objects known as *retriers*. An individual retrier represents a certain number of retries, usually at increasing time intervals.

###### Note

Retries are treated as state transitions. For information on how state transitions aﬀect billing, see [Step Functions Pricing](https://aws.amazon.com/step-functions/pricing/).

A retrier contains the following ﬁelds:

ErrorEquals (Required)

A non-empty array of strings that match error names. When a state reports an error, Step Functions scans through the retriers. When the error name appears in this array, it implements the retry policy described in this retrier.

IntervalSeconds (Optional)

An integer that represents the number of seconds before the ﬁrst retry attempt (1 by default).

MaxAttempts (Optional)

A positive integer that represents the maximum number of retry attempts (3 by default). If the error recurs more times than speciﬁed, retries cease and normal error handling resumes. A value of 0 speciﬁes that the error or errors are never retried.

BackoffRate (Optional)

The multiplier by which the retry interval increases during each attempt (2.0 by default).

This example of a Retry makes 2 retry attempts after waiting for 3 and 4.5 seconds.

"Retry": [ {

"ErrorEquals": [ "States.Timeout" ], "IntervalSeconds": 3,

"MaxAttempts": 2,

"BackoffRate": 1.5

} ]

The reserved name States.ALL that appears in a Retrier's ErrorEquals ﬁeld is a wildcard that matches any error name. It must appear alone in the ErrorEquals array and must appear in the last retrier in the Retry array.

This example of a Retry ﬁeld retries any error except States.Timeout.

"Retry": [ {

"ErrorEquals": [ "States.Timeout" ], "MaxAttempts": 0

}, {

"ErrorEquals": [ "States.ALL" ]

} ]

Complex Retry Scenarios

A retrier's parameters apply across all visits to the retrier in the context of a single-state execution. Consider the following Task state:

"X": {

"Type": "Task",

"Resource": "arn:aws:states:us-east-1:123456789012:task:X", "Next": "Y",

"Retry": [ {

"ErrorEquals": [ "ErrorA", "ErrorB" ], "IntervalSeconds": 1,

"BackoffRate": 2.0,

"MaxAttempts": 2

}, {

"ErrorEquals": [ "ErrorC" ], "IntervalSeconds": 5

} ],

"Catch": [ {

"ErrorEquals": [ "States.ALL" ], "Next": "Z"

} ]

}

This task fails ﬁve times in succession, outputting these error names: ErrorA, ErrorB, ErrorC, ErrorB, and ErrorB. The following occurs as a result:

* The ﬁrst two errors match the ﬁrst retrier and cause waits of 1 and 2 seconds.
* The third error matches the second retrier and causes a wait of 5 seconds.
* The fourth error matches the ﬁrst retrier and causes a wait of 4 seconds.
* The ﬁfth error also matches the ﬁrst retrier. However, it has already reached its limit of two retries (MaxAttempts) for that particular error (ErrorB), so it fails and execution is redirected to the Z state via the Catch ﬁeld.

### Fallback States

Task and Parallel states can have a ﬁeld named Catch. This ﬁeld's value must be an array of objects, known as *catchers*.

A catcher contains the following ﬁelds:

ErrorEquals (Required)

A non-empty array of Strings that match error names, speciﬁed exactly as they are with the retrier ﬁeld of the same name.

Next (Required)

A string that must exactly match one of the state machine's state names.

ResultPath (Optional)

A [path (p. 143)](#_bookmark182) that determines what input is sent to the state speciﬁed in the Next ﬁeld.

When a state reports an error and either there is no Retry ﬁeld, or if retries fail to resolve the error, Step Functions scans through the catchers in the order listed in the array. When the error name appears in the value of a catcher's ErrorEquals ﬁeld, the state machine transitions to the state named in the Next ﬁeld.

The reserved name States.ALL that appears in a catcher's ErrorEquals ﬁeld is a wildcard that matches any error name. It must appear alone in the ErrorEquals array and must appear in the last catcher in the Catch array.

The following example of a Catch ﬁeld transitions to the state named RecoveryState when a Lambda function outputs an unhandled Java exception. Otherwise, the ﬁeld transitions to the EndState state:

"Catch": [ {

"ErrorEquals": [ "java.lang.Exception" ], "ResultPath": "$.error-info",

"Next": "RecoveryState"

}, {

"ErrorEquals": [ "States.ALL" ], "Next": "EndState"

} ]

**Note**

Each catcher can specify multiple errors to handle.

Error Output

When Step Functions transitions to the state speciﬁed in a catch name, the object usually contains the ﬁeld Cause. This ﬁeld's value is a human-readable description of the error. This object is known as the *error output*.

In this example, the ﬁrst catcher contains a ResultPath ﬁeld. This works similarly to a ResultPath

ﬁeld in a state's top level, resulting in two possibilities:

* It takes the results of executing the state and overwrites a portion of the state's input (or all of the state's input).
* It takes the results and adds them to the input. In the case of an error handled by a catcher, the result of executing the state is the error output.

Thus, in this example, for the ﬁrst catcher the error output is added to the input as a ﬁeld named

error-info (if there isn't already a ﬁeld with this name in the input). Then, the entire input is sent to RecoveryState. For the second catcher, the error output overwrites the input and only the error output is sent to EndState.

###### Note

If you don't specify the ResultPath ﬁeld, it defaults to $, which selects and overwrites the entire input.

When a state has both Retry and Catch ﬁelds, Step Functions uses any appropriate retriers ﬁrst, and only afterward applies the matching catcher transition if the retry policy fails to resolve the error.

### Examples Using Retry and Using Catch

The state machines deﬁned in the following examples assume the existence of two Lambda functions: one that always fails and one that waits long enough to allow a timeout deﬁned in the state machine to occur.

This is a deﬁnition of a Lambda function that always fails, returning the message error. In the state machine examples that follow, this Lambda function is named FailFunction.

exports.handler = (event, context, callback) => { callback("error");

};

This is a deﬁnition of a Lambda function that sleeps for 10 seconds. In the state machine examples that follow, this Lambda function is named sleep10.

###### Note

When you create this Lambda function in the Lambda console, remember to change the

**Timeout** value in the **Advanced settings** section from 3 seconds (default) to 11 seconds.

exports.handler = (event, context, callback) => { setTimeout(function(){

}, 11000);

};

Handling a Failure Using Retry

This state machine uses a Retry ﬁeld to retry a function that fails and outputs the error name

HandledError. The function is retried twice with an exponential backoﬀ between retries.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda

function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction", "Retry": [ {

"ErrorEquals": ["HandledError"], "IntervalSeconds": 1,

"MaxAttempts": 2,

"BackoffRate": 2.0

} ],

"End": true

}

}

}

This variant uses the predeﬁned error code States.TaskFailed, which matches any error that a Lambda function outputs.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction", "Retry": [ {

"ErrorEquals": ["States.TaskFailed"], "IntervalSeconds": 1,

"MaxAttempts": 2,

"BackoffRate": 2.0

} ],

"End": true

}

}

}

###### Note

As a best practice, tasks that reference a Lambda function should handle Lambda service exceptions. For more information see [Handle Lambda Service Exceptions (p. 151)](#_bookmark193).

#### Handling a Failure Using Catch

This example uses a Catch ﬁeld. When a Lambda function outputs an error, the error is caught and the state machine transitions to the fallback state.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction", "Catch": [ {

"ErrorEquals": ["HandledError"], "Next": "fallback"

} ],

"End": true

},

"fallback": { "Type": "Pass",

"Result": "Hello, AWS Step Functions!", "End": true

}

}

}

This variant uses the predeﬁned error code States.TaskFailed, which matches any error that a Lambda function outputs.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction", "Catch": [ {

"ErrorEquals": ["States.TaskFailed"], "Next": "fallback"

} ],

"End": true

},

"fallback": { "Type": "Pass",

"Result": "Hello, AWS Step Functions!", "End": true

}

}

}

#### Handling a Timeout Using Retry

This state machine uses a Retry ﬁeld to retry a function that times out. The function is retried twice with an exponential backoﬀ between retries.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:sleep10", "TimeoutSeconds": 2,

"Retry": [ {

"ErrorEquals": ["States.Timeout"], "IntervalSeconds": 1,

"MaxAttempts": 2,

"BackoffRate": 2.0

} ],

"End": true

}

}

}

#### Handling a Timeout Using Catch

This example uses a Catch ﬁeld. When a timeout occurs, the state machine transitions to the fallback

state.

{

"Comment": "A Hello World example of the Amazon States Language using an AWS Lambda

function",

"StartAt": "HelloWorld", "States": {

"HelloWorld": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:sleep10", "TimeoutSeconds": 2,

"Catch": [ {

"ErrorEquals": ["States.Timeout"],

"Next": "fallback"

} ],

"End": true

},

"fallback": { "Type": "Pass",

"Result": "Hello, AWS Step Functions!", "End": true

}

}

}

###### Note

You can preserve the state input along with the error by using ResultPath. See [Use](#_bookmark134)

[ResultPath to Include Both Error and Input in a Catch (p. 102)](#_bookmark134)

## Read Consistency

State machine updates in AWS Step Functions are eventually consistent. All StartExecution calls within a few seconds will use the updated deﬁnition and roleArn (the Amazon Resource Name for the IAM role). Executions started immediately after calling UpdateStateMachine might use the previous state machine deﬁnition and roleArn.

For more information, see:

* [UpdateStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_UpdateStateMachine.html) — in the AWS Step Functions API Reference
* [Step 3: (Optional) Update a State Machine (p. 14)](#_bookmark26) — in the [Getting Started (p. 12)](#_bookmark21) section

## Templates

In the Step Functions console, you can choose one of the following state machine templates to automatically ﬁll the **Code** pane. Each of the templates is fully functional and you can use any blueprint as the template for your own state machine.

###### Note

Choosing any of the templates overwrites the contents of the **Code** pane.

* **Hello world** – A state machine with a Pass state.
* **Wait state** – A state machine that demonstrates diﬀerent ways of injecting a Wait state into a running state machine:
  + By waiting for a number of seconds.
  + By waiting for an absolute time (timestamp).
  + By specifying the Wait state's deﬁnition.
  + By using the state's input data.
* **Retry failure** – A state machine that retries a task after the task fails. This blueprint demonstrates how to handle multiple retries and various failure types.
* **Parallel** – A state machine that demonstrates how to execute two branches at the same time.
* **Catch failure** – A state machine that performs a diﬀerent task after its primary task fails. This blueprint demonstrates how to call diﬀerent tasks depending on the failure type.
* **Choice state** – A state machine that makes a choice: It either runs a Task state from a set of Task

states or runs a Fail state after the initial state is complete.

## Tagging

AWS Step Functions supports tagging of state machines and activities. This can help you track and manage the costs associated with your resources, and provide better security in your IAM policies.

To review the restrictions related to resource tagging, see [Restrictions Related to Tagging (p. 155)](#_bookmark202).

Topics

* + [Tagging for Cost Allocation (p. 112)](#_bookmark146)
  + [Viewing and Managing Tags in the Step Functions Console (p. 112)](#_bookmark147)
  + [Manage Tags With Step Functions API Actions. (p. 113)](#_bookmark148)

Tagging for Cost Allocation

To organize and identify your Step Functions resources for cost allocation, you can add metadata *tags* that identify the purpose of a state machine or activity. This is especially useful when you have many resources. You can use cost allocation tags to organize your AWS bill to reﬂect your own cost structure. To do this, sign up to get your AWS account bill to include the tag keys and values. For more information, see [Setting Up a Monthly Cost Allocation Report](https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/configurecostallocreport.html#allocation-report) in the *AWS Billing and Cost Management User Guide*.

For instance, you could add tags that represent the cost center and purpose of your Step Functions resources:

|  |  |  |
| --- | --- | --- |
| **Resource** | **Key** | **Value** |
| StateMachine1 | Cost Center | 34567 |
| Application | Image processing |
| StateMachine2 | Cost Center | 34567 |
| Application | Rekognition processing |
| Activity1 | Cost Center | 12345 |
| Application | Legacy database |

This tagging scheme allows you to group two state machines performing related tasks in the same cost center, while tagging an unrelated activity with a diﬀerent cost allocation tag.

### Viewing and Managing Tags in the Step Functions Console

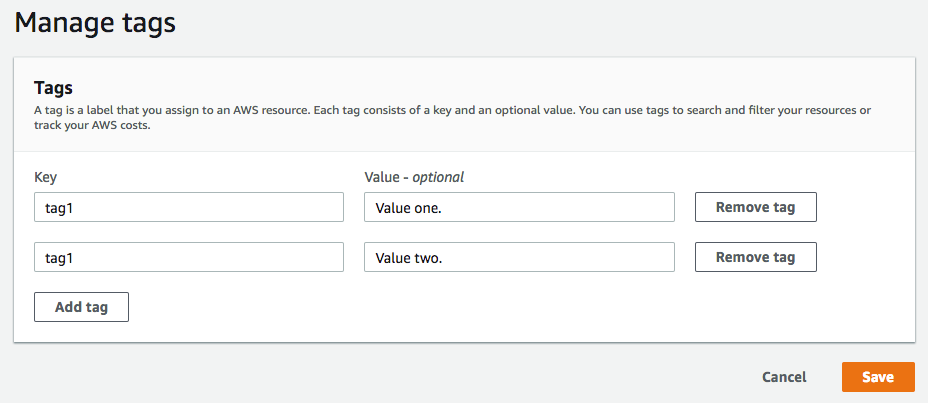
AWS Step Functions allows you to view and manage the tags for your state machines in the Step Functions console. From the **Details** page of a state machine, select **Tags**. Here, you can view the existing tags associated with your state machine.

###### Note

To manage tags for activites, see [Manage Tags With Step Functions API Actions. (p. 113)](#_bookmark148).

To add or delete tags that are associated with your state machine, select the **Manage Tags** button.

1. Browse to the details page of a state machine.
2. Select **Tags**, next to **Executions** and **Deﬁnition**.
3. Choose **Manage tags**.
4. • To modify existing tags, edit the **Key** and **Value**.
   * To remove existing tags, select **Remove tag**.
   * To add a new tag, select **Add tag** and enter a **Key** and **Value**.



1. Choose **Save**.

### Manage Tags With Step Functions API Actions.

To manage tags using the Step Functions API, use the following API actions:

* [ListTagsForResource](https://docs.aws.amazon.com/step-functions/latest/apireference/API_ListTagsForResource.html)
* [TagResource](https://docs.aws.amazon.com/step-functions/latest/apireference/API_TagResource.html)
* [UntagResource](https://docs.aws.amazon.com/step-functions/latest/apireference/API_UntagResource.html)

# Sample Projects

In the Step Functions console, you can choose one of the following state machine sample projects to automatically create the state machine **Code**, **Visual Workﬂow**, and all related AWS resources for the project. Each of the sample projects provisions a fully functional state machine, and creates the related resources for it to run. When you create a sample project, Step Functions uses AWS CloudFormation to create the related resources referenced by the state machine.

Topics

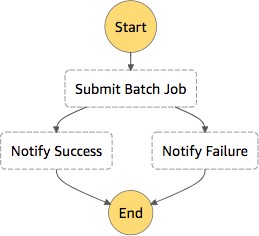
* + [Manage a Batch Job (Batch, SNS) (p. 114)](#_bookmark150)
  + [Manage a Container Task (ECS, SNS) (p. 117)](#_bookmark154)
  + [Transfer Data Records (Lambda, DynamoDB, SQS) (p. 120)](#_bookmark158)
  + [Poll for Job Status (Lambda, Batch), (p. 123)](#_bookmark162)
  + [Task Timer (p. 126)](#_bookmark164)

Manage a Batch Job (Batch, SNS)

This sample project demonstrates how to submit a AWS Batch job, and then send an Amazon SNS notiﬁcation based on whether that job succeeds or fails. Deploying this sample project will create a Step Functions state machine, an AWS Batch job, and an Amazon SNS topic. In this project, Step Functions uses a state machine to call the AWS Batch job synchronously. It then waits for the job to succeed or fail, and it sends an Amazon SNS topic with a message about whether the job succeeded or failed.

To create the **Manage a Batch Job** state machine and provision all resources:

1. Log in to the Step Functions console, and choose **Create a state machine**.
2. Select **Sample Projects** and choose **Manage a Batch Job**. The state machine **Code** and **Visual Workﬂow** are displayed.



1. Select **Next**.

The **Deploy resources** page is displayed, listing the resources that will be created. For this sample project the resources include:

* + An AWS Batch job
  + An Amazon SNS topic

1. Choose **Deploy Resources**.

###### Note

It can take up to 10 minutes as these resources and related IAM permissions are created. While the **Deploy resources** page displays, you can open the **Stack ID** link to see which resources are being provisioned.

To start a new execution

1. On the **New execution** page, enter an execution name (optional) and choose **Start Execution.**
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Optionally, you can go to the newly-created state machine on the Step Functions **Dashboard**, select

New execution.

1. Once an execution is complete, you can select states on the **Visual workﬂow** and browse the **Input**

and **Output** under **Step details**

### Example State machine code

The state machine in this sample project integrates with AWS Batch and Amazon SNS by passing parameters directly to those resources. Browse through this example state machine to see how Step Functions controls AWS Batch; and Amazon SNS by connecting to the ARN in the Resource ﬁeld, and by passing Parameters to the service API.

For more information on how AWS Step Functions can control other AWS services, see: [AWS Service](#_bookmark107) [Integrations (p. 80)](#_bookmark107).

{

"Comment": "An example of the Amazon States Language for notification on an AWS Batch job completion",

"StartAt": "Submit Batch Job", "TimeoutSeconds": 3600, "States": {

"Submit Batch Job": { "Type": "Task",

"Resource": "arn:aws:states:::batch:submitJob.sync", "Parameters": {

"JobName": "BatchJobNotification",

"JobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/ BatchJobQueue-7049d367474b4dd",

"JobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/ BatchJobDefinition-74d55ec34c4643c:1"

},

"Next": "Notify Success", "Catch": [

{

"ErrorEquals": [ "States.ALL" ], "Next": "Notify Failure"

}

]

},

"Notify Success": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"Message": "Batch job submitted through Step Functions succeeded",

"TopicArn": "arn:aws:sns:us-east-1:123456789012:batchjobnotificatiointemplate- SNSTopic-1J757CVBQ2KHM"

},

"End": true

},

"Notify Failure": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"Message": "Batch job submitted through Step Functions failed",

"TopicArn": "arn:aws:sns:us-east-1:123456789012:batchjobnotificatiointemplate- SNSTopic-1J757CVBQ2KHM"

},

"End": true

}

}

}

### IAM Example

This example IAM policy generated by the sample project includes the least privilege necessary to execute the state machine and related resources. It is a best practice to include only those permissions necessary in your IAM policies

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"sns:Publish"

],

"Resource": [

"arn:aws:sns:ap-northeast-1:123456789012:ManageBatchJob-SNSTopic-

JHLYYG7AZPZI"

],

"Effect": "Allow"

},

{

"Action": [

"batch:SubmitJob", "batch:DescribeJobs", "batch:TerminateJob"

],

"Resource": "\*",

"Effect": "Allow"

},

{

"Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [

"arn:aws:events:ap-northeast-1:123456789012:rule/ StepFunctionsGetEventsForBatchJobsRule"

],

"Effect": "Allow"

}

]

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

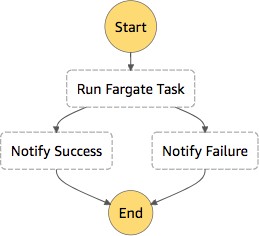
## Manage a Container Task (ECS, SNS)

This sample project demonstrates how to run a Fargate task, and then send an Amazon SNS notiﬁcation based on whether that job succeeds or fails. Deploying this sample project will create a Step Functions state machine, a Fargate Cluster, and an Amazon SNS topic. In this project, Step Functions uses a state machine to call the Fargate task synchronously. It then waits for the task to succeed or fail, and it sends an Amazon SNS topic with a message about whether the job succeeded or failed.

To create the **Manage a container task** state machine and provision all resources:

1. Log in to the Step Functions console, and choose **Create a state machine**.
2. Select **Sample Projects** and choose **Manage a container task**.

The state machine **Code** and **Visual Workﬂow** are displayed.



1. Select **Next**.

The **Deploy resources** page is displayed, listing the resources that will be created. For this sample project the resources include:

* + A Fargate Cluster
  + An Amazon SNS topic

1. Choose **Deploy Resources**.

###### Note

It can take up to 10 minutes as these resources and related IAM permissions are created. While the **Deploy resources** page displays, you can open the **Stack ID** link to see which resources are being provisioned.

To start a new execution

1. On the **New execution** page, enter an execution name (optional) and choose **Start Execution.**
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Optionally, you can go to the newly-created state machine on the Step Functions **Dashboard**, select

New execution.

1. Once an execution is complete, you can select states on the **Visual workﬂow** and browse the **Input**

and **Output** under **Step details**

### Example State machine code

The state machine in this sample project integrates with AWS Batch and Amazon SNS by passing parameters directly to those resources. Browse through this example state machine to see how Step Functions controls AWS Batch; and Amazon SNS by connecting to the ARN in the Resource ﬁeld, and by passing Parameters to the service API.

For more information on how AWS Step Functions can control other AWS services, see: [AWS Service](#_bookmark107) [Integrations (p. 80)](#_bookmark107).

{

"Comment": "An example of the Amazon States Language for notification on an AWS Fargate task completion",

"StartAt": "Run Fargate Task", "TimeoutSeconds": 3600, "States": {

"Run Fargate Task": { "Type": "Task",

"Resource": "arn:aws:states:::ecs:runTask.sync", "Parameters": {

"LaunchType": "FARGATE",

"Cluster": "arn:aws:ecs:ap-northeast-1:123456789012:cluster/ FargateTaskNotification-ECSCluster-VHLR20IF9IMP",

"TaskDefinition": "arn:aws:ecs:ap-northeast-1:123456789012:task-definition/ FargateTaskNotification-ECSTaskDefinition-13YOJT8Z2LY5Q:1",

"NetworkConfiguration": { "AwsvpcConfiguration": {

"Subnets": [

"subnet-07e1ad3abcfce6758", "subnet-04782e7f34ae3efdb"

],

"AssignPublicIp": "ENABLED"

}

}

},

"Next": "Notify Success", "Catch": [

{

"ErrorEquals": [ "States.ALL" ], "Next": "Notify Failure"

}

]

},

"Notify Success": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"Message": "AWS Fargate Task started by Step Functions succeeded", "TopicArn": "arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification-

SNSTopic-1XYW5YD5V0M7C"

},

"End": true

},

"Notify Failure": { "Type": "Task",

"Resource": "arn:aws:states:::sns:publish", "Parameters": {

"Message": "AWS Fargate Task started by Step Functions failed",

"TopicArn": "arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification- SNSTopic-1XYW5YD5V0M7C"

},

"End": true

}

}

}

### IAM Example

This example IAM policy generated by the sample project includes the least privilege necessary to execute the state machine and related resources. It is a best practice to include only those permissions necessary in your IAM policies

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"sns:Publish"

],

"Resource": [

"arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification- SNSTopic-1XYW5YD5V0M7C"

],

"Effect": "Allow"

},

{

"Action": [

"ecs:RunTask"

],

"Resource": [

"arn:aws:ecs:ap-northeast-1:123456789012:task-definition/ FargateTaskNotification-ECSTaskDefinition-13YOJT8Z2LY5Q:1"

],

"Effect": "Allow"

},

{

"Action": [

"ecs:StopTask", "ecs:DescribeTasks"

],

"Resource": "\*",

"Effect": "Allow"

},

{

"Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [

"arn:aws:events:ap-northeast-1:123456789012:rule/ StepFunctionsGetEventsForECSTaskRule"

],

"Effect": "Allow"

}

]

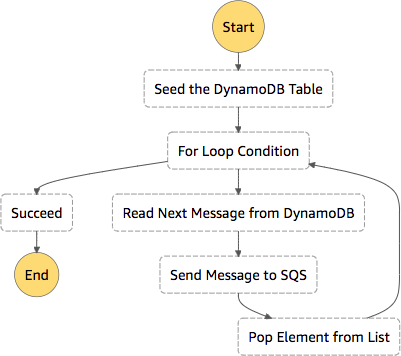
}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

## Transfer Data Records (Lambda, DynamoDB, SQS)

This sample project demonstrates how to read values from a DynamoDB table and send them to Amazon SQS using AWS Step Functions. Deploying this sample project will create a Step Functions state machine, a DynamoDB table, a Lambda function, and an Amazon SQS topic. In this project, Step Functions uses the Lambda function to populate the DynamoDB table and then uses a for loop to read each of the entries, and sends each entry to Amazon SQS.

To create the **Transfer Data Records** state machine and provision all resources:

1. Log in to the Step Functions console, and choose **Create a state machine**.
2. Select **Sample Projects** and choose **Transfer Data Records**. The state machine **Code** and **Visual Workﬂow** are displayed.

###### Note

The **Code** section in this state machine references the AWS resources that will be created for this sample project.

1. Select **Next**.

The **Deploy resources** page is displayed, listing the resources that will be created. For this sample project the resources include:

* + A Lambda function for seeding the DynamoDB table
  + An Amazon SQS queue
  + A DynamoDB table

1. Choose **Deploy Resources**.

###### Note

It can take up to 10 minutes as these resources and related IAM permissions are created. While the **Deploy resources** page displays, you can open the **Stack ID** link to see which resources are being provisioned.

To start a new execution

1. On the **New execution** page, enter an execution name (optional) and choose **Start Execution.**
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Optionally, you can go to the newly-created state machine on the Step Functions **Dashboard**, select

New execution.

1. Once an execution is complete, you can select states on the **Visual workﬂow** and browse the **Input**

and **Output** under **Step details**

### Example State machine code

The state machine in this sample project integrates with DynamoDB and Amazon SQS by passing parameters directly to those resources. Browse through this example state machine to see how Step Functions controls DynamoDB and Amazon SQS by connecting to the ARN in the Resource ﬁeld, and by passing Parameters to the service API.

For more information on how AWS Step Functions can control other AWS services, see: [AWS Service](#_bookmark107) [Integrations (p. 80)](#_bookmark107).

{

"Comment" : "An example of the Amazon States Language for reading messages from a DynamoDB table and sending them to SQS",

"StartAt": "Seed the DynamoDB Table", "TimeoutSeconds": 3600,

"States": {

"Seed the DynamoDB Table": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:sqsconnector- SeedingFunction-T3U43VYDU5OQ",

"ResultPath": "$.List", "Next": "For Loop Condition"

},

"For Loop Condition": { "Type": "Choice", "Choices": [

{

"Not": {

"Variable": "$.List[0]", "StringEquals": "DONE"

},

"Next": "Read Next Message from DynamoDB"

}

],

"Default": "Succeed"

},

"Read Next Message from DynamoDB": { "Type": "Task",

"Resource": "arn:aws:states:::dynamodb:getItem", "Parameters": {

"TableName": "sqsconnector-DDBTable-1CAFOJWP8QD6I", "Key": {

"MessageId": {"S.$": "$.List[0]"}

}

},

"ResultPath": "$.DynamoDB", "Next": "Send Message to SQS"

},

"Send Message to SQS": { "Type": "Task",

"Resource": "arn:aws:states:::sqs:sendMessage", "Parameters": {

"MessageBody.$": "$.DynamoDB.Item.Message.S",

"QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/sqsconnector- SQSQueue-QVGQBW134PWK"

},

"ResultPath": "$.SQS",

"Next": "Pop Element from List"

},

"Pop Element from List": { "Type": "Pass", "Parameters": {

"List.$": "$.List[1:]"

},

"Next": "For Loop Condition"

},

"Succeed": {

"Type": "Succeed"

}

}

}

For more information on passing parameters and managing results, see:

* [Pass Parameters to a Service API (p. 81)](#_bookmark110)
* [ResultPath (p. 97)](#_bookmark130)

### IAM Example

This example IAM policy generated by the sample project includes the least privilege necessary to execute the state machine and related resources. It is a best practice to include only those permissions necessary in your IAM policies

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"dynamodb:GetItem"

],

"Resource": [

"arn:aws:dynamodb:ap-northeast-1:123456789012:table/TransferDataRecords- DDBTable-3I41R5L5EAGT"

],

"Effect": "Allow"

},

{

"Action": [

"sqs:SendMessage"

],

"Resource": [

"arn:aws:sqs:ap-northeast-1:123456789012:TransferDataRecords-SQSQueue-

BKWXTS09LIW1"

],

"Effect": "Allow"

},

{

"Action": [

"lambda:invokeFunction"

],

"Resource": [

"arn:aws:lambda:ap-northeast-1:123456789012:function:TransferDataRecords- SeedingFunction-VN4KY2TPAZSR"

],

"Effect": "Allow"

}

]

}

For information on how to conﬁgure IAM when using Step Functions with other AWS services, see [IAM](#_bookmark224) [Policies for Integrated Services (p. 174)](#_bookmark224).

## Poll for Job Status (Lambda, Batch),

This sample project creates an AWS Batch job status poller. It implements an AWS Step Functions state machine that uses AWS Lambda to create a Wait state loop that checks on an AWS Batch job. This sample project creates and conﬁgures all resources so that your Step Functions workﬂow will submit an AWS Batch job, and will wait for that job to complete before ending successfully.

###### Note

You can also implement this pattern without using a Lambda function. For information on controlling AWS Batch directly, see [AWS Service Integrations (p. 80)](#_bookmark107).

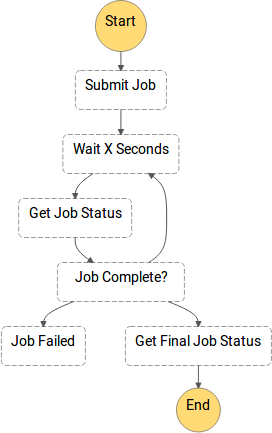
This sample project creates the state machine, two Lambda functions, an AWS Batch queue, and conﬁgures the related IAM permissions. For more information on the resources that are created with the **Job Status Poller** sample project, see:

* [AWS CloudFormation User Guide](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/)
* [AWS Batch User Guide](https://docs.aws.amazon.com/batch/latest/userguide/)
* [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/)
* [IAM Getting Started Guide](https://docs.aws.amazon.com/IAM/latest/GettingStartedGuide/)

To create the **Job Status Poller** state machine and provision all resources:

1. Log in to the Step Functions console, and choose **Create a state machine**.
2. Select **Sample Projects** and choose **Job Status Poller**.

The state machine **Code** and **Visual Workﬂow** are displayed.



###### Note

The **Code** section in this state machine references the AWS resources that will be created for this sample project.

1. Choose **Create Resources**.

The **Create Project Resources** window is displayed, listing the resources that will be created. For this sample project the resources include:

* + A SubmitJob Lambda function
  + A CheckJob Lambda function
  + A SampleJobQueue Batch Job Queue

###### Note

It can take up to 10 minutes as these resources and related IAM permissions are created. While the **Create Project Resources** window displays **Creating resources**, you can open the **Stack ID** link to see which resources are being provisioned.

Once complete, the **New execution** window is displayed, with example input similar to this:

{

"jobName": "my-job",

"jobDefinition": "arn:aws:batch:us-east-2:123456789012:job-definition/ SampleJobDefinition-343f54b445d5312:1",

"jobQueue": "arn:aws:batch:us-east-2:123456789012:job-queue/ SampleJobQueue-4d9d696031e1449",

"wait\_time": 60

}

Starting an Execution

After you create your state machine, you can start an execution.

#### To start a new execution

1. On the **New execution** page, enter an execution name (optional) and choose **Start Execution.**
2. (Optional) To help identify your execution, you can specify an ID for it in the **Enter an execution name** box. If you don't enter an ID, Step Functions generates a unique ID automatically.

###### Note

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

1. Optionally, you can go to the newly-created state machine on the Step Functions **Dashboard**, select **New execution**, and enter the input code using the names or Amazon Resource Names of your newly created resources.

For instance, the input for the above execution using only the resource names would be:

{

"jobName": "my-job",

"jobDefinition": "SampleJobDefinition-343f54b445d5312", "jobQueue": "SampleJobQueue-4d9d696031e1449", "wait\_time": 60

}

###### Note

wait\_time instructs the Wait state to loop every sixty seconds.

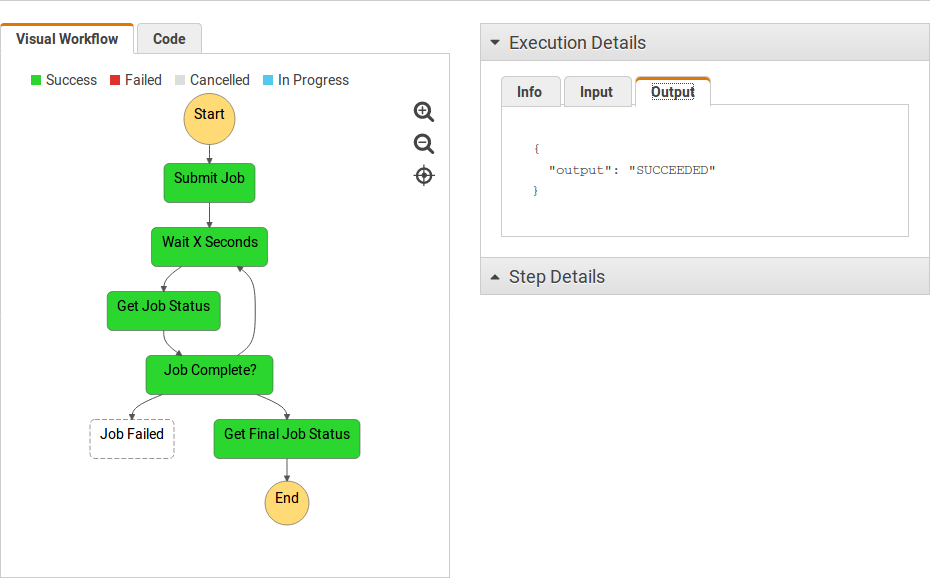
1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. (Optional) In the **Execution Details** section, choose **Info** to view the **Execution Status** and the

**Started** and **Closed** timestamps.

1. To view the changing status of your AWS Batch job and the looping results of your execution, choose

Output.

Task Timer

**Note**

This sample project implements an Lambda function to send an Amazon SNS notiﬁcation. You can also send an Amazon SNS notiﬁcation directly from the Amazon States Language. See, [AWS](#_bookmark107) [Service Integrations (p. 80)](#_bookmark107).

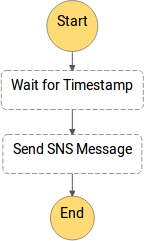
This sample project creates a task timer. It implements an AWS Step Functions state machine that implements a wait state, and uses a Lambda function to that sends an Amazon Simple Notiﬁcation Service notiﬁcation. A Wait state is a state type that waits for a trigger to perform a single unit of work.

This sample project creates the state machine, a Lambda function, an Amazon SNS topic, and conﬁgures the related IAM permissions. For more information on the resources that are created with the **Task Timer** sample project, see:

* [AWS CloudFormation User Guide](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/)
* [Amazon Simple Notiﬁcation Service Developer Guide](https://docs.aws.amazon.com/sns/latest/dg/)
* [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/)
* [IAM Getting Started Guide](https://docs.aws.amazon.com/IAM/latest/GettingStartedGuide/)

To create the **Task Timer** state machine and provision all resources:

1. Log in to the Step Functions console, and choose **Create a state machine**.
2. Select **Sample Projects** and choose **Task Timer**.

The state machine **Code** and **Visual Workﬂow** are displayed.

###### Note

The **Code** section in this state machine references the AWS resources that will be created for this sample project.

1. Choose **Create Sample Project**.

The **Create Project Resources** window is displayed, listing the resources that will be created. For this sample project the resources include:

* + A SendToSNS Lambda function
  + A TaskTimerTopic Amazon SNS topic

###### Note

It can take up to 10 minutes as these resources and related IAM permissions are created. While the **Create Project Resources** window displays **Creating resources**, you can open the **Stack ID:** link to see which resources are being provisioned.

Once complete, the **New execution** window is displayed, with example input similar to this:

{

"topic": "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-TaskTimer-517b8680- e0ad-07cf-feee-65aa5fc63ac0-SNSTopic-96RHT77RAKTS",

"message": "HelloWorld", "timer\_seconds": 10

}

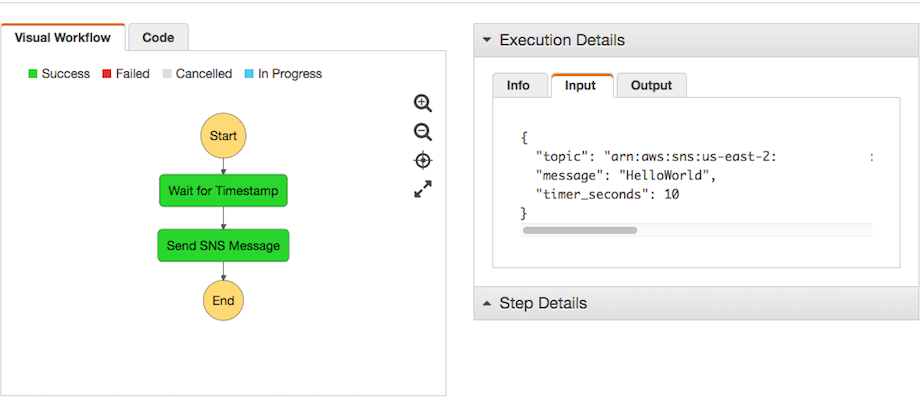
1. Choose **Start Execution**.

A new execution of your state machine starts, and a new page showing your running execution is displayed.

1. (Optional) In the **Execution Details** section, choose **Info** to view the **Execution Status** and the

**Started** and **Closed** timestamps.

1. To view the changing status of your AWS Batch job and the looping results of your execution, choose

Output.

Amazon States Language

Amazon States Language is a JSON-based, structured language used to deﬁne your state machine, a collection of [states (p. 71)](#_bookmark99), that can do work (Task states), determine which states to transition to next (Choice states), stop an execution with an error (Fail states), and so on. For more information, see the [Amazon States Language Speciﬁcation](https://states-language.net/spec.html) and [Statelint](https://github.com/awslabs/statelint), a tool that validates Amazon States Language code.

To create a state machine on the Step Functions console using Amazon States Language, see [Getting](#_bookmark21) [Started (p. 12)](#_bookmark21).

Topics

* + [Example Amazon States Language Speciﬁcation (p. 128)](#_bookmark166)
  + [State Machine Structure (p. 129)](#_bookmark167)
  + [States (p. 130)](#_bookmark168)
  + [Input and Output Processing (p. 143)](#_bookmark182)
  + [Errors (p. 146)](#_bookmark185)

## Example Amazon States Language Speciﬁcation

{

"Comment": "An example of the Amazon States Language using a choice state.",

"StartAt": "FirstState", "States": {

"FirstState": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION\_NAME", "Next": "ChoiceState"

},

"ChoiceState": { "Type" : "Choice", "Choices": [

{

"Variable": "$.foo", "NumericEquals": 1, "Next": "FirstMatchState"

},

{

"Variable": "$.foo", "NumericEquals": 2, "Next": "SecondMatchState"

}

],

"Default": "DefaultState"

},

"FirstMatchState": { "Type" : "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnFirstMatch", "Next": "NextState"

},

"SecondMatchState": {

"Type" : "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnSecondMatch", "Next": "NextState"

},

"DefaultState": { "Type": "Fail",

"Error": "DefaultStateError", "Cause": "No Matches!"

},

"NextState": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION\_NAME", "End": true

}

}

}

State Machine Structure

State machines are deﬁned using JSON text that represents a structure containing the following ﬁelds:

Comment (Optional)

A human-readable description of the state machine.

StartAt (Required)

A string that must exactly match (case-sensitive) the name of one of the state objects.

TimeoutSeconds (Optional)

The maximum number of seconds an execution of the state machine may run; if it runs longer than the speciﬁed time, then the execution fails with an States.Timeout [Error name (p. 146)](#_bookmark186).

Version (Optional)

The version of Amazon States Language used in the state machine, default is "1.0".

States (Required)

This ﬁeld's value is an object containing a comma-delimited set of states.

The States ﬁeld contains a number of [States (p. 130)](#_bookmark168):

{

"State1" : {

},

"State2" : {

},

...

}

A state machine is deﬁned by the states it contains and the relationships between them. Here's an example:

{

"Comment": "A Hello World example of the Amazon States Language using a Pass state", "StartAt": "HelloWorld",

"States": { "HelloWorld": {

"Type": "Pass",

"Result": "Hello World!", "End": true

}

}

}

When an execution of this state machine is launched, the system begins with the state referenced in the StartAt ﬁeld ("HelloWorld"). If this state has an "End": true ﬁeld, the execution stops and returns a result. Otherwise, the system looks for a "Next": ﬁeld and continues with that state next. This process repeats until the system reaches a terminal state (a state with "Type": "Succeed", "Type": "Fail", or "End": true), or a runtime error occurs.

The following rules apply to states within a state machine:

* States can occur in any order within the enclosing block, but the order in which they're listed doesn't aﬀect the order in which they're run, which is determined by the contents of the states themselves.
* Within a state machine, there can be only one state that's designated as the start state, designated by the value of the StartAt ﬁeld in the top-level structure. This state is the one that is executed ﬁrst when the execution starts.
* Any state for which the End ﬁeld is true is considered to be an end (or terminal) state. Depending on your state machine logic—for example, if your state machine has multiple branches of execution— you may have more than one end state.
* If your state machine consists of only one state, it can be both the start state and the end state.

## States

States are top-level elements within a state machine's States ﬁeld, and can take a number of diﬀerent roles in your state machine depending on their type.

"FirstState" : { "Type" : "Task",

...

}

States are identiﬁed by their name, which must be unique within the state machine speciﬁcation, but otherwise can be any valid string in JSON text format. Each state also contains a number of ﬁelds with options that vary according to the contents of the state's required Type ﬁeld.

###### Note

State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:

* + Whitespace
  + Wildcard characters (? \*)
  + Bracket characters (< > { } [ ])
  + Special characters (: ; , \ | ^ ~ $ # % & ` ")
  + Control characters (\\u0000 - \\u001f or \\u007f - \\u009f).

Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters.

Topics

* [Common State Fields (p. 131)](#_bookmark169)
* [Pass (p. 131)](#_bookmark170)
* [Task (p. 132)](#_bookmark171)
* [Choice (p. 135)](#_bookmark175)
* [Wait (p. 139)](#_bookmark177)
* [Succeed (p. 140)](#_bookmark178)
* [Fail (p. 140)](#_bookmark179)
* [Parallel (p. 140)](#_bookmark180)

Common State Fields

Type (Required)

The state's type.

Next

The name of the next state that will be run when the current state ﬁnishes. Some state types, such as Choice, allow multiple transition states.

End

Designates this state as a terminal state (it ends the execution) if set to true. There can be any number of terminal states per state machine. Only one of Next or End can be used in a state. Some state types, such as Choice, do not support or use the End ﬁeld.

Comment (Optional)

Holds a human-readable description of the state.

InputPath (Optional)

A [path (p. 143)](#_bookmark182) that selects a portion of the state's input to be passed to the state's task for processing. If omitted, it has the value $ which designates the entire input. For more information, see [Input and Output Processing (p. 143)](#_bookmark182)).

OutputPath (Optional)

A [path (p. 143)](#_bookmark182) that selects a portion of the state's input to be passed to the state's output. If omitted, it has the value $ which designates the entire input. For more information, see [Input and](#_bookmark182) [Output Processing (p. 143)](#_bookmark182).

### Pass

A Pass state ("Type": "Pass") simply passes its input to its output, performing no work. Pass states are useful when constructing and debugging state machines.

In addition to the [common state ﬁelds (p. 131)](#_bookmark169), Pass states allow the following ﬁelds:

Result (Optional)

Treated as the output of a virtual task to be passed on to the next state, and ﬁltered as prescribed by the ResultPath ﬁeld (if present).

ResultPath (Optional)

Speciﬁes where (in the input) to place the "output" of the virtual task speciﬁed in Result. The input is further ﬁltered as prescribed by the OutputPath ﬁeld (if present) before being used as the state's output. For more information, see [Input and Output Processing (p. 143)](#_bookmark182).

Parameters (Optional)

Create a collection of key-value pairs that will be passed as input. Values can be static, or selected from the input with a path. See, [InputPath and Parameters (p. 95)](#_bookmark129).

Here is an example of a Pass state that injects some ﬁxed data into the state machine, probably for testing purposes.

"No-op": {

"Type": "Pass", "Result": {

"x-datum": 0.381018,

"y-datum": 622.2269926397355

},

"ResultPath": "$.coords", "Next": "End"

}

Suppose the input to this state is:

{

"georefOf": "Home"

}

Then the output would be:

{

"georefOf": "Home", "coords": {

"x-datum": 0.381018,

"y-datum": 622.2269926397355

}

}

### Task

A Task state ("Type": "Task") represents a single unit of work performed by a state machine. In addition to the [common state ﬁelds (p. 131)](#_bookmark169), Task states have the following ﬁelds:

Resource (Required)

A URI, especially an Amazon Resource Name (ARN) that uniquely identiﬁes the speciﬁc task to execute.

Parameters (Optional)

Use Parameters to pass information to the API actions of connected resources.

ResultPath (Optional)

Speciﬁes where (in the input) to place the results of executing the task speciﬁed in Resource. The input is then ﬁltered as prescribed by the OutputPath ﬁeld (if present) before being used as the state's output. For more information, see [path (p. 143)](#_bookmark182).

Retry (Optional)

An array of objects, called Retriers, that deﬁne a retry policy in case the state encounters runtime errors. For more information, see [Retrying After an Error (p. 146)](#_bookmark187).

Catch (Optional)

An array of objects, called Catchers, that deﬁne a fallback state which is executed in case the state encounters runtime errors and its retry policy has been exhausted or is not deﬁned. For more information, see [Fallback States (p. 148)](#_bookmark188).

TimeoutSeconds (Optional)

If the task runs longer than the speciﬁed seconds, then this state fails with a States.Timeout Error Name. Must be a positive, non-zero integer. If not provided, the default value is 99999999. The count begins after the task has been started, for instance when ActivityStarted or LambdaFunctionStarted are logged in the **Execution event history**.

HeartbeatSeconds (Optional)

If more time than the speciﬁed seconds elapses between heartbeats from the task, then this state fails with an States.Timeout Error Name. Must be a positive, non-zero integer less than the number of seconds speciﬁed in the TimeoutSeconds ﬁeld. If not provided, the default value is 99999999. This value applies only to Activity tasks. The count begins when GetActivityTask receives a token and ActivityStarted is logged in the **Execution event history**.

A Task state must set either the End ﬁeld to true if the state ends the execution, or must provide a state in the Next ﬁeld that will be run upon completion of the Task state.

Here is an example:

"ActivityState": { "Type": "Task",

"Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld", "TimeoutSeconds": 300,

"HeartbeatSeconds": 60, "Next": "NextState"

}

In this example, ActivityState will schedule the HelloWorld activity for execution in the us- east-1 region on the caller's AWS account. When HelloWorld completes, the next state (here called NextState) will be run.

If this task fails to complete within 300 seconds, or does not send heartbeat notiﬁcations in intervals of 60 seconds, then the task is marked as failed. It's a good practice to set a timeout value and a heartbeat interval for long-running activities.

Specifying Resource ARNs in Tasks

The Resource ﬁeld's Amazon Resource Name (ARN) is speciﬁed using the following pattern:

arn:*partition*:*service*:*region*:*account*:*task\_type*:*name*

Where:

* partition is the AWS Step Functions partition to use, most commonly aws.
* service indicates the AWS service used to execute the task, and is either:
  + states for an [activity (p. 134)](#_bookmark173).
  + lambda for a [Lambda function (p. 134)](#_bookmark174).
* region is the [AWS region](https://docs.aws.amazon.com/general/latest/gr/rande.html) in which the Step Functions activity/state machine type or Lambda function has been created.
* account is your AWS account id.
* task\_type is the type of task to run. It will be one of the following values:
  + activity – an [activity (p. 134)](#_bookmark173).
  + function – a [Lambda function (p. 134)](#_bookmark174).
  + *servicename* – the name of a supported connected service (see [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114)).
* name is the registered resource name (activity name, Lambda function name, or service API action).

###### Note

Step Functions does not support referencing ARNs across partitions (For example: "aws-cn" cannot invoke tasks in the "aws" partition, and vice versa);

#### Task Types

The following task types are supported:

* [activity (p. 134)](#_bookmark173)
* [Lambda functions (p. 134)](#_bookmark174)
* [A supported AWS service (p. 80)](#_bookmark107)

The following sections will provide more detail about each type.

##### Activity

Activities represent workers (processes or threads), implemented and hosted by you, that perform a speciﬁc task.

Activity resource ARNs use the following syntax:

arn:*partition*:states:*region*:*account*:activity:*name*

For more information about these ﬁelds, see [Specifying Resource ARNs in Tasks (p. 133)](#_bookmark172).

###### Note

activities must be created with Step Functions (using a [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html), API action, or the Step Functions console) before their ﬁrst use.

For more information about creating an activity and implementing workers, see [Activities (p. 72)](#_bookmark101).

##### Lambda Functions

Lambda tasks execute a function using AWS Lambda. To specify a Lambda function, use the ARN of the Lambda function in the Resource ﬁeld.

Lambda function Resource ARNs use the following syntax:

arn:*partition*:lambda:*region*:*account*:function:*function\_name*

For more information about these ﬁelds, see [Specifying Resource ARNs in Tasks (p. 133)](#_bookmark172). For example:

"LambdaState": { "Type": "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloWorld", "Next": "NextState"

}

Once the Lambda function speciﬁed in the Resource ﬁeld completes, its output is sent to the state identiﬁed in the Next ﬁeld ("NextState").

##### A supported AWS service

When you reference a connected resource, Step Functions directly calls the API actions of a supported service. Specify the service and action in the Resource ﬁeld.

Connected service Resource ARNs use the following syntax:

arn:*partition*:states:*region*:*account*:*servicename*:*APIname*

###### Note

To create a synchronouse connection to a connected resource, append .sync to the *APIname*

entry in the ARN. For more information, see [Service Integrations (p. 80)](#_bookmark107).

For example:

{

"StartAt": "BATCH\_JOB",

"States": {

"BATCH\_JOB": {

"Type": "Task",

"Resource": "arn:aws:states:::batch:submitJob.sync", "Parameters": {

"JobDefinition": "preprocessing", "JobName": "PreprocessingBatchJob", "JobQueue": "SecondaryQueue", "Parameters.$": "$.batchjob.parameters", "RetryStrategy": {

"attempts": 5

}

},

"End": true

}

}

}

### Choice

A Choice state ("Type": "Choice") adds branching logic to a state machine.

In addition to the [common state ﬁelds (p. 131)](#_bookmark169), Choice states introduce the following additional ﬁelds:

Choices (Required)

An array of [Choice Rules (p. 137)](#_bookmark176) that determines which state the state machine transitions to next.

Default (Optional, Recommended)

The name of the state to transition to if none of the transitions in Choices is taken.

**Important**

Choice states do not support the End ﬁeld. In addition, they use Next only inside their

Choices ﬁeld.

The following is an example of a Choice state and other states that it transitions to.

###### Note

You must specify the $.type ﬁeld. If the state input doesn't contain the $.type ﬁeld, the execution fails and an error is displayed in the execution history.

"ChoiceStateX": { "Type": "Choice", "Choices": [

{

"Not": {

"Variable": "$.type", "StringEquals": "Private"

},

"Next": "Public"

},

{

"Variable": "$.value", "NumericEquals": 0, "Next": "ValueIsZero"

},

{

"And": [

{

"Variable": "$.value", "NumericGreaterThanEquals": 20

},

{

"Variable": "$.value", "NumericLessThan": 30

}

],

"Next": "ValueInTwenties"

}

],

"Default": "DefaultState"

},

"Public": {

"Type" : "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:Foo", "Next": "NextState"

},

"ValueIsZero": { "Type" : "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:Zero", "Next": "NextState"

},

"ValueInTwenties": { "Type" : "Task",

"Resource": "arn:aws:lambda:us-east-1:123456789012:function:Bar", "Next": "NextState"

},

"DefaultState": { "Type": "Fail", "Cause": "No Matches!"

}

In this example the state machine starts with the following input value:

{

"type": "Private", "value": 22

}

Step Functions transitions to the ValueInTwenties state, based on the value ﬁeld.

If there are no matches for the Choice state's Choices, the state provided in the Default ﬁeld runs instead. If the Default state isn't speciﬁed, the execution fails with an error.

Choice Rules

A Choice state must have a Choices ﬁeld whose value is a non-empty array, whose every element is a object called a Choice Rule. A Choice Rule contains the following:

* A **comparison** – Two ﬁelds that specify an input variable to compared, the type of comparison, and the value to compare the variable to.
* A **Next ﬁeld** – The value of this ﬁeld must match a state name in the state machine.

The following example checks whether the numerical value is equal to 1:

{

"Variable": "$.foo", "NumericEquals": 1, "Next": "FirstMatchState"

}

The following example checks whether the string is equal to MyString:

{

"Variable": "$.foo", "StringEquals": "MyString", "Next": "FirstMatchState"

}

The following example checks whether the string is greater than MyStringABC:

{

"Variable": "$.foo", "StringGreaterThan": "MyStringABC", "Next": "FirstMatchState"

}

The following example checks whether the timestamp is equal to 2001-01-01T12:00:00Z:

{

"Variable": "$.foo",

"TimestampEquals": "2001-01-01T12:00:00Z",

"Next": "FirstMatchState"

}

Step Functions examines each of the Choice Rules in the order listed in the Choices ﬁeld and transitions to the state speciﬁed in the Next ﬁeld of the ﬁrst Choice Rule in which the variable matches the value according to the comparison operator.

The following comparison operators are supported:

* And
* BooleanEquals
* Not
* NumericEquals
* NumericGreaterThan
* NumericGreaterThanEquals
* NumericLessThan
* NumericLessThanEquals
* Or
* StringEquals
* StringGreaterThan
* StringGreaterThanEquals
* StringLessThan
* StringLessThanEquals
* TimestampEquals
* TimestampGreaterThan
* TimestampGreaterThanEquals
* TimestampLessThan
* TimestampLessThanEquals

For each of these operators, the corresponding value must be of the appropriate type: string, number, Boolean, or timestamp. Step Functions doesn't attempt to match a numeric ﬁeld to a string value.

However, because timestamp ﬁelds are logically strings, it is possible that a ﬁeld considered to be a timestamp can be matched by a StringEquals comparator.

###### Note

For interoperability, don't assume that numeric comparisons work with values outside the magnitude or precision that the [IEEE 754-2008 binary64 data type](https://en.wikipedia.org/wiki/IEEE_754#Basic_and_interchange_formats) represents. In particular, integers outside of the range [-253+1, 253-1] might fail to compare in the expected way. Timestamps (for example, 2016-08-18T17:33:00Z) must conform to [RFC3339 proﬁle ISO](https://www.ietf.org/rfc/rfc3339.txt) [8601](https://www.ietf.org/rfc/rfc3339.txt), with further restrictions:

* + An uppercase T must separate the date and time portions.
  + An uppercase Z must denote that a numeric time zone oﬀset isn't present.

To understand the behavior of string comparisons, see the [Java compareTo documentation](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-). The values of the And and Or operators must be non-empty arrays of Choice Rules that must not themselves contain Next ﬁelds. Likewise, the value of a Not operator must be a single Choice Rule that must not contain Next ﬁelds.

You can create complex, nested Choice Rules using And, Not, and Or. However, the Next ﬁeld can appear only in a top-level Choice Rule.

### Wait

A Wait state ("Type": "Wait") delays the state machine from continuing for a speciﬁed time. You can choose either a relative time, speciﬁed in seconds from when the state begins, or an absolute end-time, speciﬁed as a timestamp.

In addition to the [common state ﬁelds (p. 131)](#_bookmark169), Wait states have one of the following ﬁelds:

Seconds

A time, in seconds, to wait before beginning the state speciﬁed in the Next ﬁeld.

Timestamp

An absolute time to wait until before beginning the state speciﬁed in the Next ﬁeld.

Timestamps must conform to the RFC3339 proﬁle of ISO 8601, with the further restrictions that an uppercase T must separate the date and time portions, and an uppercase Z must denote that a numeric time zone oﬀset is not present, for example, 2016-08-18T17:33:00Z.

SecondsPath

A time, in seconds, to wait before beginning the state speciﬁed in the Next ﬁeld, speciﬁed using a [path (p. 143)](#_bookmark182) from the state's input data.

TimestampPath

An absolute time to wait until before beginning the state speciﬁed in the Next ﬁeld, speciﬁed using a [path (p. 143)](#_bookmark182) from the state's input data.

**Note**

You must specify exactly one of Seconds, Timestamp, SecondsPath, or TimestampPath.

For example, the following Wait state introduces a ten second delay into a state machine:

"wait\_ten\_seconds": { "Type": "Wait", "Seconds": 10, "Next": "NextState"

}

In the next example, the Wait state waits until an absolute time: March 14th, 2016, at 1:59 PM UTC.

"wait\_until" : { "Type": "Wait",

"Timestamp": "2016-03-14T01:59:00Z",

"Next": "NextState"

}

The wait duration does not have to be hard-coded. For example, given the following input data:

{

"expirydate": "2016-03-14T01:59:00Z"

}

You can select the value of "expirydate" from the input using a reference [path (p. 143)](#_bookmark182) to select it from the input data:

"wait\_until" : { "Type": "Wait",

"TimestampPath": "$.expirydate", "Next": "NextState"

}

### Succeed

A Succeed state ("Type": "Succeed") stops an execution successfully. The Succeed state is a useful target for Choice state branches that don't do anything but stop the execution.

Because Succeed states are terminal states, they have no Next ﬁeld, nor do they have need of an End

ﬁeld, for example:

"SuccessState": { "Type": "Succeed"

}

### Fail

A Fail state ("Type": "Fail") stops the execution of the state machine and marks it as a failure.

The Fail state only allows the use of Type and Comment ﬁelds from the set of [common state](#_bookmark169) [ﬁelds (p. 131)](#_bookmark169). In addition, the Fail state allows the following ﬁelds:

Cause (Optional)

Provides a custom failure string that can be used for operational or diagnostic purposes.

Error (Optional)

Provides an error name that can be used for error handling (Retry/Catch), operational or diagnostic purposes.

Because Fail states always exit the state machine, they have no Next ﬁeld nor do they require an End

ﬁeld.

For example:

"FailState": { "Type": "Fail",

"Cause": "Invalid response.", "Error": "ErrorA"

}

### Parallel

The Parallel state ("Type": "Parallel") can be used to create parallel branches of execution in your state machine.

In addition to the [common state ﬁelds (p. 131)](#_bookmark169), Parallel states introduce these additional ﬁelds:

Branches (Required)

An array of objects that specify state machines to execute in parallel. Each such state machine object must have ﬁelds named States and StartAt whose meanings are exactly like those in the top level of a state machine.

ResultPath (Optional)

Speciﬁes where (in the input) to place the output of the branches. The input is then ﬁltered as prescribed by the OutputPath ﬁeld (if present) before being used as the state's output. For more information, see [Input and Output Processing (p. 143)](#_bookmark182).

Retry (Optional)

An array of objects, called Retriers that deﬁne a retry policy in case the state encounters runtime errors. For more information, see [Retrying After an Error (p. 146)](#_bookmark187).

Catch (Optional)

An array of objects, called Catchers that deﬁne a fallback state which is executed in case the state encounters runtime errors and its retry policy has been exhausted or is not deﬁned. For more information, see [Fallback States (p. 148)](#_bookmark188).

A Parallel state causes AWS Step Functions to execute each branch, starting with the state named in that branch's StartAt ﬁeld, as concurrently as possible, and wait until all branches terminate (reach a terminal state) before processing the Parallel state's Next ﬁeld.

Here is an example:

{

"Comment": "Parallel Example.", "StartAt": "LookupCustomerInfo", "States": {

"LookupCustomerInfo": { "Type": "Parallel", "End": true,

"Branches": [

{

"StartAt": "LookupAddress", "States": {

"LookupAddress": { "Type": "Task", "Resource":

"arn:aws:lambda:us-east-1:123456789012:function:AddressFinder", "End": true

}

}

},

{

"StartAt": "LookupPhone", "States": {

"LookupPhone": { "Type": "Task", "Resource":

"arn:aws:lambda:us-east-1:123456789012:function:PhoneFinder", "End": true

}

}

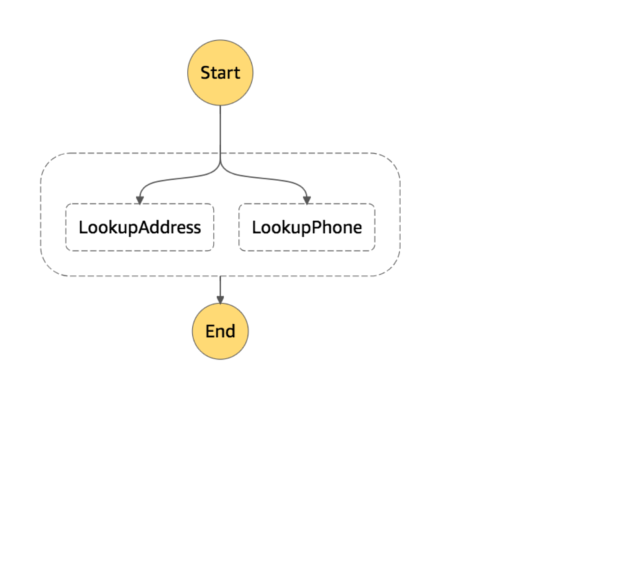
}

]

}

}

}

In this example, the LookupAddress and LookupPhone branches are executed in parallel. Here is how the visual workﬂow looks in the Step Functions console:

Each branch must be self-contained. A state in one branch of a Parallel state must not have a Next ﬁeld that targets a ﬁeld outside of that branch, nor can any other state outside the branch transition into that branch.

Parallel State Output

A Parallel state provides each branch with a copy of its own input data (subject to modiﬁcation by the InputPath ﬁeld). It generates output which is an array with one element for each branch containing the output from that branch. There is no requirement that all elements be of the same type. The output array can be inserted into the input data (and the whole sent as the Parallel state's output) by using a ResultPath ﬁeld in the usual way (see [Input and Output Processing (p. 143)](#_bookmark182)).

Here is another example:

{

"Comment": "Parallel Example.",

"StartAt": "FunWithMath", "States": {

"FunWithMath": { "Type": "Parallel", "End": true,

"Branches": [

{

"StartAt": "Add", "States": {

"Add": {

"Type": "Task",

"Resource": "arn:aws:swf:us-east-1:123456789012:task:Add", "End": true

}

}

},

{

"StartAt": "Subtract", "States": {

"Subtract": { "Type": "Task",

"Resource": "arn:aws:swf:us-east-1:123456789012:task:Subtract",

"End": true

}

}

}

]

}

}

}

If the FunWithMath state was given the array [3, 2] as input, then both the Add and Subtract states receive that array as input. The output of Add would be 5, that of Subtract would be 1, and the output of the Parallel state would be an array:

[ 5, 1 ]

##### Error Handling

If any branch fails, due to either an unhandled error or by transitioning to a Fail state, the entire Parallel state is considered to have failed and all its branches are stopped. If the error is not handled by the Parallel state itself, Step Functions will stop the execution with an error.

###### Note

When a parallel state fails, invoked Lambda functions continue to run and activity workers processing a task token will not be stopped:

* + To stop long-running Activities use heartbeats to detect if its branch has been stopped by Step Functions, and stop workers that are processing tasks. Calling [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html), [SendTaskSuccess](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskSuccess.html), or [SendTaskFailure](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskFailure.html) will throw an error if the state has failed. See [Heartbeat Errors](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html#API_SendTaskHeartbeat_Errors).
  + Running Lambda functions cannot be stopped. If you have implemented a fallback, use a

Wait state so that cleanup work happens after the Lambda function has ﬁnished.

## Input and Output Processing

In this section you will learn how to use paths and reference paths for input and output processing.

###### Note

For an overview, see [Input and Output Processing in Step Functions (p. 94)](#_bookmark128) in the [How Step](#_bookmark98) [Functions Works (p. 71)](#_bookmark98) section.

Paths

In Amazon States Language, a *path* is a string beginning with $ that you can use to identify components within JSON text. Paths follow [JsonPath](https://github.com/json-path/JsonPath) syntax.

### Reference Paths

A *reference path* is a path whose syntax is limited in such a way that it can identify only a single node in a JSON structure:

* You can access object ﬁelds using only dot (.) and square bracket ([ ]) notation.
* The operators @ .. , : ? \* aren't supported.
* Functions such as length() aren't supported.

For example, state input data contains the following values:

{

"foo": 123,

"bar": ["a", "b", "c"], "car": {

"cdr": true

}

}

In this case, the following reference paths would return:

$.foo => 123

$.bar => ["a", "b", "c"]

$.car.cdr => true

Certain states use paths and reference paths to control the ﬂow of a state machine or conﬁgure a states's settings or options.

#### Paths in InputPath, ResultPath, and OutputPath Fields

To specify how to use part of the state's input and what to send as output to the next state, you can use

InputPath, OutputPath, and ResultPath:

* For InputPath and OutputPath, you must use a [path (p. 143)](#_bookmark183) that follows the [JsonPath](https://github.com/json-path/JsonPath) syntax.
* For ResultPath, you must use a [reference path (p. 143)](#_bookmark184).

##### InputPath

The InputPath ﬁeld selects a portion of the state's input to pass to the state's task for processing. If you omit the ﬁeld, it gets the $ value, representing the entire input. If you use null, the input is discarded (not sent to the state's task) and the task receives JSON text representing an empty object {}.

###### Note

A path can yield a selection of values. Consider the following example:

{ "a": [1, 2, 3, 4] }

If you apply the path $.a[0:2], the following is the result:

[ 1, 2 ]

##### ResultPath

Usually, if a state executes a task, the task results are sent along as the state's output (which becomes the input for the next task).

If a state doesn't execute a task, the state's own input is sent, unmodiﬁed, as its output. However, when you specify a path in the value of a state's ResultPath and OutputPath ﬁelds, diﬀerent scenarios become possible.

The ResultPath takes the results of executing the state's task and places them in the input. Next, the OutputPath selects a portion of the input to send as the state's output. The ResultPath might add the results of executing the state's task to the input, overwrite an existing part, or overwrite the entire input:

* If the ResultPath matches an item in the state's input, only that input item is overwritten with the results of executing the state's task. The entire modiﬁed input becomes available to the state's output.
* If the ResultPath doesn't match an item in the state's input, an item is added to the input. The item contains the results of executing the state's task. The expanded input becomes available to the state's output.
* If the ResultPath has the default value of $, it matches the entire input. In this case, the results of the state execution overwrite the input entirely and the input becomes available to pass along.
* If the ResultPath is null, the results of executing the state are discarded and the input is untouched.

###### Note

ResultPath ﬁeld values must be [reference paths (p. 143)](#_bookmark184). For more information on

ResultPath see [ResultPath (p. 97)](#_bookmark130)

##### OutputPath

* If the OutputPath matches an item in the state's input, only that input item is selected. This input item becomes the state's output.
* If the OutputPath doesn't match an item in the state's input, an exception speciﬁes an invalid path. For more information, see [Errors (p. 146)](#_bookmark185).
* If the OutputPath has the default value of $, this matches the entire input completely. In this case, the entire input is passed to the next state.
* If the OutputPath is null, JSON text representing an empty object {} is sent to the next state.

The following example demonstrates how InputPath, ResultPath, and OutputPath ﬁelds work in practice. Consider the following input for the current state:

{

"title": "Numbers to add", "numbers": { "val1": 3, "val2": 4 }

}

In addition, the state has the following InputPath, ResultPath, and OutputPath ﬁelds:

"InputPath": "$.numbers", "ResultPath": "$.sum",

"OutputPath": "$"

The state's task receives only the numbers object from the input. In turn, if this task returns 7, the output of this state is as follows:

{

"title": "Numbers to add", "numbers": { "val1": 3, "val2": 4 } "sum": 7

}

You can slightly modify the OutputPath:

"InputPath": "$.numbers", "ResultPath": "$.sum", "OutputPath": "$.sum"

As before, you use the following state input data:

{

"numbers": { "val1": 3, "val2": 4 }

}

However, now the state output data is 7.

## Errors

Any state can encounter runtime errors. Errors can arise because of state machine deﬁnition issues (for example, no matching rule in a Choice state), task failures (for example, an exception from a Lambda function) or because of transient issues, such as network partition events. When a state reports an error, the default course of action for AWS Step Functions is to fail the execution entirely.

### Error Representation

Errors are identiﬁed in Amazon States Language by case-sensitive strings, called Error Names. Amazon States Language deﬁnes a set of built-in strings naming well-known errors, all of which begin with the preﬁx "States.":

Predeﬁned Error Codes

States.ALL

A wild-card that matches any Error Name.

States.Timeout

A Task state either ran longer than the "TimeoutSeconds" value, or failed to send a heartbeat for a time longer than the "HeartbeatSeconds" value.

States.TaskFailed

A Task state failed during the execution.

States.Permissions

A Task state failed because it had insuﬃcient privileges to execute the speciﬁed code.

States may report errors with other names, which must not begin with the preﬁx "States.".

### Retrying After an Error

Task and Parallel states may have a ﬁeld named Retry, whose value must be an array of objects, called Retriers. An individual Retrier represents a certain number of retries, usually at increasing time intervals.

###### Note

Retries are treated as state transitions. For information on how state transitions aﬀect billing, see [Step Functions Pricing](https://aws.amazon.com/step-functions/pricing/).

A Retrier contains the following ﬁelds:

ErrorEquals (Required)

A non-empty array of Strings that match Error Names. When a state reports an error, Step Functions scans through the Retriers and, when the Error Name appears in this array, it implements the retry policy described in this Retrier.

IntervalSeconds (Optional)

An integer that represents the number of seconds before the ﬁrst retry attempt (default 1).

MaxAttempts (Optional)

A positive integer, representing the maximum number of retry attempts (default 3). If the error recurs more times than speciﬁed, retries cease and normal error handling resumes. A value of 0 is permitted and indicates that the error or errors should never be retried.

BackoffRate (Optional)

A number that is the multiplier by which the retry interval increases on each attempt (default 2.0).

Here is an example of a Retry ﬁeld that will make 4 retry attempts after waits of 3, 4.5, 6.75 and 10.125 seconds:

"Retry" : [

{

"ErrorEquals": [ "States.Timeout" ], "IntervalSeconds": 3,

"MaxAttempts": 4,

"BackoffRate": 1.5

}

]

The reserved name States.ALL appearing in a Retrier's ErrorEquals ﬁeld is a wildcard that matches any Error Name. It must appear alone in the ErrorEquals array and must appear in the last Retrier in the Retry array.

Here is an example of a Retry ﬁeld that will retry any error except for States.Timeout:

"Retry" : [

{

"ErrorEquals": [ "States.Timeout" ], "MaxAttempts": 0

},

{

"ErrorEquals": [ "States.ALL" ]

}

]

Complex Retry Scenarios

A Retrier's parameters apply across all visits to that Retrier in the context of a single state execution. This is best illustrated by an example; consider the following Task state:

"X": {

"Type": "Task",

"Resource": "arn:aws:states:us-east-1:123456789012:task:X", "Next": "Y",

"Retry": [

{

"ErrorEquals": [ "ErrorA", "ErrorB" ], "IntervalSeconds": 1,

"BackoffRate": 2.0,

"MaxAttempts": 2

},

{

"ErrorEquals": [ "ErrorC" ], "IntervalSeconds": 5

}

],

"Catch": [

{

"ErrorEquals": [ "States.ALL" ], "Next": "Z"

}

]

}

Suppose that this task fails ﬁve successive times, outputting Error Names "ErrorA", "ErrorB", "ErrorC", "ErrorB", and "ErrorB". The ﬁrst two errors match the ﬁrst retrier, and cause waits of one and two seconds. The third error matches the second retrier, and causes a wait of ﬁve seconds. The fourth error matches the ﬁrst retrier and causes a wait of four seconds. The ﬁfth error also matches the ﬁrst retrier, but it has already reached its limit of two retries ("MaxAttempts") for that particular error ("ErrorB") so it fails and execution is redirected to the "Z" state via the "Catch" ﬁeld.

Note that once the system transitions to another state, no matter how, all Retrier parameters are reset.

###### Note

You can generate custom error names (such as ErrorA and ErrorB above) using either an [activity (p. 134)](#_bookmark173) or [Lambda functions (p. 134)](#_bookmark174). For more information, see [Handling Error](#_bookmark58) [Conditions Using a State Machine (p. 34)](#_bookmark58).

### Fallback States

Task and Parallel states may have a ﬁeld named Catch, whose value must be an array of objects, called Catchers.

A Catcher contains the following ﬁelds:

ErrorEquals (Required)

A non-empty array of Strings that match Error Names, speciﬁed exactly as with the Retrier ﬁeld of the same name.

Next (Required)

A string which must exactly match one of the state machine's state names.

ResultPath (Optional)

A [path (p. 143)](#_bookmark182) which determines what is sent as input to the state speciﬁed by the Next ﬁeld.

When a state reports an error and either there is no Retry ﬁeld, or retries have failed to resolve the error, AWS Step Functions scans through the Catchers in the order listed in the array, and when the Error Name appears in the value of a Catcher's ErrorEquals ﬁeld, the state machine transitions to the state named in the Next ﬁeld.

The reserved name States.ALL appearing in a Catcher's ErrorEquals ﬁeld is a wildcard that matches any Error Name. It must appear alone in the ErrorEquals array and must appear in the last Catcher in the Catch array.

Here is an example of a Catch ﬁeld that will transition to the state named "RecoveryState" when a Lambda function outputs an unhandled Java exception, and otherwise to the "EndState" state.

"Catch": [

{

"ErrorEquals": [ "java.lang.Exception" ], "ResultPath": "$.error-info",

"Next": "RecoveryState"

},

{

"ErrorEquals": [ "States.ALL" ], "Next": "EndState"

}

]

Each Catcher can specify multiple errors to handle.

When AWS Step Functions transitions to the state speciﬁed in a Catcher, it sends along as input JSON text that is diﬀerent than what it would normally send to the next state when there was no error. This JSON text represents an object containing a ﬁeld "Error" whose value is a string containing the error name. The object will also, usually, contain a ﬁeld "Cause" that has a human-readable description of the error. We refer to this object as the Error Output.

In this example, the ﬁrst Catcher contains a ResultPath ﬁeld. This works in a similar fashion to a ResultPath ﬁeld in a state's top level—it takes the results of executing the state and overwrites a portion of the state's input, or all of the state's input, or it takes the results and adds them to the input. In the case of an error handled by a Catcher, the result of executing the state is the Error Output.

So in the example, for the ﬁrst Catcher the Error Output will be added to the input as a ﬁeld named error-info (assuming there is not already a ﬁeld by that name in the input) and the entire input will be sent to RecoveryState. For the second Catcher, the Error Output will overwrite the input and so just the Error Output will be sent to EndState. When not speciﬁed, the ResultPath ﬁeld defaults to $ which selects, and so overwrites, the entire input.

When a state has both Retry and Catch ﬁelds, Step Functions uses any appropriate Retriers ﬁrst and only applies the matching Catcher transition if the retry policy fails to resolve the error.

# Best Practices for Step Functions

The following best practices for implementing Step Functions workﬂows can help you optimize the performance of your implementations.

Topics

* + [Use Timeouts to Avoid Stuck Executions (p. 150)](#_bookmark190)
  + [Use ARNs Instead of Passing Large Payloads (p. 150)](#_bookmark191)
  + [Avoid Reaching the History Limit (p. 150)](#_bookmark192)
  + [Handle Lambda Service Exceptions (p. 151)](#_bookmark193)
  + [Avoid Latency When Polling for Activity Tasks (p. 151)](#_bookmark194)

Use Timeouts to Avoid Stuck Executions

By default, the Amazon States Language doesn't set timeouts in state machine deﬁnitions. Without an explicit timeout, Step Functions often relies solely on a response from an activity worker to know that a task is complete. If something goes wrong and TimeoutSeconds isn't speciﬁed, an execution is stuck waiting for a response that will never come.

To avoid this, specify a reasonable timeout limit when you create a task in your state machine. For example:

"ActivityState": { "Type": "Task",

"Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld", "TimeoutSeconds": 300,

"HeartbeatSeconds": 60, "Next": "NextState"

}

For more information, see [Task (p. 132)](#_bookmark171) in the Amazon States Language documentation.

## Use ARNs Instead of Passing Large Payloads

Executions that pass large payloads of data between states can be terminated. If the data you are passing between states might grow to over 32 KB, use Amazon Simple Storage Service (Amazon S3) to store the data, and pass the Amazon Resource Name instead of the raw data. Alternatively, adjust your implementation so that you pass smaller payloads in your executions.

For more information, see:

* [Amazon Simple Storage Service Developer Guide](https://docs.aws.amazon.com/AmazonS3/latest/dev/)
* [Amazon Resource Names (ARNs)](https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html)

## Avoid Reaching the History Limit

AWS Step Functions has a hard limit of 25,000 entries in the execution history. To avoid reaching this limit for long-running executions, implement a pattern that uses an AWS Lambda function that can start a new execution of your state machine to split ongoing work across multiple workﬂow executions.

For more information, see the [Continue as a New Execution (p. 58)](#_bookmark87) tutorial.

## Handle Lambda Service Exceptions

AWS Lambda can occasionally experience transient service errors. In this case, invoking Lambda will result in a 500 error such as ServiceException, AWSLambdaException, or SdkClientException. As a best practice, proactively handle these exceptions in your state machine to Retry invoking your Lambda function, or to Catch the error.

Lambda errors are reported as Lambda.*ErrorName*. To retry a Lambda service exception error, you could use the following Retry code:

"Retry": [ {

"ErrorEquals": [ "Lambda.ServiceException", "Lambda.AWSLambdaException", "Lambda.SdkClientException"],

"IntervalSeconds": 2,

"MaxAttempts": 6,

"BackoffRate": 2

} ]

###### Note

Unhandled errors in Lambda are reported as Lambda.Unknown in the error output. These include out-of-memory errors, function timeouts, and hitting the concurrent Lambda invoke limit. You can match on Lambda.Unknown, States.ALL, or States.TaskFailed to handle these errors. For more information about Lambda Handled and Unhandled errors, see FunctionError in the [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html#API_Invoke_ResponseSyntax).

For more information, see:

* [Retrying After an Error (p. 105)](#_bookmark140)
* [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58)
* [Lambda Invoke Errors](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html#API_Invoke_Errors)

## Avoid Latency When Polling for Activity Tasks

The [GetActivityTask](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html) API is designed to provide a [taskToken](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html#StepFunctions-GetActivityTask-response-taskToken) *exactly-once*. If a taskToken is dropped while communicating with an activity worker, a number of GetActivityTask requests can be blocked for 60 seconds waiting for a response until GetActivityTask times out. If you only have a small number of polls waiting for a response, it is possible that all requests will queue up behind the blocked request and stop. However, if you have a large number of outstanding polls for each activity

ARN, and some percentage of your requests are stuck waiting, there will be many more that can still get a taskToken and begin to process work.

For production systems, we recommend at least 100 open polls per activity ARN's at each point in time. If one poll gets blocked, and a portion of those polls queue up behind it, there are still many more requests that will receive a taskToken to process work while the GetActivityTask request is blocked.

To avoid these kinds of latency problems when polling for tasks:

* Implement your pollers as separate threads from the work in your activity worker implementation.
* Have at least 100 open polls per activity ARN at each point in time.

For an example activity worker where the poller threads are separate from the work threads, see [Example Activity Worker in Ruby (p. 74)](#_bookmark106). For more information on activities and activity workers see [Activities (p. 72)](#_bookmark101).

# Limits

AWS Step Functions places limits on the sizes of certain state machine parameters, such as the number of API actions that you can make during a certain time period or the number of state machines that you can deﬁne. Although these limits are designed to prevent a misconﬁgured state machine from consuming all of the resources of the system, they aren't hard limits.

###### Note

If a particular stage of your state machine execution or activity execution takes too long, you can conﬁgure a state machine timeout to cause a timeout event.

Topics

* + [General Limits (p. 152)](#_bookmark196)
  + [Limits Related to Accounts (p. 153)](#_bookmark197)
  + [Limits Related to State Machine Executions (p. 153)](#_bookmark198)
  + [Limits Related to Task Executions (p. 153)](#_bookmark199)
  + [Limits Related to API Action Throttling (p. 154)](#_bookmark200)
  + [Limits Related to State Throttling (p. 155)](#_bookmark201)
  + [Restrictions Related to Tagging (p. 155)](#_bookmark202)
  + [Requesting a Limit Increase (p. 156)](#_bookmark203)

General Limits

|  |  |
| --- | --- |
| **Limit** | **Description** |
| Names in Step Functions | State machine, execution, and activity names must be 1–80 characters in length, must be unique for your account and region, and must not contain any of the following:   * Whitespace * Wildcard characters (? \*) * Bracket characters (< > { } [ ]) * Special characters (: ; , \ | ^ ~ $ # % &   ` ")   * Control characters (\\u0000 - \\u001f or \   \u007f - \\u009f).  Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters. |

Limits Related to Accounts

|  |  |
| --- | --- |
| **Limit** | **Description** |
| Maximum number of registered activities | 10,000 |
| Maximum number of registered state machines | 10,000 |
| Maximum number of API actions | Beyond infrequent spikes, applications may be throttled if they make a large number of API actions in a very short period of time. |
| Maximum request size | 1 MB per request. This is the total data size per Step Functions API request, including the request header and all other associated request data. |

Limits Related to State Machine Executions

|  |  |
| --- | --- |
| **Limit** | **Description** |
| Maximum open executions | 1,000,000. Exceeding this limit will cause an ExecutionLimitExceeded  error. |
| Maximum execution time | 1 year. If an execution runs for more than the 1 year limit, it will fail with a States.Timeout error and emit a ExecutionsTimedout CloudWatch metric. |
| Maximum execution history size | 25,000 events. If the execution history reaches this limit the execution will fail. To avoid this, see [Avoid Reaching the History Limit (p. 150)](#_bookmark192). |
| Maximum execution idle time | 1 year (constrained by execution time limit) |
| Maximum execution history retention time | 90 days. After this time, you can no longer retrieve or view the execution history. There is no further limit to the number of closed executions that Step Functions retains. |

Limits Related to Task Executions

|  |  |
| --- | --- |
| **Limit** | **Description** |
| Maximum task execution time | 1 year (constrained by execution time limit) |
| Maximum time Step Functions keeps a task in the queue | 1 year (constrained by execution time limit) |
| Maximum activity pollers per ARN | 1,000 pollers calling GetActivityTask per Amazon Resource Name. Exceeding this limit results in the error: *"The maximum number of workers concurrently polling for activity tasks has been reached."* |

|  |  |
| --- | --- |
| **Limit** | **Description** |
| Maximum input or result data size for a task, state, or execution | 32,768 characters. This limit aﬀects tasks (activity or Lambda function), state or execution result data, and input data when scheduling a task, entering a state, or starting an execution. |

Limits Related to API Action Throttling

Some Step Functions API actions are throttled using a token bucket scheme to maintain service bandwidth.

###### Note

Throttling limits are per account, per region. AWS Step Functions may increase both the bucket size and reﬁll rate at any time. Do not rely on these throttling rates to limit your costs.

|  |  |  |
| --- | --- | --- |
| **API Name** | **Bucket Size** | **Reﬁll Rate per Second** |
| CreateActivity | 100 | 1 |
| CreateStateMachine | 100 | 1 |
| DeleteActivity | 100 | 1 |
| DeleteStateMachine | 100 | 1 |
| DescribeActivity | 200 | 1 |
| DescribeExecution | 200 | 2 |
| DescribeStateMachine | 200 | 1 |
| DescribeStateMachineForExe | c1u0t0ion | 1 |
| GetActivityTask | 1,000 | 25 |
| GetExecutionHistory | 250 | 5 |
| ListActivities | 100 | 1 |
| ListExecutions | 100 | 2 |
| ListStateMachines | 100 | 1 |
| SendTaskFailure | 1,000 | 25 |
| SendTaskHeartbeat | 1,000 | 25 |
| SendTaskSuccess | 1,000 | 25 |
| StartExecution — *In US East (N. Virginia), US West (Oregon), and EU (Ireland)* | 1000 | 200 |
| StartExecution — *All other regions* | 500 | 25 |

|  |  |  |
| --- | --- | --- |
| **API Name** | **Bucket Size** | **Reﬁll Rate per Second** |
| StopExecution — *In US East (N. Virginia), US West (Oregon), and EU (Ireland)* | 1000 | 200 |
| StopExecution — *All other regions* | 500 | 25 |
| UpdateStateMachine | 200 | 1 |

## Limits Related to State Throttling

Step Functions state transitions are throttled using a token bucket scheme to maintain service bandwidth.

###### Note

Throttling on the StateTransition service metric is reported as ExecutionThrottled

in CloudWatch. For more information, see the [ExecutionThrottled CloudWatch](#_bookmark208) [metric (p. 158)](#_bookmark208).

|  |  |  |
| --- | --- | --- |
| **Service Metric** | **Bucket Size** | **Reﬁll Rate per Second** |
| StateTransition — *In US East (N. Virginia), US West (Oregon), and EU (Ireland)* | 5000 | 1000 |
| StateTransition — *All other regions* | 800 | 400 |

## Restrictions Related to Tagging

Be aware of these restrictions when tagging Step Functions resources.

###### Note

Tagging restrictions cannot be increased like other limits.

|  |  |
| --- | --- |
| **Restriction** | **Description** |
| Maximum number of tags per resource | 50 |
| Maximum key length | 128 Unicode characters in UTF-8 |
| Maximum value length | 256 Unicode characters in UTF-8 |
| Preﬁx restriction | Do not use the aws: preﬁx in your tag names or values because it is reserved for AWS use. You can't edit or delete tag names or values with this preﬁx. Tags with this preﬁx do not count against your tags per resource limit. |
| Character restrictions | Tags may only contain unicode letters, digits, whitespace, or these symbols: \_ . : / = + - @. |

## Requesting a Limit Increase

Use the **Support Center** page in the AWS Management Console to request a limit increase for resources provided by AWS Step Functions on a per-region basis. For more information, see [To Request a Limit](https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html) [Increase](https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html) in the *AWS General Reference*.

# Monitoring and Logging

This section provides information about monitoring and logging Step Functions.

Topics

* + [Monitoring Step Functions Using CloudWatch (p. 157)](#_bookmark205)
  + [Logging Step Functions using AWS CloudTrail (p. 164)](#_bookmark211)

Monitoring Step Functions Using CloudWatch

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Step Functions and your AWS solutions. You should collect as much monitoring data from the AWS services that you use so that you can more easily debug any multi-point failures. Before you start monitoring Step Functions, you should create a monitoring plan that answers the following questions:

* What are your monitoring goals?
* What resources will you monitor?
* How often will you monitor these resources?
* What monitoring tools will you use?
* Who will perform the monitoring tasks?
* Who should be notiﬁed when something goes wrong?

The next step is to establish a baseline for normal Step Functions performance in your environment. To do this, measure performance at various times and under diﬀerent load conditions. As you monitor Step Functions, you should consider storing historical monitoring data. Such data can give you a baseline to compare against current performance data, to identify normal performance patterns and performance anomalies, and to devise ways to address issues.

For example, with Step Functions, you can monitor how many activities or Lambda tasks fail due to a heartbeat timeout. When performance falls outside your established baseline, you might have to change your heartbeat interval.

To establish a baseline you should, at a minimum, monitor the following metrics:

* ActivitiesStarted
* ActivitiesTimedOut
* ExecutionsStarted
* ExecutionsTimedOut
* LambdaFunctionsStarted
* LambdaFunctionsTimedOut

The following sections describe metrics that Step Functions provides to CloudWatch. You can use these metrics to track your state machines and activities and to set alarms on threshold values. You can view metrics using the AWS Management Console.

Topics

* + [Metrics that Report a Time Interval (p. 158)](#_bookmark206)
  + [Metrics that Report a Count (p. 158)](#_bookmark207)
  + [State Machine Metrics (p. 158)](#_bookmark208)
  + [Viewing Metrics for Step Functions (p. 161)](#_bookmark209)
  + [Setting Alarms for Step Functions (p. 162)](#_bookmark210)

### Metrics that Report a Time Interval

Some of the Step Functions CloudWatch metrics are *time intervals*, always measured in milliseconds. These metrics generally correspond to stages of your execution for which you can set state machine, activity, and Lambda function timeouts, with descriptive names.

For example, the ActivityRunTime metric measures the time it takes for an activity to complete after it begins to execute. You can set a timeout value for the same time period.

In the CloudWatch console, you can get the best results if you choose **average** as the display statistic for time interval metrics.

### Metrics that Report a Count

Some of the Step Functions CloudWatch metrics report results as a *count*. For example,

ExecutionsFailed records the number of failed state machine executions.

In the CloudWatch console, you can get the best results if you choose **sum** as the display statistic for count metrics.

### State Machine Metrics

The following metrics are available for Step Functions state machines:

#### Execution Metrics

The AWS/States namespace includes the following metrics for Step Functions executions:

|  |  |
| --- | --- |
| **Metric** | **Description** |
| ExecutionTime | The interval, in milliseconds, between the time the execution starts and the time it closes. |
| ExecutionThrottled | The number of StateEntered events and retries that have been throttled. This is related to StateTransition throttling. For more information, see [Limits Related to State Throttling](https://docs.aws.amazon.com/step-functions/latest/dg/limits.html#service-limits-api-state-throttling) in the *AWS Step Functions Developer Guide*. |
| ExecutionsAborted | The number of aborted or terminated executions. |
| ExecutionsFailed | The number of failed executions. |
| ExecutionsStarted | The number of started executions. |
| ExecutionsSucceeded | The number of successfully completed executions. |
| ExecutionsTimedOut | The number of executions that time out for any reason. |

##### Dimension for Step Functions Execution Metrics

|  |  |
| --- | --- |
| **Dimension** | **Description** |
| StateMachineArn | The ARN of the state machine for the execution in question. |

Activity Metrics

The AWS/States namespace includes the following metrics for Step Functions activities:

|  |  |
| --- | --- |
| **Metric** | **Description** |
| ActivityRunTime | The interval, in milliseconds, between the time the activity starts and the time it closes. |
| ActivityScheduleTime | The interval, in milliseconds, for which the activity stays in the schedule state. |
| ActivityTime | The interval, in milliseconds, between the time the activity is scheduled and the time it closes. |
| ActivitiesFailed | The number of failed activities. |
| ActivitiesHeartbeatTimedOu | tThe number of activities that time out due to a heartbeat timeout. |
| ActivitiesScheduled | The number of scheduled activities. |
| ActivitiesStarted | The number of started activities. |
| ActivitiesSucceeded | The number of successfully completed activities. |
| ActivitiesTimedOut | The number of activities that time out on close. |

##### Dimension for Step Functions Activity Metrics

|  |  |
| --- | --- |
| **Dimension** | **Description** |
| ActivityArn | The ARN of the activity. |

Lambda Function Metrics

The AWS/States namespace includes the following metrics for Step Functions Lambda functions:

|  |  |
| --- | --- |
| **Metric** | **Description** |
| LambdaFunctionRunTime | The interval, in milliseconds, between the time the Lambda function starts and the time it closes. |
| LambdaFunctionScheduleTime | The interval, in milliseconds, for which the Lambda function stays in the schedule state. |
| LambdaFunctionTime | The interval, in milliseconds, between the time the Lambda function is scheduled and the time it closes. |

|  |  |
| --- | --- |
| **Metric** | **Description** |
| LambdaFunctionsFailed | The number of failed Lambda functions. |
| LambdaFunctionsHeartbeatTi | mTehdeOnuutmber of Lambda functions that time out due to a heartbeat timeout. |
| LambdaFunctionsScheduled | The number of scheduled Lambda functions. |
| LambdaFunctionsStarted | The number of started Lambda functions. |
| LambdaFunctionsSucceeded | The number of successfully completed Lambda functions. |
| LambdaFunctionsTimedOut | The number of Lambda functions that time out on close. |

##### Dimension for Step Functions Lambda Function Metrics

|  |  |
| --- | --- |
| **Dimension** | **Description** |
| LambdaFunctionArn | The ARN of the Lambda function. |

Service Metrics

The AWS/States namespace includes the following metrics for the Step Functions service:

|  |  |
| --- | --- |
| **Metric** | **Description** |
| ThrottledEvents | The count of requests that have been throttled. |
| ProvisionedBucketSize | The count of available requests per second. |
| ProvisionedRefillRate | The count of requests per second that are allowed into the bucket. |
| ConsumedCapacity | The count of requests per second. |

##### Dimension for Step Functions Service Metrics

|  |  |
| --- | --- |
| **Dimension** | **Description** |
| StateTransition | Filters data to show State Transitions metrics. |

API Metrics

The AWS/States namespace includes the following metrics for the Step Functions API:

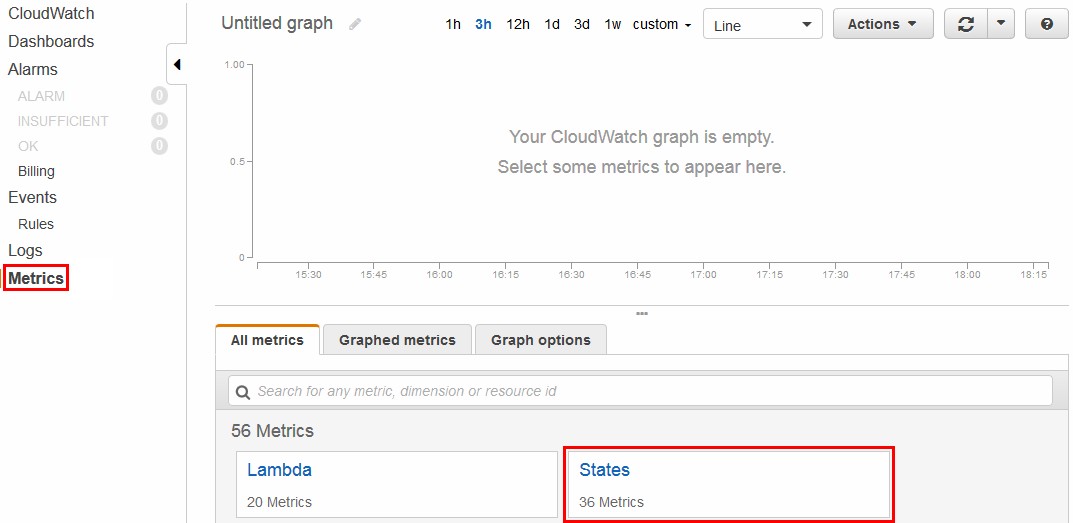
|  |  |
| --- | --- |
| **Metric** | **Description** |
| ThrottledEvents | The count of requests that have been throttled. |
| ProvisionedBucketSize | The count of available requests per second. |

|  |  |
| --- | --- |
| **Metric** | **Description** |
| ProvisionedRefillRate | The count of requests per second that are allowed into the bucket. |
| ConsumedCapacity | The count of requests per second. |

##### Dimension for Step Functions API Metrics

|  |  |
| --- | --- |
| **Dimension** | **Description** |
| APIName | Filters data to an API of the speciﬁed API name. |

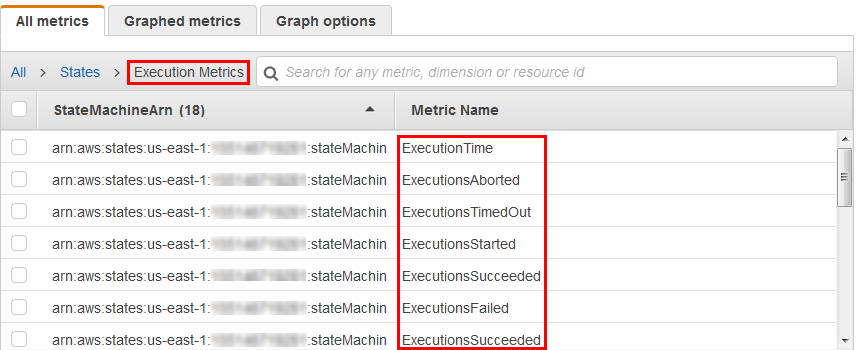
### Viewing Metrics for Step Functions

1. Open the AWS Management Console and navigate to **CloudWatch**.
2. Choose **Metrics** and on the **All Metrics** tab, choose **States**.

If you ran any executions recently, you will see up to three types of metrics:

* + Execution Metrics
  + **Activity Function Metrics**
  + **Lambda Function Metrics**

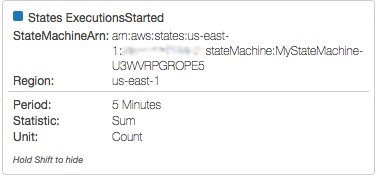
1. Choose a metric type to see a list of metrics.



* + To sort your metrics by **Metric Name** or **StateMachineArn**, use the column headings.
  + To view graphs for a metric, choose the box next to the metric on the list. You can change the graph parameters using the time range controls above the graph view.

You can choose custom time ranges using relative or absolute values (speciﬁc days and times). You can also use the drop-down list to display values as lines, stacked areas, or numbers (values).

* + To view the details about a graph, hover over the metric color code which appears below the graph.

The metric's details are displayed.

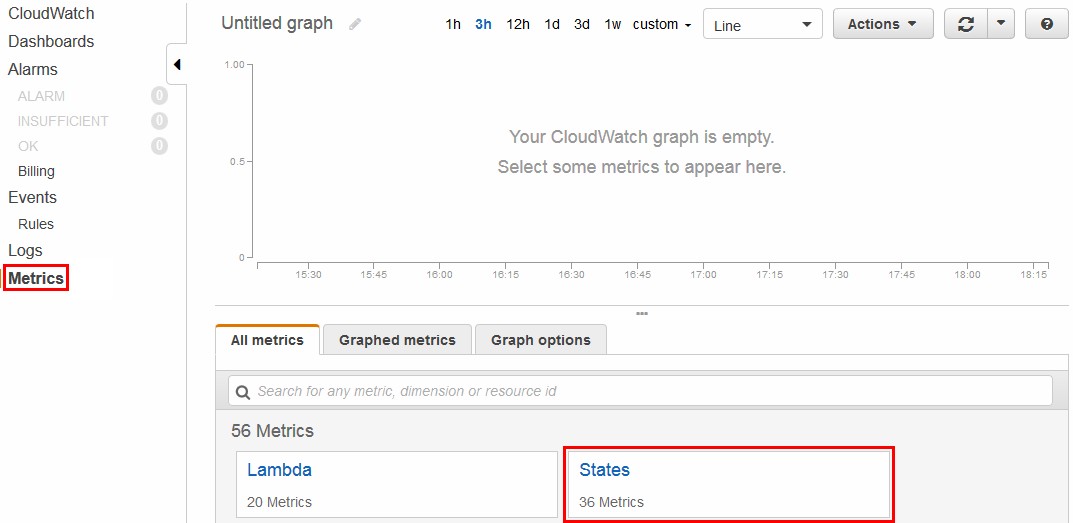
For more information about working with CloudWatch metrics, see [Using Amazon CloudWatch Metrics](https://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/working_with_metrics.html) in the *Amazon CloudWatch User Guide*.

### Setting Alarms for Step Functions

You can use CloudWatch alarms to perform actions. For example, if you want to know when an alarm threshold is reached, you can set an alarm to send a notiﬁcation to an Amazon SNS topic or to send an email when the StateMachinesFailed metric rises above a certain threshold.

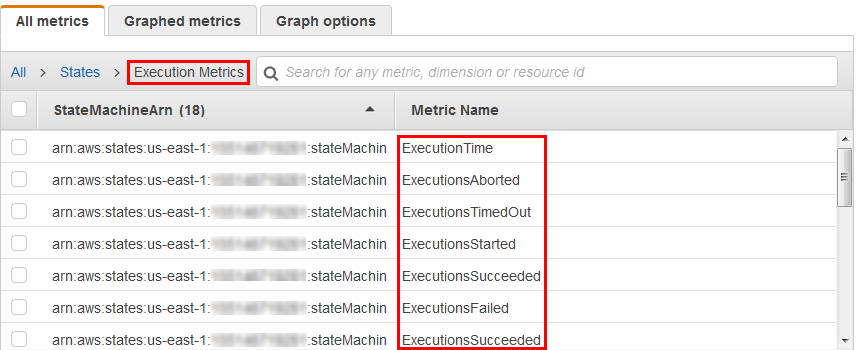
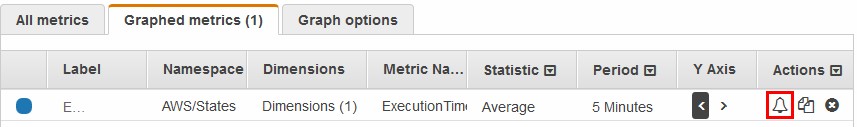
#### To set an alarm on a metric

1. Open the AWS Management Console and navigate to **CloudWatch**.
2. Choose **Metrics** and on the **All Metrics** tab, choose **States**.

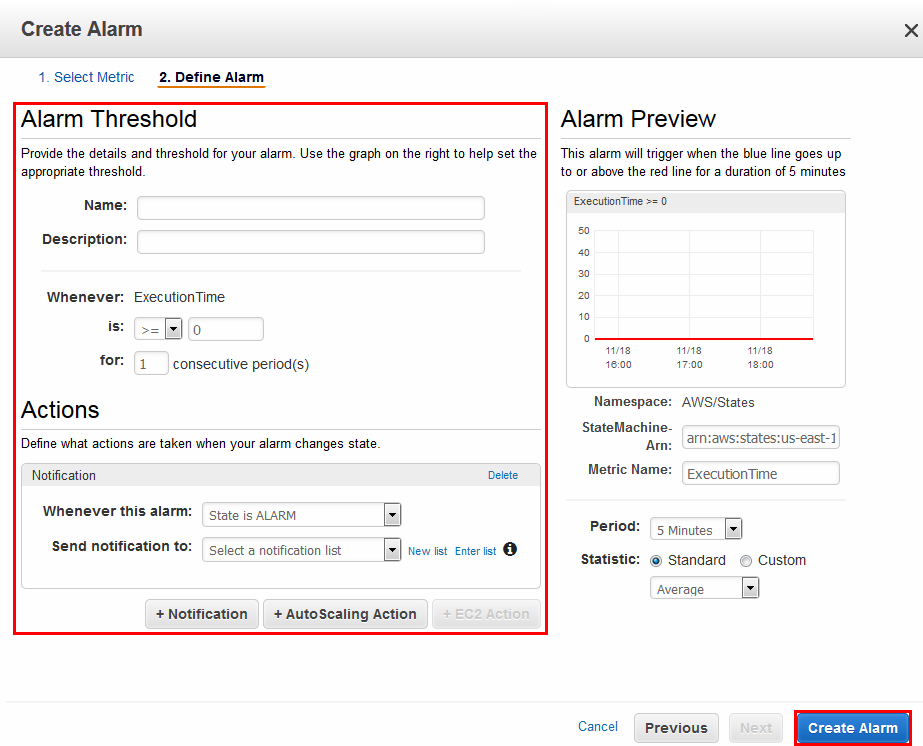


If you ran any executions recently, you will see up to three types of metrics:

* + Execution Metrics
  + **Activity Function Metrics**
  + **Lambda Function Metrics**

1. Choose a metric type to see a list of metrics.
2. Choose a metric and then choose **Graphed metrics**.
3. Choose next to a metric on the list.

The **Create Alarm** dialog box is displayed.



1. Enter the values for the **Alarm threshold** and **Actions** and then choose **Create Alarm**.

For more information about setting and using CloudWatch alarms, see [Creating Amazon CloudWatch](https://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/AlarmThatSendsEmail.html) [Alarms](https://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/AlarmThatSendsEmail.html) in the *Amazon CloudWatch User Guide*.

## Logging Step Functions using AWS CloudTrail

Step Functions is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Step Functions. CloudTrail captures all API calls for Step Functions as events, including calls from the Step Functions console and from code calls to the Step Functions APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Step Functions. If you don't conﬁgure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you

can determine the request that was made to Step Functions, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/).

### Step Functions Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Step Functions, that activity is recorded in a CloudTrail event along with other AWS service events in **Event**

**history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events.html).

For an ongoing record of events in your AWS account, including events for Step Functions, create a trail. A trail enables CloudTrail to deliver log ﬁles to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log ﬁles to the Amazon S3 bucket that you specify. Additionally, you can conﬁgure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

* [Overview for Creating a Trail](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-create-and-update-a-trail.html)
* [CloudTrail Supported Services and Integrations](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-aws-service-specific-topics.html#cloudtrail-aws-service-specific-topics-integrations)
* [Conﬁguring Amazon SNS Notiﬁcations for CloudTrail](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/getting_notifications_top_level.html)
* [Receiving CloudTrail Log Files from Multiple Regions](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/receive-cloudtrail-log-files-from-multiple-regions.html) and [Receiving CloudTrail Log Files from Multiple Accounts](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-receive-logs-from-multiple-accounts.html)

Step Functions supports logging the following actions as events in CloudTrail log ﬁles:

* [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html)
* [CreateStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateStateMachine.html)
* [DeleteActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DeleteActivity.html)
* [DeleteStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DeleteStateMachine.html)
* [StartExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html)
* [StopExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StopExecution.html)
* [UpdateStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_UpdateStateMachine.html)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

* Whether the request was made with root or IAM user credentials.
* Whether the request was made with temporary security credentials for a role or federated user.
* Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-user-identity.html).

### Example: Step Functions Log File Entries

A trail is a conﬁguration that enables delivery of events as log ﬁles to an Amazon S3 bucket that you specify. CloudTrail log ﬁles contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log ﬁles are not an ordered stack trace of the public API calls, so they do not appear in any speciﬁc order.

#### CreateActivity

The following example shows a CloudTrail log entry that demonstrates the CreateActivity action:

{

"eventVersion": "1.04",

"userIdentity": { "type": "IAMUser",

"principalId": "AIDAJYDLDBVBI4EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAIOSFODNN7EXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:17:56Z",

"eventSource": "states.amazonaws.com", "eventName": "CreateActivity", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "requestParameters": {

"name":

"OtherActivityPrefix.2016-10-27-18-16-56.894c791e-2ced-4cf4-8523-376469410c25"

},

"responseElements": {

"activityArn": "arn:aws:states:us-

east-1:123456789012:activity:OtherActivityPrefix.2016-10-27-18-16-56.894c791e-2ced-4cf4-8523-376469410c "creationDate": "Oct 28, 2016 1:17:56 AM"

},

"requestID": "37c67602-9cac-11e6-aed5-5b57d226e9ef", "eventID": "dc3becef-d06d-49bf-bc93-9b76b5f00774", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

#### CreateStateMachine

The following example shows a CloudTrail log entry that demonstrates the CreateStateMachine

action:

"responseElements": {

"stateMachineArn": "arn:aws:states:us-

east-1:123456789012:stateMachine:testUser.2016-10-27-18-17-06.bd144e18-0437-476e-9bb", "creationDate": "Oct 28, 2016 1:18:07 AM"

},

"requestID": "3da6370c-9cac-11e6-aed5-5b57d226e9ef", "eventID": "84a0441d-fa06-4691-a60a-aab9e46d689c", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

} }}"

\"End\": true

\"Type\": \"Pass\",

\": {

},

"eventVersion": "1.04",

"userIdentity": { "type": "IAMUser",

"principalId": "AIDAJYDLDBVBI4EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAJL5C75K4ZEXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:18:07Z",

"eventSource": "states.amazonaws.com", "eventName": "CreateStateMachine", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "requestParameters": {

"name": "testUser.2016-10-27-18-17-06.bd144e18-0437-476e-9bb",

"roleArn": "arn:aws:iam::123456789012:role/graphene/tests/graphene-execution-role", "definition": "{ \"StartAt\": \"SinglePass\", \"States\": { \"SinglePass

{

#### DeleteActivity

The following example shows a CloudTrail log entry that demonstrates the DeleteActivity action:

{

"eventVersion": "1.04", "userIdentity": {

"type": "IAMUser",

"principalId": "AIDAJYDLDBVBI4EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAJL5C75K4ZEXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:18:27Z",

"eventSource": "states.amazonaws.com", "eventName": "DeleteActivity", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "requestParameters": {

"activityArn": "arn:aws:states:us-

east-1:123456789012:activity:testUser.2016-10-27-18-11-35.f017c391-9363-481a-be2e-"

},

"responseElements": null,

"requestID": "490374ea-9cac-11e6-aed5-5b57d226e9ef", "eventID": "e5eb9a3d-13bc-4fa1-9531-232d1914d263", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

#### DeleteStateMachine

The following example shows a CloudTrail log entry that demonstrates the DeleteStateMachine

action:

{

"eventVersion": "1.04", "userIdentity": {

"type": "IAMUser",

"principalId": "AIDAJABK5MNKNAEXAMPLE",

"arn": "arn:aws:iam::123456789012:user/graphene/tests/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAJA2ELRVCPEXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:17:37Z",

"eventSource": "states.amazonaws.com", "eventName": "DeleteStateMachine", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "errorCode": "AccessDenied",

"errorMessage": "User: arn:aws:iam::123456789012:user/graphene/tests/test-user is not authorized to perform: states:DeleteStateMachine on resource: arn:aws:states:us-

east-1:123456789012:stateMachine:testUser.2016-10-27-18-16-38.ec6e261f-1323-4555-9fa", "requestParameters": null,

"responseElements": null,

"requestID": "2cf23f3c-9cac-11e6-aed5-5b57d226e9ef", "eventID": "4a622d5c-e9cf-4051-90f2-4cdb69792cd8", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

#### StartExecution

The following example shows a CloudTrail log entry that demonstrates the StartExecution action:

{

"eventVersion": "1.04", "userIdentity": {

"type": "IAMUser",

"principalId": "AIDAJYDLDBVBI4EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAJL5C75K4ZEXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:17:25Z",

"eventSource": "states.amazonaws.com", "eventName": "StartExecution", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "requestParameters": {

"input": "{}",

"stateMachineArn": "arn:aws:states:us-

east-1:123456789012:stateMachine:testUser.2016-10-27-18-16-26.482bea32-560f-4a36-bd", "name": "testUser.2016-10-27-18-16-26.6e229586-3698-4ce5-8d"

},

"responseElements": {

"startDate": "Oct 28, 2016 1:17:25 AM",

"executionArn": "arn:aws:states:us-

east-1:123456789012:execution:testUser.2016-10-27-18-16-26.482bea32-560f-4a36- bd:testUser.2016-10-27-18-16-26.6e229586-3698-4ce5-8d"

},

"requestID": "264c6f08-9cac-11e6-aed5-5b57d226e9ef", "eventID": "30a20c8e-a3a1-4b07-9139-cd9cd73b5eb8", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

#### StopExecution

The following example shows a CloudTrail log entry that demonstrates the StopExecution action:

{

"eventVersion": "1.04",

"userIdentity": { "type": "IAMUser",

"principalId": "AIDAJYDLDBVBI4EXAMPLE",

"arn": "arn:aws:iam::123456789012:user/test-user", "accountId": "123456789012",

"accessKeyId": "AKIAJL5C75K4ZEXAMPLE",

"userName": "test-user"

},

"eventTime": "2016-10-28T01:18:20Z",

"eventSource": "states.amazonaws.com", "eventName": "StopExecution", "awsRegion": "us-east-1", "sourceIPAddress": "10.61.88.189", "userAgent": "Coral/Netty", "requestParameters": {

"executionArn": "arn:aws:states:us-

east-1:123456789012:execution:testUser.2016-10-27-18-17-00.337b3344-83:testUser.2016-10-27-18-17-00.3a0

},

"responseElements": {

"stopDate": "Oct 28, 2016 1:18:20 AM"

},

"requestID": "4567625b-9cac-11e6-aed5-5b57d226e9ef", "eventID": "e658c743-c537-459a-aea7-dafb83c18c53", "eventType": "AwsApiCall",

"recipientAccountId": "123456789012"

}

# Security

This section provides information about Step Functions security and authentication.

Topics

* + [Authentication (p. 170)](#_bookmark215)
  + [Creating IAM Roles for AWS Step Functions (p. 171)](#_bookmark216)
  + [Creating Granular IAM Permissions for Non-Admin Users (p. 172)](#_bookmark219)
  + [IAM Policies for Integrated Services (p. 174)](#_bookmark224)

Step Functions uses IAM to control access to other AWS services and resources. For an overview of how IAM works, see [Overview of Access Management](https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_access-management.html) in the *IAM User Guide*. For an overview of security credentials, see [AWS Security Credentials](https://docs.aws.amazon.com/general/latest/gr/aws-security-credentials.html) in the Amazon Web Services General Reference.

Authentication

You can access AWS as any of the following types of identities:

* **AWS account root user** – When you ﬁrst create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your ﬁrst IAM user](https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#create-iam-users). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
* **IAM user** – An [IAM user](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html) is an identity within your AWS account that has speciﬁc custom permissions (for example, permissions to create a state machine in Step Functions). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](https://console.aws.amazon.com/), [AWS Discussion Forums](https://forums.aws.amazon.com/), or the [AWS Support Center](https://console.aws.amazon.com/support/home%23/).

In addition to a user name and password, you can also generate [access keys](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several](https://aws.amazon.com/tools/) [SDKs](https://aws.amazon.com/tools/) or by using the [AWS Command Line Interface (CLI)](https://aws.amazon.com/cli/). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don’t use AWS tools, you must sign the request yourself. Step Functions supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html) in the *AWS General Reference*.

* **IAM role** – An [IAM role](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html) is an IAM identity that you can create in your account that has speciﬁc permissions. It is similar to an *IAM user*, but it is not associated with a speciﬁc person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
  + **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as

*federated users*. AWS assigns a role to a federated user when access is requested through an [identity](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers.html) [provider](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers.html). For more information about federated users, see [Federated Users and Roles](https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_access-management.html#intro-access-roles) in the *IAM User Guide*.

* + **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account’s resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-service.html) in the *IAM User Guide*.
  + **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make

it available to all of its applications, you create an instance proﬁle that is attached to the instance. An instance proﬁle contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html) [Applications Running on Amazon EC2 Instances](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html) in the *IAM User Guide*.

## Creating IAM Roles for AWS Step Functions

AWS Step Functions can execute code and access AWS resources (such as invoking an AWS Lambda function). To maintain security, you must grant Step Functions access to those resources by using an IAM role.

The [Tutorials (p. 16)](#_bookmark29) in this guide enable you to take advantage of automatically generated IAM roles that are valid for the region in which you create the state machine. To create your own IAM role for a state machine, follow the steps in this section.

In this example, you create an IAM role with permission to invoke a Lambda function.

### To create a role for Step Functions

1. Sign in to the [IAM console](https://console.aws.amazon.com/iam/home), and then choose **Roles**, **Create role**.
2. On the **Select type of trusted entity** page, under **AWS service**, select **Step Functions** from the list and then choose **Next: Permissions**.
3. On the **Attached permissions policy** page, choose **Next: Review**.
4. On the **Review** page, type StepFunctionsLambdaRole for **Role Name**, and then choose **Create role**.

The IAM role appears in the list of roles.

For more information about IAM permissions and policies, see [Access Management](https://docs.aws.amazon.com/IAM/latest/UserGuide/access.html) in the *IAM User Guide*.

### Attach an Inline Policy

Step Functions can control other services directly in a task state. Attach inline policies to allow Step Functions to access the API actions of the services you need to control.

1. Sign in to the [IAM console](https://console.aws.amazon.com/iam/home), choose **Roles**, search for your Step Functions role, and select that role.
2. Select **Add inline policy**.
3. Use the **Visual editor** or the **JSON** tab to create policies for your role.

For more information on how AWS Step Functions can control other AWS services, see: [AWS Service](#_bookmark107) [Integrations (p. 80)](#_bookmark107).

###### Note

For examples of IAM policies created by the Step Functions console, see [IAM Policies for](#_bookmark224) [Integrated Services (p. 174)](#_bookmark224)

## Creating Granular IAM Permissions for Non-Admin Users

The default managed policies in IAM, such as ReadOnly, don't fully cover all types of Step Functions permissions. This section describes these diﬀerent types of permissions and provides some example conﬁgurations.

AWS Step Functions has four categories of permissions. Depending on what access you want to provide to a user, you can control access by using permissions in these categories.

[Service-Level Permissions (p. 172)](#_bookmark220)

Apply to components of the API that do not act on a speciﬁc resource.

[State Machine-Level Permissions (p. 173)](#_bookmark221)

Apply to all API components that act on a speciﬁc state machine.

[Execution-Level Permissions (p. 173)](#_bookmark222)

Apply to all API components that act on a speciﬁc execution.

[Activity-Level Permissions (p. 173)](#_bookmark223)

Apply to all API components that act on a speciﬁc activity or on a particular instance of an activity.

### Service-Level Permissions

This permission level applies to all API actions that do not act on a speciﬁc resource. These include

[CreateStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateStateMachine.html), [CreateActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_CreateActivity.html), [ListStateMachines](https://docs.aws.amazon.com/step-functions/latest/apireference/API_ListStateMachines.html), and [ListActivities](https://docs.aws.amazon.com/step-functions/latest/apireference/API_ListActivities.html).

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"states:ListStateMachines", "states:ListActivities", "states:CreateStateMachine", "states:CreateActivity"

],

"Resource": [ "arn:aws:states:\*:\*:\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [

"arn:aws:iam:::role/my-execution-role"

]

}

]

}

### State Machine-Level Permissions

This permission level applies to all API actions that act on a speciﬁc state machine. These API require the ARN of the state machine as part of the request, such as [DeleteStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DeleteStateMachine.html), [DescribeStateMachine](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DescribeStateMachine.html), [StartExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html), and [ListExecutions](https://docs.aws.amazon.com/step-functions/latest/apireference/API_ListExecutions.html).

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"states:DescribeStateMachine", "states:StartExecution", "states:DeleteStateMachine", "states:ListExecutions", "states:UpdateStateMachine"

],

"Resource": [ "arn:aws:states:\*:\*:stateMachine:StateMachinePrefix\*"

]

}

]

}

### Execution-Level Permissions

This permission level applies to all the API actions that act on a speciﬁc execution. These API operations require the ARN of the execution as part of the request, such as [DescribeExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DescribeExecution.html), [GetExecutionHistory](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetExecutionHistory.html), and [StopExecution](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StopExecution.html).

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"states:DescribeExecution", "states:DescribeStateMachineForExecution", "states:GetExecutionHistory", "states:StopExecution"

],

"Resource": [ "arn:aws:states:\*:\*:execution:\*:ExecutionPrefix\*"

]

}

]

}

### Activity-Level Permissions

This permission level applies to all the API actions that act on a speciﬁc activity or on a particular instance of it. These API operations require the ARN of the activity or the token of the instance as part of

the request, such as [DeleteActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DeleteActivity.html), [DescribeActivity](https://docs.aws.amazon.com/step-functions/latest/apireference/API_DescribeActivity.html), [GetActivityTask](https://docs.aws.amazon.com/step-functions/latest/apireference/API_GetActivityTask.html), [SendTaskSuccess](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskSuccess.html), [SendTaskFailure](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskFailure.html), and [SendTaskHeartbeat](https://docs.aws.amazon.com/step-functions/latest/apireference/API_SendTaskHeartbeat.html).

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"states:DescribeActivity", "states:DeleteActivity", "states:GetActivityTask", "states:SendTaskSuccess", "states:SendTaskFailure", "states:SendTaskHeartbeat"

],

"Resource": [ "arn:aws:states:\*:\*:activity:ActivityPrefix\*"

]

}

]

}

## IAM Policies for Integrated Services

When you create a state machine in the AWS Step Functions console, Step Functions will produce an IAM policy based on the resources used in your state machine deﬁnition.

These examples show how Step Functions generates an IAM policy based on your state machine deﬁnition. Items in the example code such as *[[resourceName]]* are replaced with the static resources listed in your state machine deﬁnition. If you have multiple static resources, there will be an entry for each in the IAM role.

### Dynamic vs. Static Resources

Static resources are deﬁned directly in the task state of your state machine. When you include the information about the API actions you call directly in your task states, Step Functions creates an IAM role for only those resources.

Dynamic resources are those that are passed in to your state input, and accessed using a Path (see, [Paths (p. 143)](#_bookmark183)). If you are passing dynamic resources to your task, Step Functions will create a more privileged policy that speciﬁes: "Resource": "\*".

### Synchronous vs. Asynchronous IAM Policies

For synchronous connections (those ending in .sync), additional permissions are needed to monitor and receive a response from the API actions of connected services. The related policies need more permissions than non-synchronous connected services. See [Connect to Resources (p. 80)](#_bookmark108) for information on synchronous connections.

###### Note

Review these templates to understand how Step Functions creates your IAM policies, and as an example of how to manually create IAM policies for Step Functions when working with other AWS services. For more information on Step Functions service integration, see: [AWS Service](#_bookmark107) [Integrations (p. 80)](#_bookmark107)

Topics

* + [IAM Policy for AWS Lambda (p. 175)](#_bookmark227)
  + [AWS Batch (p. 175)](#_bookmark228)
  + [Amazon DynamoDB (p. 176)](#_bookmark229)
  + [Amazon Elastic Container Service/Fargate (p. 177)](#_bookmark230)
  + [Amazon Simple Notiﬁcation Service (p. 179)](#_bookmark231)
  + [Amazon Simple Queue Service (p. 180)](#_bookmark232)
  + [AWS Glue (p. 180)](#_bookmark233)
  + [Amazon SageMaker (p. 181)](#_bookmark234)
  + [Activities or no Tasks (p. 188)](#_bookmark237)

### IAM Policy for AWS Lambda

AWS Step Functions will generate an IAM policy based on your state machine deﬁnition. For a state machine with two Lambda task states that call function1 and function2, a policy with lambda:Invoke permissions for the two functions must be used. For example:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"lambda:InvokeFunction"

],

"Resource": [

"arn:aws:lambda:*[[region]]*:*[[accountId]]*:function:*[[function1]]*",

"arn:aws:lambda:*[[region]]*:*[[accountId]]*:function:*[[function2]]*"

]

}

]

}

### AWS Batch

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

AWS Batch does not support resource level access control. You must use "Resource": "\*". Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"batch:SubmitJob", "batch:DescribeJobs", "batch:TerminateJob"

],

"Resource": "\*"

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:*[[accountId]]*:rule/

StepFunctionsGetEventsForBatchJobsRule"

]

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"batch:SubmitJob"

],

"Resource": "\*"

}

]

}

### Amazon DynamoDB

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

*Static resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"dynamodb:GetItem", "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem"

],

"Resource": [

"arn:aws:dynamodb:*[[region]]*:*[[accountId]]*:table/*[[tableName]]*"

]

}

]

}

*Dynamic resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"dynamodb:GetItem", "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem"

],

"Resource": "\*"

}

]

}

### Amazon Elastic Container Service/Fargate

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

Since the value for TaskId is not known until the task is submitted, Step Functions creates a more privileged "Resource": "\*" policy.

###### Note

You can only stop Amazon ECS tasks that were started by Step Functions, despite the "\*" IAM policy.

Synchronous

*Static resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"ecs:RunTask"

],

"Resource": [

"arn:aws:ecs:*[[region]]*:

*[[accountId]]*:task-definition/*[[taskDefinition]]*"

]

},

{

"Effect": "Allow", "Action": [

"ecs:StopTask", "ecs:DescribeTasks"

],

"Resource": "\*"

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:

*[[accountID]]*:rule/StepFunctionsGetEventsForECSTaskRule"

]

}

]

}

*Dynamic resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"ecs:RunTask", "ecs:StopTask", "ecs:DescribeTasks"

],

"Resource": "\*"

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:

*[[accountId]]*:rule/StepFunctionsGetEventsForECSTaskRule"

]

}

]

}

Asynchronous

*Static resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"ecs:RunTask"

],

"Resource": [

"arn:aws:ecs:*[[region]]*:

*[[accountID]]*:task-definition/*[[taskDefinition]]*"

]

}

]

}

*Dynamic resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"ecs:RunTask"

],

"Resource": "\*"

}

]

}

### Amazon Simple Notiﬁcation Service

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

*Static resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sns:Publish"

],

"Resource": [

"arn:aws:sns:*[[region]]*:*[[accountId]]*:*[[topicName]]*"

]

}

]

}

*Resources based on a Path, or publishing to TargetArn or PhoneNumber:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sns:Publish"

],

"Resource": "\*"

}

]

}

### Amazon Simple Queue Service

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

*Static resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sqs:SendMessage"

],

"Resource": [

"arn:aws:sqs:*[[region]]*:*[[accountId]]*:*[[queueName]]*"

]

}

]

}

*Dynamic resources:*

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sqs:SendMessage"

],

"Resource": "\*"

}

]

}

### AWS Glue

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

AWS Glue does not have resource-based control. Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"glue:StartJobRun", "glue:GetJobRun", "glue:GetJobRuns", "glue:BatchStopJobRun"

],

"Resource": "\*"

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"glue:StartJobRun"

],

"Resource": "\*"

}

]

}

### Amazon SageMaker

These example templates show how AWS Step Functions generates IAM policies based on the resources in your state machine deﬁnition. For more information see:

* [IAM Policies for Integrated Services (p. 174)](#_bookmark224)
* [AWS Service Integrations (p. 80)](#_bookmark107)

###### Note

For these examples, *[[roleArn]]* refers to the Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker uses to access model artifacts and docker images for deployment

on ML compute instances, or for batch transform jobs. For more information, see [Amazon](https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-roles.html) [SageMaker Roles](https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-roles.html).

Topics

* + [CreateTrainingJob (p. 181)](#_bookmark235)
  + [CreateTransformJob (p. 185)](#_bookmark236)

CreateTrainingJob

*Static resources:*

Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTrainingJob", "sagemaker:DescribeTrainingJob", "sagemaker:StopTrainingJob"

],

"Resource": [

"arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:training-job/*[[trainingJobName]]*\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:*[[accountId]]*:rule/

StepFunctionsGetEventsForSageMakerTrainingJobsRule"

]

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTrainingJob"

],

"Resource": [

"arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:training-job/*[[trainingJobName]]*\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

}

]

}

*Dynamic resources:*

Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTrainingJob", "sagemaker:DescribeTrainingJob", "sagemaker:StopTrainingJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:training-job/\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": {

"StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:*[[accountId]]*:rule/

StepFunctionsGetEventsForSageMakerTrainingJobsRule"

]

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTrainingJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:training-job/\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

}

]

}

#### CreateTransformJob

###### Note

AWS Step Functions will not automatically create a policy for CreateTransformJob when you create a state machine that integrates with Amazon SageMaker. You must attach an inline policy to the created role based on one of the following IAM examples.

*Static resources:*

Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTransformJob", "sagemaker:DescribeTransformJob", "sagemaker:StopTransformJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:transform-

job/*[[transformJobName]]*\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:*[[accountId]]*:rule/

StepFunctionsGetEventsForSageMakerTransformJobsRule"

]

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTransformJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:transform-

job/*[[transformJobName]]*\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

}

]

}

*Dynamic resources:*

Synchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTransformJob", "sagemaker:DescribeTransformJob", "sagemaker:StopTransformJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:transform-job/\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [ "*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

},

{

"Effect": "Allow", "Action": [

"events:PutTargets", "events:PutRule", "events:DescribeRule"

],

"Resource": [ "arn:aws:events:*[[region]]*:*[[accountId]]*:rule/

StepFunctionsGetEventsForSageMakerTransformJobsRule"

]

}

]

}

Asynchronous

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow", "Action": [

"sagemaker:CreateTransformJob"

],

"Resource": [ "arn:aws:sagemaker:*[[region]]*:*[[accountId]]*:transform-job/\*"

]

},

{

"Effect": "Allow", "Action": [

"sagemaker:ListTags"

],

"Resource": [ "\*"

]

},

{

"Effect": "Allow", "Action": [

"iam:PassRole"

],

"Resource": [

Activities

"*[[roleArn]]*"

],

"Condition": { "StringEquals": {

"iam:PassedToService": "sagemaker.amazonaws.com"

}

}

}

]

}

### Activities or no Tasks

For a state machine that has only Activity tasks, or no tasks at all, use an IAM policy that denies access to all actions and resources.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Deny",

"Action": "\*",

"Resource": "\*"

}

]

}

For more information on using activity tasks, see: [Activities (p. 72)](#_bookmark101).

# Related Step Functions Resources

The following table lists related resources that you might ﬁnd useful as you work with this service.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [AWS Step Functions API Reference](https://docs.aws.amazon.com/step-functions/latest/apireference/) | Descriptions of API actions, parameters, and data types and a list of errors that the service returns. |
| [AWS Step Functions Command Line Reference](https://docs.aws.amazon.com/cli/latest/reference/stepfunctions/) | Descriptions of the AWS CLI commands that you can use to work with AWS Step Functions. |
| [Product information for Step Functions](https://aws.amazon.com/step-functions) | The primary web page for information about Step Functions. |
| [Discussion Forums](https://forums.aws.amazon.com/) | A community-based forum for developers to discuss technical questions related to Step Functions and other AWS services. |
| [AWS Premium Support Information](https://aws.amazon.com/premiumsupport/) | The primary web page for information about AWS Premium Support, a one-on-one, fast- response support channel to help you build and run applications on AWS infrastructure services. |

# Document History

This section lists major changes to the *AWS Step Functions Developer Guide*.

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| New feature | Step Functions Local is now available. You can run Step Functions on your local machine for testing and development. Step Functions Local is available for download as either a Java application, or as a Docker image. See, [Setting](#_bookmark9) [Up Step Functions Local (Downloadable Version) (p. 5)](#_bookmark9). | February 4, 2019 |
| New feature | Step Functions supports resource tagging to help track your cost allocation. You can tag state machines on the **Details** page, or through API actions. See, [Tagging (p. 112)](#_bookmark145). | January 7,  2019 |
| New feature | AWS Step Functions is now available in the EU (Paris), and South America (São Paulo) regions. See [Supported Regions (p. 1)](#_bookmark2). | December 13, 2018 |
| New feature | AWS Step Functions is now available the EU (Stockholm) region. See [Supported Regions (p. 1)](#_bookmark2) for a list of supported regions. | December 12, 2018 |
| New feature | You can now easily conﬁgure and generate a state deﬁnition for integrated services when editing your state deﬁnition. For more information, see:   * [Code Snippets (p. 82)](#_bookmark113) * [Using Code Snippets (p. 66)](#_bookmark94) | December 10, 2018 |
| New feature | Step Functions now integrates with some AWS services. You can now directly call and pass parameters to the API of these integrated services from a task state in the Amazon States Language. For more information, see:   * [AWS Service Integrations (p. 80)](#_bookmark107) * [Pass Parameters to a Service API (p. 81)](#_bookmark110) * [Supported AWS Service Integrations for Step Functions (p. 83)](#_bookmark114) | November 29, 2018 |
| Update | Improved the description of TimeoutSeconds and HeartbeatSeconds in the documentation for task states. See [Task (p. 132)](#_bookmark171). | October 24, 2018 |
| Update | Improved the description for the *Maximum execution history size* limit and provided a link to the related best practices topic.   * [Limits Related to State Machine Executions (p. 153)](#_bookmark198) * [Avoid Reaching the History Limit (p. 150)](#_bookmark192) | October 17, 2018 |
| Update | Added a new tutorial to the AWS Step Functions documentation: See [Starting a State Machine Execution in Response to Amazon S3 Events (p. 42)](#_bookmark69). | September 25, 2018 |
| Update | Removed the entry *Maximum executions displayed in Step Functions console*  from the limits documentation. See [Limits (p. 152)](#_bookmark195). | September 13, 2018 |
| Update | Added a best practices topic to the AWS Step Functions documentation on improving latency when polling for activity tasks. See [Avoid Latency When](#_bookmark194) [Polling for Activity Tasks (p. 151)](#_bookmark194). | August 30,  2018 |

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Improved the AWS Step Functions topic on activities and activity workers. See [Activities (p. 72)](#_bookmark101). | August 29,  2018 |
| Update | Improved the AWS Step Functions topic on CloudTrail integration. See [Logging Step Functions using AWS CloudTrail (p. 164)](#_bookmark211). | August 7,  2018 |
| Update | Added JSON examples to AWS CloudFormation tutorial. See [Creating a](#_bookmark47) [Lambda State Machine Using AWS CloudFormation (p. 22)](#_bookmark47). | June 23,  2018 |
| Update | Added a new topic on handling Lambda service errors. See [Handle Lambda](#_bookmark193) [Service Exceptions (p. 151)](#_bookmark193). | June 20,  2018 |
| New feature | AWS Step Functions is now available the Asia Paciﬁc (Mumbai) region. See [Supported Regions (p. 1)](#_bookmark2) for a list of supported regions. | June 28,  2018 |
| New feature | AWS Step Functions is now available the AWS GovCloud (US-West) region. See [Supported Regions (p. 1)](#_bookmark2) for a list of supported regions. For information about using Step Functions in the AWS GovCloud (US-West) Region, see [AWS GovCloud (US) Endpoints](https://docs.aws.amazon.com/govcloud-us/latest/UserGuide/using-govcloud-endpoints.html). | June 28,  2018 |
| Update | Improved documentation on error handling for Parallel states. See [Error](#_bookmark181) [Handling (p. 143)](#_bookmark181). | June 20,  2018 |
| Update | Improved documentation about Input and Output processing in Step Functions. Learn how to use InputPath, ResultPath, and OutputPath to control the ﬂow of JSON through your workﬂows, states, and tasks. See:   * [Input and Output Processing in Step Functions (p. 94)](#_bookmark128) * [ResultPath (p. 97)](#_bookmark130) | June 7,  2018 |
| Update | Improved code examples for parallel states. See [Parallel (p. 140)](#_bookmark180). | June 4,  2018 |
| New feature | You can now monitor API and Service metrics in CloudWatch. See [Monitoring](#_bookmark205) [Step Functions Using CloudWatch (p. 157)](#_bookmark205). | May 25,  2018 |
| Update | StartExecution, StopExecution, and StateTransition now have increased throttling limits in the following regions:   * US East (N. Virginia) * US West (Oregon) * EU (Ireland)   For more information see [Limits (p. 152)](#_bookmark195). | May 16,  2018 |
| New feature | AWS Step Functions is now available the US West (N. California) and Asia Paciﬁc (Seoul) regions. See [Supported Regions (p. 1)](#_bookmark2) for a list of supported regions. | May 5,  2018 |
| Update | Updated procedures and images to match changes to the interface. | April 25,  2018 |

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Added a new tutorial that shows how to start a new execution to continue your work. See [Continue as a New Execution (p. 58)](#_bookmark87). This tutorial describes a design pattern that can help avoid some service limitations. See, [Avoid](#_bookmark192) [Reaching the History Limit (p. 150)](#_bookmark192). | April 19,  2018 |
| Update | Improved introduction to states documentation by adding conceptual information about state machines. See [States (p. 71)](#_bookmark99). | March 9,  2018 |
| Update | In addition to HTML, PDF, and Kindle, the AWS Step Functions Developer Guide is available on GitHub. To leave feedback, choose the GitHub icon in the upper right-hand corner. | March 2,  2018 |
| Update | Added a topic describing other resources relating to Step Functions.  See [Related Step Functions Resources (p. 189)](#_bookmark238). | February 20, 2018 |
| New feature | * When you create a new state machine, you must acknowledge that AWS Step Functions will create an IAM role which allows access to your Lambda functions. * Updated the following tutorials to reﬂect the minor changes in the state machine creation workﬂow:   + [Getting Started (p. 12)](#_bookmark21)   + [Creating a Lambda State Machine (p. 18)](#_bookmark38)   + [Creating an Activity State Machine (p. 30)](#_bookmark51)   + [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58)   + [Iterating a Loop Using Lambda (p. 51)](#_bookmark81) | February 19, 2018 |
| Update | Added a topic that describes an example activity worker written in Ruby. This implementation can be used to create a Ruby activity worker directly, or as a design pattern for creating an activity worker in another language.  See [Example Activity Worker in Ruby (p. 74)](#_bookmark106). | February 6, 2018 |
| Update | Added a new tutorial describing a design pattern that uses a Lambda function to iterate a count.  See [Creating a Lambda State Machine (p. 18)](#_bookmark38). | January 31, 2018 |
| Update | Updated content on IAM permissions to include  DescribeStateMachineForExecution and UpdateStateMachine APIs.  See [Creating Granular IAM Permissions for Non-Admin Users (p. 172)](#_bookmark219). | January 26, 2018 |
| Update | Added newly available regions: Canada (Central), Asia Paciﬁc (Singapore).  See [Supported Regions (p. 1)](#_bookmark2). | January 25, 2018 |

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Updated tutorials and procedures to reﬂect that IAM allows you to select  *Step Functions* as a role. | January 24, 2018 |
| Update | Added a new *Best Practices* topic that suggests not passing large payloads between states.  See [Use ARNs Instead of Passing Large Payloads (p. 150)](#_bookmark191). | January 23, 2018 |
| New Feature | Corrected procedures to match updated interface for creating a state machine:   * [Getting Started (p. 12)](#_bookmark21) * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * [Creating an Activity State Machine (p. 30)](#_bookmark51) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) | January 17, 2018 |
| New Feature | You can use *Sample Projects* to quickly provision state machines and all related AWS resources. See [Sample Projects (p. 114)](#_bookmark149),  Available sample projects include:   * [Poll for Job Status (Lambda, Batch),](#_bookmark162) [(p. 123)](#_bookmark162) * [Task Timer (p. 126)](#_bookmark164)   **Note**  These sample projects and related documentation replace tutorials that described implementing the same functionality. | January 11, 2018 |
| Update | Added a *Best Practices* section that includes information on avoiding stuck executions. See [Best Practices for Step Functions (p. 150)](#_bookmark189). | January 5,  2018 |
| Update | Added a note on how retries can aﬀect pricing:  **Note**  Retries are treated as state transitions. For information on how state transitions aﬀect billing, see [Step Functions Pricing](https://aws.amazon.com/step-functions/pricing/). | December 8, 2017 |
| Update | Added information related to resource names:  **Note**  Step Functions allows you to create state machine, execution, and activity names that contain non-ASCII characters. These non-ASCII names don't work with Amazon CloudWatch. To ensure that you can track CloudWatch metrics, choose a name that uses only ASCII characters. | December 6, 2017 |
| Update | Improved security overview information and added a topic on granular IAM permissions. See [Security (p. 170)](#_bookmark214) and [Creating Granular IAM Permissions for](#_bookmark219) [Non-Admin Users (p. 172)](#_bookmark219). | November 27, 2017 |
| New Feature | You can update an existing state machine. See [Update a State](#_bookmark26) [Machine (p. 14)](#_bookmark26). | November 15, 2017 |

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Added a note to clarify Lambda.Unknown errors and linked to the Lambda documentation in the following sections:   * [Error Names (p. 104)](#_bookmark138) * [To create the state machine (p. 36)](#_bookmark64)   **Note**  Unhandled errors in Lambda are reported as Lambda.Unknown in the error output. These include out-of-memory errors, function timeouts, and hitting the concurrent Lambda invoke limit. You can match on Lambda.Unknown, States.ALL,  or States.TaskFailed to handle these errors. For more information about Lambda Handled and Unhandled errors, see FunctionError in the [AWS Lambda Developer Guide](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html#API_Invoke_ResponseSyntax). | October 17, 2017 |
| Update | Corrected and clariﬁed IAM instructions and updated the screenshots in all [tutorials (p. 16)](#_bookmark29). | October 11, 2017 |
| Update | * Added new screenshots for state machine execution results to reﬂect changes in the Step Functions console. Rewrote the Lambda instructions in the following tutorials to reﬂect changes in the Lambda console:   + [Creating a Lambda State Machine (p. 18)](#_bookmark38)   + Creating a Job Status Poller   + Creating a Task Timer   + [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) * Corrected and clariﬁed information about creating state machines in the following sections:   + [Getting Started (p. 12)](#_bookmark21)   + [Creating an Activity State Machine (p. 30)](#_bookmark51) | October 6,  2017 |
| Update | Rewrote the IAM instructions in the following sections to reﬂect changes in the IAM console:   * [Creating IAM Roles for AWS Step Functions (p. 171)](#_bookmark216) * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * Creating a Job Status Poller * Creating a Task Timer * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) * [Creating a Step Functions API Using API Gateway (p. 47)](#_bookmark76) | October 5,  2017 |
| Update | Rewrote the [State Machine Data (p. 92)](#_bookmark124) section. | September 28, 2017 |
| New feature | The [limits related to API action throttling (p. 154)](#_bookmark200) are increased for all regions where Step Functions is available. | September 18, 2017 |
| Update | * Corrected and clariﬁed information about starting new executions in all tutorials. * Corrected and clariﬁed information in the [Limits Related to Accounts (p. 153)](#_bookmark197) section. | September 14, 2017 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Change** | **Description** | | | **Date Changed** |
| Update | Corrected and clariﬁed information in the [Templates (p. 111)](#_bookmark144) section. | | | September 1, 2017 |
| Update | Rewrote the following tutorials to reﬂect changes in the Lambda console:   * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) * Creating a Job Status Poller | | | August 28,  2017 |
| New feature | Step Functions is available in EU (London). | | | August 23,  2017 |
| New feature | The visual workﬂows of state machines let you zoom in, zoom out, and center the graph. | | | August 21,  2017 |
| New feature | **Important**  An execution can't use the name of another execution for 90 days.  When you make multiple StartExecution calls with the same name, the new execution doesn't run and the following rules apply. | | | August 18,  2017  nAlreadyExi sltrseadyExi |
| **Input Type** | **Execution State** | |
| Open | Closed |
| Identical | Success | Executio |
| Diﬀerent | Execution | AElxrecaudtyiEoxniA |
| For more information, see the [name](https://docs.aws.amazon.com/step-functions/latest/apireference/API_StartExecution.html#API_StartExecution_RequestParameters) request parameter of the  StartExecution API action in the *AWS Step Functions API Reference*. | | |
| Update | Added information about an alternative way of passing the state machine ARN to the [Creating a Step Functions API Using API Gateway (p. 47)](#_bookmark76) tutorial. | | | August 17,  2017 |
| Update | Added the new *Creating a Job Status Poller* tutorial. | | | August 10,  2017 |
| New feature | * Step Functions emits the ExecutionThrottled CloudWatch metric. For more information, see [State Machine Metrics (p. 158)](#_bookmark208). * Added the [Limits Related to State Throttling (p. 155)](#_bookmark201) section. | | | August 3,  2017 |
| Update | Updated the instructions in the [To create a role for API Gateway (p. 47)](#_bookmark78) section. | | | July 18,  2017 |
| Update | Corrected and clariﬁed information in the [Choice (p. 135)](#_bookmark175) section. | | | June 23,  2017 |
| Update | Added information about using resources under other AWS accounts to the following tutorials:   * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * [Creating a Lambda State Machine Using AWS CloudFormation (p. 22)](#_bookmark47) * [Creating an Activity State Machine (p. 30)](#_bookmark51) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) | | | June 22,  2017 |

sts sts

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Corrected and clariﬁed information in the following sections:   * [Getting Started (p. 12)](#_bookmark21) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) * [States (p. 130)](#_bookmark168) * [Error Handling (p. 104)](#_bookmark138) | June 21,  2017 |
| Update | Rewrote all tutorials to match the Step Functions console refresh. | June 12,  2017 |
| New feature | Step Functions is available in Asia Paciﬁc (Sydney). | June 8,  2017 |
| Update | Restructured the [Amazon States Language (p. 128)](#_bookmark165) section. | June 7,  2017 |
| Update | Corrected and clariﬁed information in the [Creating an Activity State](#_bookmark51) [Machine (p. 30)](#_bookmark51) section. | June 6,  2017 |
| Update | Corrected the code examples in the [Examples Using Retry and Using](#_bookmark142) [Catch (p. 108)](#_bookmark142) section. | June 5,  2017 |
| Update | Restructured this guide using AWS documentation standards. | May 31,  2017 |
| Update | Corrected and clariﬁed information in the [Parallel (p. 140)](#_bookmark180) section. | May 25,  2017 |
| Update | Merged the Paths and Filters sections into the [Input and Output](#_bookmark182) [Processing (p. 143)](#_bookmark182) section. | May 24,  2017 |
| Update | Corrected and clariﬁed information in the [Templates (p. 111)](#_bookmark144) section. | May 16,  2017 |
| Update | Corrected and clariﬁed information in the [Monitoring Step Functions Using](#_bookmark205) [CloudWatch (p. 157)](#_bookmark205) section. | May 15,  2017 |
| Update | Updated the GreeterActivities.java worker code in the [Creating an](#_bookmark51) [Activity State Machine (p. 30)](#_bookmark51) tutorial. | May 9,  2017 |
| Update | Added an introductory video to the [What Is AWS Step Functions? (p. 1)](#_bookmark0) section. | April 19,  2017 |
| Update | Corrected and clariﬁed information in the following tutorials:   * [Getting Started (p. 12)](#_bookmark21) * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * [Creating an Activity State Machine (p. 30)](#_bookmark51) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) | April 19,  2017 |
| Update | Added information about Lambda templates to the [Creating a Lambda State](#_bookmark38) [Machine (p. 18)](#_bookmark38) and [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) tutorials. | April 6,  2017 |

|  |  |  |
| --- | --- | --- |
| **Change** | **Description** | **Date Changed** |
| Update | Changed the "Maximum input or result data size" limit to "Maximum input or result data size for a task, state, or execution" (32,768 characters). For more information, see [Limits Related to Task Executions (p. 153)](#_bookmark199). | March 31,  2017 |
| New feature | * Step Functions supports executing state machines by setting Step Functions as Amazon CloudWatch Events targets. * Added the [Periodically Start a State Machine Execution Using CloudWatch Events (p. 39)](#_bookmark66) tutorial. | March 21,  2017 |
| New feature | * Step Functions allows Lambda function error handling as the preferred error handling method. * Updated the [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) tutorial and the [Error Handling (p. 104)](#_bookmark138) section. | March 16,  2017 |
| New feature | Step Functions is available in EU (Frankfurt). | March 7,  2017 |
| Update | Reorganized the topics in the table of contents and updated the following tutorials:   * [Getting Started (p. 12)](#_bookmark21) * [Creating a Lambda State Machine (p. 18)](#_bookmark38) * [Creating an Activity State Machine (p. 30)](#_bookmark51) * [Handling Error Conditions Using a State Machine (p. 34)](#_bookmark58) | February 23, 2017 |
| New feature | * The **State Machines** page of the Step Functions console includes the **Copy to New** and **Delete** buttons. * Updated the screenshots to match the console changes. | February 23, 2017 |
| New feature | * Step Functions supports creating APIs using API Gateway. * Added the [Creating a Step Functions API Using API Gateway (p. 47)](#_bookmark76) tutorial. | February 14, 2017 |
| New feature | * Step Functions supports integration with AWS CloudFormation. * Added the [Creating a Lambda State Machine Using AWS CloudFormation (p. 22)](#_bookmark47) tutorial. | February 10, 2017 |
| Update | Clariﬁed the current behavior of the ResultPath and OutputPath ﬁelds in relation to Parallel states. | February 6, 2017 |
| Update | * Clariﬁed state machine naming restrictions in tutorials. * Corrected some code examples. | January 5,  2017 |
| Update | Updated Lambda function examples to use the latest programming model. | December 9, 2016 |
| New feature | The initial release of Step Functions. | December 1, 2016 |

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](https://docs.aws.amazon.com/general/latest/gr/glos-chap.html) in the *AWS General Reference*.