aws

開発者ガイド

AWS Step Functions

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Step Functions: 開発者ガイド

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有しない他の商標はすべてそれぞれの所有者に帰属します。所有者は必ずしも Amazon との提携や関連があるわけではありません。また、Amazon の支援を受けているとはかぎりません。

Table of Contents

とは何ですか AWS Step Functions? 1
AWS SDK と最適化されたインテグレーション 2
Standard ワークフローと Express ワークフロー 2
標準ワークフロー仕様 2
Express ワークフローの仕様 2
ユースケース 3
ユースケース #1: 関数オーケストレーション 3
ユースケース #2: 分岐 4
ユースケース #3: エラー処理 4
ユースケース #4: ヒューマンインザループ 5
ユースケース #5: 並列処理 6
ユースケース #6: 動的並列処理 6
サービス統合 6
サポートされるリージョン 10
Step Functions を使うのは今回が初めてですか? 11
前提条件 12
ステップ 1: アカウントと IAM ユーザーを作成する 12
AWS アカウントへのサインアップ 12
管理ユーザーの作成 13
ステップ 2: プログラム的なアクセス権を付与する 14
開始方法 16
主要なコンセプト 16
このシリーズのチュートリアル 18
チュートリアル 1: ステートマシンのプロトタイプを作成する 21
次のステップ 23
チュートリアル 2: Lambda 関数を使用して最初のサービス統合を定義する 23
ステップ 1: Lambda 関数を作成およびテストする 23
ステップ 2: ワークフローの更新 - [クレジット限度額を取得] 状態を設定する 24
次のステップ 25
チュートリアル 3: ワークフローに if-else 条件を実装する 25
ステップ 1: コールバックトークンを受け取る Amazon SNS トピックを作成する 26
ステップ 2: コールバックを処理する Lambda 関数を作成する 26
ステップ 3: ワークフローの更新 - 選択状態に if-else 条件ロジックを追加する 29
次のステップ 31

チュートリアル 4: 並列で実行する複数のタスクを定義する 31
ステップ 1: Lambda 関数を作成して必要な確認を行う 32
ステップ 2: ワークフローの更新 - 並列で実行するタスクを追加する 34
チュートリアル 5: 項目のコレクションを同時に反復処理する 35
ステップ 1: すべての信用調査機関の名前を保存する DynamoDB テーブルを作成する 36
ステップ 2: ステートマシンの更新 - DynamoDB テーブルから結果を取得する 36
ステップ 3: すべての信用調査機関のクレジットスコアを返す Lambda 関数を作成する 37
ステップ 4: ステートマシンの更新 - マップステートを追加してクレジットスコアを反復取得する 37
チュートリアル 6: ワークフローを保存してステートマシンを実行する 38
ステップ 1: ステートマシンを保存する 38
ステップ 2: 残りの IAM ポリシーを追加する 40
ステップ 3: ステートマシンを実行する 40
チュートリアル 7: 入力と出力を構成する 41
フィルターを使用して、 InputPath 未加工の入力の特定の部分を選択します 43
パラメータフィルタを使用して選択した入力を操作する 46
ResultSelector、ResultPath、OutputPath フィルターを使用して出力を設定する 47
チュートリアル 8: コンソールでエラーをデバッグする 50
無効なパス選択状態のエラーのデバッグ 50
入力フィルターと出力フィルターの適用中の JSON パス式エラーのデバッグ 52
ユースケース 55
データ処理 55
Machine Learning 56
マイクロサービスのオーケストレーション 58
IT およびセキュリティのオートメーション 59
Step Functions 仕組み 61
標準ワークフロー対 Express ワークフロー 61
同期および非同期 Express ワークフロー 64
実行の保証 65
Express ワークフローによるコスト最適化 67
状態 70
Amazon ステートメント言語 72
パス 92
タスク 94
選択 115
待機 121

成功 ... 123
失敗 ... 124
並行 ... 126
マッピング ... 130
マップステート処理モード ... 131
インラインモードと分散モードの違い ... 132
インラインモードでのマップステートの使用 ... 134
分散マップ状態の使用 ... 143
分散マップ状態の許容される失敗しきい値 ... 154
Transitions ... 156
分散マップの状態の遷移 ... 157
ステートマシンデータ ... 158
データ形式 ... 158
ステートマシンの入出力 ... 159
状態の入出力 ... 159
入力および出力処理 ... 161
パス ... 163
InputPath、パラメータ、 ResultSelector ... 165
ResultPath ... 171
OutputPath ... 180
InputPath、ResultPath、OutputPath の例 ... 181
マップステートの入力および出力フィールド ... 186
コンテキストオブジェクト ... 218
データフローシミュレータ ... 224
データフローシミュレータの使用 ... 225
データフローシミュレーターの考慮事項 ... 226
バージョニングとエイリアス ... 228
バージョン ... 229
エイリアス ... 232
バージョニングとエイリアスの認証 ... 236
バージョンまたはエイリアスへの実行の関連付け ... 238
デプロイの例 ... 242
バージョンの段階的なデプロイ ... 244
実行 ... 254
タスクからの実行の開始 ... 255
EventBridge スケジューラの使用 ... 257

Standard ワークフローと Express ワークフローの実行 264
実行の表示とデバッグ 269
Redriving の実行 291
マップ実行の確認 301
エラー処理 313
エラー名 314
エラー後の再試行 317
Fallback 状態 321
Retry と Catch を使用するステートマシンの例 323
Step Functions 呼び出し 328
読み込み整合性 329
Step Functions でのタグ付け 329
コスト割り当てのタグ付け 330
セキュリティのためのタグ付け 330
表示と管理 331
タグ付け API 332
Workflow Studio 333
インターフェースの概要 334
デザインモード 335
コードモード 341
設定モード 345
キーボードショートカット 349
Workflow Studio を使用する 349
ワークフローの作成 350
ワークフローの設計 352
ワークフローを実行 359
ワークフローの編集 360
ワークフローのエクスポート 362
ワークフロープロトタイプの作成 363
入力と出力を構成する 364
状態へ入力を構成する 365
状態の出力を構成 368
Workflow Studio での実行ロール 373
自動生成されたロールについて 374
ロールを自動生成する 374
ロール生成に関する問題を解決する 376

Workflow Studio で HTTP タスクをテストするためのロール 377
Workflow Studio で最適化されたサービス統合をテストするためのロール 377
Workflow Studio で AWS SDK サービス統合をテストするためのロール 378
Workflow Studio でフローステートをテストするためのロール 378
エラー処理 379
エラーを再試行 380
エラーの捕捉 381
タイムアウト 381
HeartbeatSeconds 381
チュートリアル: AWS Step Functions Workflow Studio 使用を学ぶ 382
ステップ 1: Workflow Studio に移動する 383
ステップ 2: ステートマシンを作成する 383
ステップ 3: 自動生成された Amazon States Language 定義を確認する 385
ステップ 4: コードモードでワークフローの定義を編集する 387
ステップ 5: ステートマシンを保存する 389
ステップ 6: ステートマシンを実行する 390
ステップ 7: ステートマシンを更新する 391
ステップ 8: クリーンアップする 392
チュートリアル 394
Lambda を使用する Step Functions ステートマシン状態を作成する 394
ステップ 1: Lambda 関数を作成する 395
ステップ 2: Lambda 関数をテストする 396
ステップ 3: ステートマシンを作成する 397
ステップ 4: ステートマシンを実行する 399
ステートマシンを使用してエラー条件を処理する 401
ステップ 1: 失敗する Lambda 関数を作成する 401
ステップ 2: Lambda 関数をテストする 402
ステップ 3: Catch フィールドを使用するステートマシンを作成する 403
ステップ 4: ステートマシンを実行する 405
インラインマップステートを使用したアクションの反復 406
ステップ 1: ワークフロープロトタイプを作成する 407
ステップ 2: 入力と出力を構成する 407
ステップ 3: 自動生成された Amazon States Language 定義を確認してワークフローを保存する 409
ステップ 4: ステートマシンを実行する 411
分散マップ状態の使用開始 412

前提条件 412
ステップ 1: ワークフロープロトタイプを作成する 413
ステップ 2: マップステートの必須フィールドを設定する 413
ステップ 3: 追加オプションを設定する 415
ステップ 4: Lambda 関数を設定する 416
ステップ 5: ワークフロープロトタイプを更新する 417
ステップ 6: 自動生成された Amazon States Language 定義を確認してワークフローを保存する 417
ステップ 7: ステートマシンを実行する 419
Lambda 関数でデータバッチ全体を処理する 420
ステップ 1: ステートマシンを作成する 421
ステップ 2 : Lambda 関数を作成する 422
ステップ 3: ステートマシンを実行する 424
Lambda 関数で個々のデータ項目を処理する 425
ステップ 1: ステートマシンを作成する 426
ステップ 2 : Lambda 関数を作成する 428
ステップ 3: ステートマシンを実行する 424
Amazon S3 イベント発生時にステートマシンの実行をスタートする 433
前提条件: ステートマシンを作成する 433
ステップ 1: Amazon S3 バケットを作成する 434
ステップ 2: EventBridge で Amazon S3 イベント通知を有効にする 434
ステップ 3: Amazon EventBridge ルールを作成する 435
ステップ 4 : ルールをテストする 436
実行入力の例 437
API Gateway を使用した Step Functions API の作成 437
ステップ 1: API Gateway 用にIAM ロールを作成する 438
ステップ 2: API Gateway で API を作成する 439
ステップ 3: API Gateway API のテストとデプロイ 442
AWS SAM を使用して Step Functions ステートマシンを作成する 445
前提条件 445
ステップ 1: サンプル AWS SAM アプリケーションをダウンロードする 446
ステップ 2: アプリケーションを構築する 447
ステップ 3: アプリケーションを AWS クラウドにデプロイする 448
トラブルシューティング 449
クリーンアップ 450
アクティビティステートマシンを作成する 450

ステップ 1: アクティビティを作成する 451
ステップ 2: ステートマシンを作成する 452
ステップ 3: ワーカーを実装する 454
ステップ 4: ステートマシンを実行する 456
ステップ 5: ワーカーを実行して停止する 457
Lambda を使用してループを反復処理する 458
ステップ 1: Lambda 関数を作成してカウントを反復する 458
ステップ 2: Lambda 関数をテストする 459
ステップ 3: ステートマシンを作成する 461
ステップ 4: 新しい実行をスタートする 463
進行中の作業を新しい実行として継続する 464
Step Functions API アクションを使用する (推奨) 465
Lambda 関数を使用する 469
人間による承諾プロジェクト例をデプロイする 481
ステップ 1: テンプレートの作成 482
ステップ 2: スタックを作成する 482
ステップ 3: SNS サブスクリプションを承認する 483
ステップ 4: ステートマシンを実行する 484
テンプレートソースコード 486
Step Functions で X-Ray によるトレースを表示する 496
ステップ 1: Lambda 用に IAM ロールを作成する 497
ステップ 2: Lambda 関数を作成する 497
ステップ 3: さらに 2 つの Lambda 関数を作成する 499
ステップ 4: ステートマシンを作成する 499
ステップ 5: ステートマシンを実行する 502
AWS SDK サービスインテグレーションを使用して Amazon S3 バケット情報を収集する 505
ステップ 1: ステートマシンを作成する 505
ステップ 2: 必要な IAM ロールを追加する 508
ステップ 3: Standard ステートステートマシンを実行する 508
ステップ 4: Express ステートマシンを実行する 509
デベロッパーツール 511
開発オプション 511
Step Functions コンソール 512
AWS SDK 512
標準ワークフローと Express ワークフロー 513
HTTPS サービス API 513

[Development environments] (開発環境) 513
エンドポイント 514
AWS CLI 514
Step Functions Local 514
AWS Toolkit for Visual Studio Code 515
AWS Serverless Application Model とStep Functions 515
Terraform と Step Functions 515
定義形式のサポート 516
Step Functions と AWS SAM 523
AWS SAMでStep Functions を使用する理由 523
Step Functions と AWS SAM 仕様 との統合 524
SAM CLI との Step Functions の統合 524
DefinitionSubstitutions テンプレートで AWS SAM 525
次のステップ 529
Application Composer の Workflow Studio を使用する 530
Application Composer の Workflow Studio を使用する 530
CloudFormation の定義の置換を使用してリソースを動的に参照する 531
サービス統合タスクを拡張コンポーネントカードに接続する 531
既存のプロジェクトをインポートしてローカルに同期する 532
AWS Application Composer の Workflow Studio で使用できない機能 532
を使用して Lambda ステートマシンを作成する AWS CloudFormation 533
ステップ 1: AWS CloudFormation テンプレートを設定する 534
ステップ 2: AWS CloudFormation テンプレートを使用して Lambda ステートマシンを作成する 539
ステップ 3: ステートマシンの実行をスタートする 544
AWS CDK を使用して Lambda ステートマシンを作成する 545
ステップ 1:AWS CDK プロジェクトを設定する 546
ステップ 2: AWS CDK を使用してステートマシンを作成する 548
ステップ 3: ステートマシンの実行を開始する 556
ステップ 4: クリーンアップする 557
次のステップ 557
を使用した同期 Express ステートマシンを使用した API Gateway REST API の作成 AWS CDK 558
ステップ 1: AWS CDK プロジェクトを設定する 559
ステップ 2: AWS CDK を使用して、同期 Express ステートマシンバックエンド統合で API Gateway REST API を作成する 563

ステップ 3: API Gateway をテストする 573
ステップ 4: クリーンアップする 575
データサイエンス SDK 575
Terraform を使用する、ステートマシンのデプロイ 576
前提条件 577
Terraform を使用した開発ライフサイクル 577
ステートマシンの IAM ロールとポリシー 579
テストとデバッグ 581
TestState API の使用 581
TestState API の使用に関する考慮事項 582
TestState API での検査レベルの使用 583
IAM TestState API を使用するための アクセス許可 590
ステートのテスト (コンソール) 591
AWS CLI を使用してステートをテストする 592
入力データと出力データフローのテストとデバッグ 598
ステートマシンのローカルテスト 602
Step Functions Local (ダウンロード可能バージョン) と Docker のセットアップ 603
Step Functions Local (ダウンロード可能バージョン) - Java バージョンのセットアップ 604
Step Functions Local の設定オプションを指定する 605
Step Functions Local を自分のコンピュータで実行する 607
Step Functions と AWS SAM CLI Local のテスト 609
モックサービス統合の使用 614
ベストプラクティス 632
タイムアウトを使用して実行のスタックを回避する 632
ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する 633
履歴のクォータに到達しないようにする 635
Lambda サービスの例外を処理する 636
アクティビティタスクのポーリング時のレイテンシーを回避する 637
標準ワークフローまたは Express ワークフローの選択 638
Amazon CloudWatch Logs リソースポリシーのサイズ制限 638
他の サービスでの使用 640
AWS 他のサービスに電話してください。 640
最適化された統合 641
AWS SDK インテグレーション 641
統合パターンのサポート 641
クロスアカウントアクセス 645

AWS SDK サービスインテグレーション 645
AWS SDK サービスインテグレーションを使用する 646
サポートされる サービス 647
サポートされているサービスでサポートされていない API アクション 689
非推奨の AWS SDK サービス統合 691
最適化された統合 691
Amazon API Gateway 695
Amazon Athena 703
AWS Batch 706
Amazon Bedrock 708
AWS CodeBuild 712
Amazon DynamoDB 717
Amazon ECS/Fargate 721
Amazon EMR 724
Amazon EMR on EKS 736
Amazon EKS 740
Amazon EMR Serverless 754
Amazon EventBridge 763
AWS Glue 765
AWS Glue DataBrew 766
AWS Lambda 767
Amazon SageMaker 771
Amazon SNS 782
Amazon SQS 785
AWS Step Functions 787
サードパーティーの API を呼び出す 791
HTTP タスク定義 791
HTTP タスクフィールド 792
HTTP タスクの認証 798
EventBridge 接続データと HTTP タスク定義データをマージする 799
リクエスト本文に URL エンコーディングを適用します。 802
HTTP タスクを実行するための IAM アクセス許可 804
HTTP タスク例 805
HTTP タスクのテスト 808
サポートされていない HTTP タスクレスポンス 810
サービス統合パターン 810

レスポンスのリクエスト 811
ジョブの実行 (.sync) 811
タスクトークンのコールバックまで待機する 813
サービス API にパラメータを渡す 819
静的 JSON をパラメータとして渡す 819
パスを使用して状態入力をパラメータとして渡す 820
コンテキストオブジェクトノードをパラメータとして渡す 820
インテグレーションの変更ログ 821
Step Functions サンプルプロジェクト 845
バッチジョブの管理 (AWS Batch、Amazon SNS) 846
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 846
ステップ 2: ステートマシンを実行する 849
ステートマシンのコード例 850
IAM の例 851
コンテナタスクの管理 (Amazon ECS、Amazon SNS) 852
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 852
ステップ 2: ステートマシンを実行する 854
ステートマシンのコード例 855
IAM の例 857
データレコードの転送 (Lambda、DynamoDB、Amazon SQS) 858
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 859
ステップ 2: ステートマシンを実行する 861
ステートマシンのコード例 862
IAM の例 864
Job ステータスの投票 (Lambda、) AWS Batch 865
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 866
ステップ 2: ステートマシンを実行する 868
ステートマシンのコード例 870
タスクタイマー (Lambda、Amazon SNS) 872
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 873
ステップ 2: ステートマシンを実行する 875
コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda) 877
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 878
ステップ 2: ステートマシンを実行する 880
Lambda コールバックの例 881
Amazon EMR ジョブを管理する 882

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 883
ステップ 2: ステートマシンを実行する 854
ステートマシンのコード例 855
IAM の例 857
EMR Serverless ジョブを実行する 891
AWS CloudFormation テンプレートと追加リソース 891
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 892
ステップ 2: ステートマシンを実行する 894
ワークフロー内でワークフローをスタートする (Step Functions、Lambda) 895
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 895
ステップ 2: ステートマシンを実行する 898
ステートマシンのコード例 899
マップ状態を使用してデータを動的に処理する 901
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 901
ステップ 2: Amazon SNS トピックを登録する 905
ステップ 3: Amazon SQS キューにメッセージを追加する 905
ステップ 4: ステートマシンを実行する 906
ステートマシンのコード例 907
IAM の例 909
分散マップで CSV ファイルを処理する 910
AWS CloudFormation テンプレートと追加リソース 911
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 912
ステップ 2: ステートマシンを実行する 914
Amazon S3 バケットのデータを分散マップで処理する 915
AWS CloudFormation テンプレートと追加リソース 917
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 917
ステップ 2: ステートマシンを実行する 921
機械学習モデルのトレーニング 922
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 922
ステップ 2: ステートマシンを実行する 924
ステートマシンのコード例 925
IAM の例 928
機械学習モデルのチューニング 929
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 930
ステップ 2: ステートマシンを実行する 932
ステートマシンのコード例 933

IAM の例 ... 938
Amazon SQS からの大容量メッセージの処理 (Express ワークフロー) ... 940
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 941
ステップ 2: ステートマシンの実行をトリガーする ... 944
Lambda 関数のコードの例 ... 945
ステートマシンのコード例 ... 945
IAM の例 ... 946
選択的チェックポイントの例 (Express ワークフロー) ... 948
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 948
ステップ 2: ステートマシンを実行する ... 950
親のステートマシンのコード例 (標準ワークフロー) ... 951
親ステートマシンの IAM ロール の例 ... 954
ネストされたステートマシン (Express ワークフロー) のステートマシンコードの例 ... 951
子ステートマシンの IAM ロールの例 ... 958
AWS CodeBuild プロジェクトを構築する (CodeBuild、Amazon SNS) ... 959
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 959
ステップ 2: ステートマシンを実行する ... 961
ステートマシンのコード例 ... 962
データを前処理し、機械学習モデルをトレーニングする ... 964
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 965
ステップ 2: ステートマシンを実行する ... 967
ステートマシンのコード例 ... 968
IAM の例 ... 972
Lambda オーケストレーションの例 ... 973
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 973
ステップ 2: ステートマシンを実行する ... 976
ステートマシンとその実行について ... 977
IAM の例 ... 981
Athena クエリをスタートする ... 983
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 984
ステップ 2: ステートマシンを実行する ... 986
ステートマシンのコード例 ... 987
IAM の例 ... 989
複数のクエリを実行します (Amazon Athena、Amazon SNS) ... 991
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする ... 991
ステップ 2: ステートマシンを実行する ... 995

ステートマシンのコード例 995
IAM の例 998
大規模なデータセット (Amazon Athena、Amazon S3 AWS Glue、Amazon SNS) へのクエリ 1002
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1002
ステップ 2: ステートマシンを実行する 1005
ステートマシンのコード例 1006
IAM の例 1007
データを最新の状態に保つ (Amazon Athena、Amazon S3、 AWS Glue) 1011
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1011
ステップ 2: ステートマシンを実行する 1013
ステートマシンのコード例 1014
IAM の例 1015
Amazon EKS クラスターの管理 1017
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1018
ステップ 2: ステートマシンを実行する 1021
ステートマシンのコード例 1022
IAM の例 1026
API Gateway を呼び出す 1028
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1028
ステップ 2: ステートマシンを実行する 1030
ステートマシンのコード例 1031
IAM の例 1033
API Gateway でマイクロサービスを呼び出す 1033
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1034
ステップ 2: ステートマシンを実行する 1036
ステートマシンのコード例 1037
IAM の例 1039
カスタムイベントの送信先 EventBridge 1040
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1041
ステップ 2: ステートマシンを実行する 1043
ステートマシンのコード例 1044
IAM の例 1045
同期 Express ワークフローを呼び出す 1046
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1046
ステップ 2: ステートマシンを実行する 1048

ステートマシンのコード例 1049
IAM の例 1051
Amazon Redshift を使用した ETL/ELT ワークフローの実行 1052
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1053
ステップ 2: ステートマシンを実行する 1056
ステートマシンのコード例 1057
IAM の例 1078
エラー処理で Step Functions と AWS Batch を使用する 1079
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1079
ステップ 2: ステートマシンを実行する 1081
ステートマシンのコード例 1082
IAM の例 1084
AWS Batch ジョブのファンアウト 1085
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1085
ステップ 2: ステートマシンを実行する 1087
ステートマシンのコード例 1088
IAM の例 1090
AWS Batch Lambda 付き 1091
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1091
ステップ 2: ステートマシンを実行する 1093
ステートマシンのコード例 1094
IAM の例 1095
Amazon Bedrock で AI プロンプトチェーンを実行する 1097
AWS CloudFormation テンプレートと追加リソース 1097
前提条件 1098
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする 1098
ステップ 2: ステートマシンを実行する 1100
クォータ 1102
一般的なクォータ 1103
アカウントに関連するクォータ 1104
HTTP タスクに関連するクォータ 1105
状態のスロットリングに関連するクォータ 1105
API アクションのスロットリングに関連するクォータ 1106
API に関連するクォータ TestState 1107
その他のクォータ 1107
ステートマシンの実行に関連するクォータ 1111

タスクの実行に関連するクォータ 1113
バージョンとエイリアスに関連するクォータ 1114
タグ付けに関連する制限 1114
記録とモニタリング 1116
Amazon CloudWatch メトリクス 1116
時間間隔を報告するメトリクス 1117
カウントを報告するメトリクス 1118
実行メトリクス 1118
バージョンとエイリアスのリソース数メトリクス 1121
アクティビティのメトリクス 1122
Lambda 関数メトリクス 1123
サービス統合メトリクス 1124
サービスメトリクス 1125
API のメトリクス 1126
ベストエフォート型の CloudWatch メトリクス配信 1126
Step Functions メトリクスの表示 1127
Step Functions アラームの設定 1129
Amazon EventBridge イベント 1131
EventBridge ペイロード 1132
Step Functions イベントの例 1133
Step Functions イベントを にルーティングする EventBridge 1137
AWS CloudTrail を使用して Step Functions を口グ記録 1139
CloudTrail における Step Functions 情報 1139
例:Step Functions ログファイルエントリ 1140
CloudWatch Logs を使用したログ記録 1145
のログ記録の設定 1146
CloudWatch Logs ペイロード 1146
CloudWatch Logs にログ記録するための IAM ポリシー 1146
ログレベル 1148
X-Ray 1152
セットアップと設定 1153
概念 1157
サービス統合 1158
X-Ray コンソールの表示 1159
Step Functions の X-Ray トレース情報を表示 1160
トレース 1160

サービスマップ 1160
セグメントとサブセグメント 1162
分析 1163
構成 1164
トレースマップまたはサービスマップにデータがない場合はどうなりますか。 1165
Step Functions で AWS User Notifications を使用する 1166
セキュリティ 1167
データ保護 1167
暗号化 1168
アイデンティティとアクセスの管理 1169
対象者 1169
アイデンティティを使用した認証 1170
ポリシーを使用したアクセスの管理 1173
アクセスコントロール 1176
ポリシーアクション 1176
ポリシーリソース 1177
ポリシー条件キー 1178
ACL 1179
ABAC 1179
テンポラリ認証情報 1180
プリンシパル権限 1180
サービスロール 1181
サービスにリンクされたロール 1181
IAM AWS Step Functions との連携方法 1181
アイデンティティベースポリシーの例 1182
アイデンティティベースポリシー 1185
リソースベースのポリシー 1186
ステートマシンの IAM ロールの作成 1187
管理者以外のユーザー用の詳細な IAM 許可の作成 1189
AWS クロスアカウントリソースへのアクセス 1193
VPC エンドポイント 1204
統合サービスの IAM ポリシー 1207
分散マップ状態を使用するための IAM ポリシー 1296
タグベースのポリシー 1302
トラブルシューティング 1303
ログ記録とモニタリング 1305

コンプライアンス検証 1305
耐障害性 1306
インフラストラクチャセキュリティ 1306
設定と脆弱性の分析 1307
からのワークロードの移行 AWS Data Pipeline 1308
ワークロードの移行 1308
コンセプトマッピング 1309
Step Functions サンプルプロジェクト 1310
料金比較 1311
トラブルシューティング 1312
一般的なトラブルシューティング 1312
ステートマシンを作成できません。 1312
JsonPath を使用して前のタスクの出力を参照できません。 1312
状態遷移に遅延がありました。 1313
新しいStandard ワークフローの実行をスタートすると、ExecutionLimitExceeded エラーがあれば、失敗します。 1313
並列状態の 1 つのブランチで障害が発生すると、実行全体が失敗となります。 1313
サービス統合のトラブルシューティング 1313
ジョブはダウンストリームサービスで完了していますが、Step Functions ではタスクの状態が「進行中」のままになるか、完了が遅れます。 1313
ネストされたステートマシンの実行から JSON 出力を返したいと思っています。 1314
Lambda 関数を別のアカウントから呼び出すことはできません。 1314
.waitForTaskToken 状態から渡されたタスクークンが表示されません。 1315
アクティビティのトラブルシューティング。 1316
ステートマシンの実行がアクティビティ状態でスタックされます。 1316
タスクトークンを待っている間に、アクティビティワーカーがタイムアウトします。 1317
Express ワークフローのトラブルシューティング 1317
StartSyncExecution API コールからの応答を受信する前にアプリケーションをタイムアウトします。 1317
Express Workflow 障害をトラブルシューティングするために、実行履歴を表示できません。 1317
関連情報 1319
最近リリースされた機能 1320
ドキュメント履歴 1323
AWS 用語集 1361
..... mccclxii

とは何ですか AWS Step Functions?

AWS Step Functions は、AWS Lambda AWS のサービス 機能などと統合してビジネスクリティカルなアプリケーションを構築できるサーバーレスオーケストレーションサービスです。Step Functions のグラフィカルコンソールでは、アプリケーションのワークフローを一連のイベント駆動型ステップとして確認できます。

Step Functions はステートマシンとタスクに基づいています。Step Functions では、ワークフローをステートマシンと呼びます。ステートマシンは、一連のイベント駆動型ステップです。ワークフローの各ステップはステートと呼ばれます。タスクステートは、 AWS などの別のサービスが実行する作業単位を表します。 AWS Lambdaタスクステートは任意の API AWS のサービス または API を呼び出すことができます。

Step Functions の組み込みコントロールを使用して、ワークフロー内の各ステップの状態を調べて、アプリケーションが期待どおりに実行されていることを確認します。ユースケースによっては、Step Functions が Lambda AWS などのサービスを呼び出してタスクを実行できます。機械学習モデルを処理してパブリッシュするワークフローを作成できます。Step Functions AWS などのコントロールサービスを使用してAWS Glue、抽出、変換、読み込み (ETL) ワークフローを作成できます。また、手動による介入が必要なアプリケーション用に実行時間が長い自動化されたワークフローを作成することもできます。

> **Tip**
>
> 一連のインタラクティブモジュールを通じて Step Functions の主な機能に慣れるには、「ワークショップ」を参照してください。 AWS Step Functionsまたは、以下の「入門チュートリアル」に従って Step Functions の使用を開始し、クレジットカード申請ワークフローを作成してください。

トピック

- AWS SDK と最適化されたインテグレーション
- Standard ワークフローと Express ワークフロー
- ユースケース
- サービス統合
- サポートされるリージョン

- Step Functions を使うのは今回が初めてですか?

AWS SDK と最適化されたインテグレーション

AWS 他のサービスを呼び出すには、Step Functions の AWS SDK インテグレーションを使用するか、Step Functions の最適化インテグレーションのいずれかを使用できます。

- AWS SDK インテグレーションでは、200 AWS を超えるサービスのいずれかをステートマシンから直接呼び出すことができるため、9,000 を超える API アクションにアクセスできます。
- Step Functions の最適化された統合は、カスタマイズされており、ステートマシンで使いやすくなっています。

Standard ワークフローと Express ワークフロー

Step Functions には 2 つのワークフロータイプがあります。Standard ワークフローでは 1 度だけワークフローが実行され、最大 1 年間実行できます。つまり、標準ワークフローの各ステップは 1 回だけ実行されます。ただし、Express at-least-once ワークフローにはワークフロー実行機能があり、最大 5 分間実行できます。つまり、Express ワークフローの 1 つ以上のステップが複数回実行される可能性があり、ワークフロー内の各ステップは少なくとも 1 回実行されます。

実行は、ワークフローを実行してタスクを実施するインスタンスです。Standard ワークフローは、実行履歴と視覚的なデバッグを示すため、実行時間が長い監査可能なワークフローに最適です。Express ワークフローは、ストリーミングデータ処理や IoT high-event-rate データの取り込みなどのワークロードに最適です。

標準ワークフロー仕様

- 毎秒 2,000 の実行レート
- 毎秒 4,000 のステート移行レート
- ステート移行別の価格設定
- 実行履歴と視覚的なデバッグの表示
- すべてのサービス統合とパターンのサポート

Express ワークフローの仕様

- 毎秒 100,000 の実行レート

- ほぼ無制限のステート移行レート
- 実行回数および実行期間別の価格設定
- 実行履歴をAmazon に送信 CloudWatch
- 有効になっている[ログレベル]に基づいて、実行履歴と視覚的なデバッグを表示します。
- すべてのサービス統合とほとんどのパターンをサポート

Step Functions 料金など、Standard ワークフローと Express ワークフローの詳細については、次を参照してください。

- 標準ワークフロー対 Express ワークフロー
- AWS Step Functions 料金表

ユースケース

Step Functions はアプリケーションのコンポーネントとロジックを管理するため、コードの書き込みを減らし、アプリケーションの迅速な構築と更新に集中することができます。このセクションでは、Step Functions を使用する一般的なユースケースについて説明します。

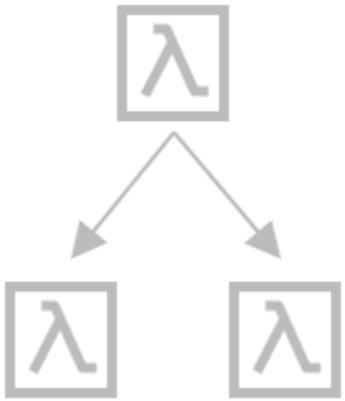
ユースケース #1: 関数オーケストレーション

特定の順序で Lambda 関数 (ステップ) のグループを実行するワークフローを作成します。1 つの Lambda 関数の出力が次の Lambda 関数の入力に渡されます。ワークフローの最後のステップで結果が得られます。Step Functions を使用すると、ワークフローの各ステップがどのように相互作用するかを確認できるため、各ステップが意図通りの機能を確実に実行できるようになります。

関数のグループを含むステートマシンを作成する方法を示すチュートリアルについては、以下を参照してください。

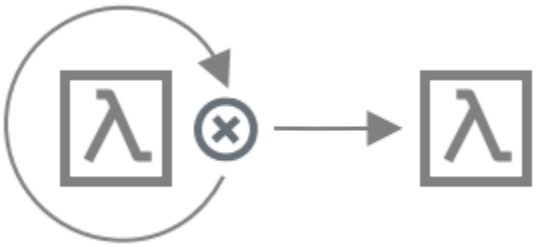
- AWS Step Functions の開始方法

ユースケース #2: 分岐



顧客が与信限度額の増加を要求します。`Choice` 状態を使って、Choice 状態の入力に基づいた判断を Step Functions にさせることができます。リクエストが事前承認した顧客のクレジット上限を超える場合は、Step Functions から顧客のリクエストをマネージャーに送信してサインオフを求めることができます。リクエストが事前承認した顧客のクレジット上限を下回っている場合は、Step Functions でリクエストを自動的に承認することができます。

ユースケース #3: エラー処理



`Retry`

このユースケースでは、顧客はユーザーネームをリクエストしています。初めて、顧客のリクエストは正常に行われません。`Retry` ステートメントを使って、Step Functions で顧客のリクエストを再試行することができます。2 回目に、顧客のリクエストは正常に終了します。

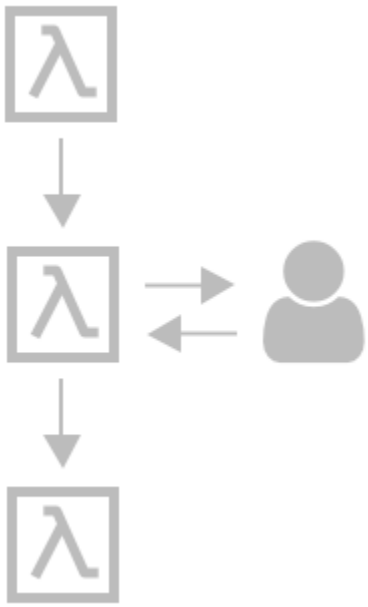
`Catch`

同じようなユースケースでは、顧客は利用できないユーザーネームをリクエストしてします。`Catch` ステートメントを使って、Step Functions より利用可能なユーザーネームを提案します。顧客が利用可能なユーザーネームを使用する場合、Step Functions でワークフローの次のステップに進み、確認メールを送信します。顧客が利用可能なユーザーネームを使用しない場合、Step Functions はワークフローの別のステップに進み、サインアッププロセスを最初からやり直します。

`Retry` そして `Catch` ステートメントの詳細な例については、以下を参照してください。

- Step Functions のエラー処理

ユースケース #4: ヒューマンインザループ

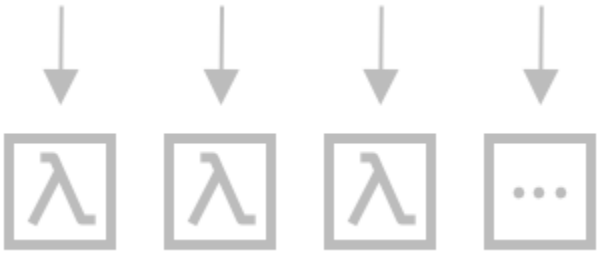


バンキングアプリケーションを使用して、顧客の一人が友達に送金します。カスタマーは確認メールを待ちます。コールバックとタスクトークンを使って、Step Functions より、顧客の金を送金して、顧客の友人が送金を受領した時に報告するように Lambda に指示します。Lambda から、顧客の友だちが送金を受領したと報告を受けた後、Step Functions をワークフローの次のステップに進み、顧客に確認メールを送信します。

タスクトークンを使ってコールバックを表示するサンプルプロジェクトを表示するには、以下を参照してください。

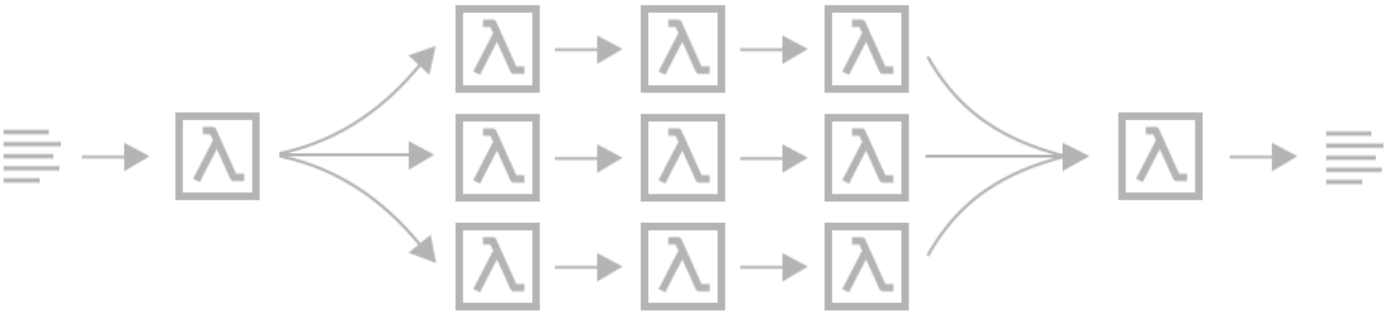
- コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda)

ユースケース #5: 並列処理



顧客がビデオファイルを 5 つの異なる表示解像度に変換するため、ビューワーは複数のデバイスで動画を視聴できます。`Parallel` 状態を使用して、Step Functions はビデオファイルを入力するので、Lambda は同時に 5 つの表示解像度に処理できます。

ユースケース #6: 動的並列処理



顧客から 3 つの商品の注文を受けると、各商品の発送準備が必要となります。各商品の在庫状況をチェック、各商品を収集、各商品を梱包した後で配送します。`Map` 状態を使用して、Step Functions は、Lambda に顧客の各商品を並列で処理させます。顧客が選んだ全商品を配送用に梱包すると、Step Functions はワークフローの次のステップに進み、追跡情報を記載した確認メールを顧客に送信します。

`Map` を使用した動的並列処理を示すサンプルプロジェクトを表示するには、以下を参照してください。

- マップ状態を使用してデータを動的に処理する

サービス統合

Step Functions AWS は複数のサービスと統合されています。Step Functions とこれらのサービスを組み合わせるには、次のサービス統合パターンを使用します。

レスポンスをリクエスト (デフォルト)

- サービスを呼び出し、Step Functions が HTTP レスポンスを取得した直後に次のステップに進むことができるようにします。

ジョブの実行 (.sync)

- サービスを呼び出し、ジョブが完了するまで Step Functions が待機するようにします。

タスクトークン (. waitForTaskトークン)

- タスクトークンでサービスを呼び出し、タスクトークンがコールバックとともに返されるまで Step Functions を待機させます。

次の表に、Step Functions で使用可能なサービス統合とサービス統合パターンを示します。

標準ワークフローとエクスプレスワークフローは同じ統合をサポートしますが、同じ統合パターンはサポートしません。

- 最適化された統合パターンのサポートは、各統合ごとに異なります。
- エクスプレスワークフローは、 Job 実行 (.sync) またはコールバックを待機 (. waitForTaskトークン)。
- 詳細については、「標準ワークフロー対 Express ワークフロー」を参照してください。

Standard Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
AWS SDK インテグレーション	200 以上	✓		✓

Express Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	Amazon SageMaker	✓		
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS SDK インテグレーション	200 以上	✓		

サポートされるリージョン

AWS ほとんどの地域がStep Functions をサポートしています。Step Functions AWS を使用できるリージョンの全リストについては、AWS リージョンテーブルを参照してください。

Step Functions を使うのは今回が初めてですか?

Step Functions を初めて使用する場合は、Step Functions AWS を他のサービスとどのように組み合わせるかなど、Step Functions を使用する際のさまざまな部分を理解するのに役立つトピックを以下に示します。

- Step Functions チュートリアル
- Step Functions サンプルプロジェクト
- AWS Step Functions Python 用データサイエンス SDK

AWS Step Functions のスタート方法の前提条件

初めて AWS Step Functions を使ってスタートする前に、このページに記載されている前提条件をすべて実行してください。

トピック

- ステップ 1: AWS アカウント と IAM ユーザーにサインアップする
- ステップ 2: プログラム的なアクセス権を付与する

ステップ 1: AWS アカウント と IAM ユーザーにサインアップする

任意の AWS のサービスにアクセスするには、まず AWS アカウント を作成する必要があります。AWS アカウント は、アクティビティおよび使用状況レポートの表示や、認証およびアクセスの管理に使用できます。使用する製品とサービスだけが課金され、無料で AWS の使用をスタートできます。詳細については、AWS 無料利用枠を参照してください。

Step Functions アクションに AWS アカウントのルートユーザー を使用しないようにするには、Step Functions への管理アクセスが必要なユーザーごとに IAM ユーザーを作成することがベストプラクティスです。

AWS アカウントを既にお持ちの場合は、次のステップに進んでください。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話のキーパッドを使用して検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービスとリソースへのアクセス権があります。セキュリティのベストプラクティスとして、管理ユーザーに管理アクセスを割り当

て、ルートユーザーアクセスが必要なタスクを実行する場合にのみ、ルートユーザーを使用してください。

サインアップ処理が完了すると、AWS からユーザーに確認メールが送信されます。https://aws.amazon.com/ の [アカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理ユーザーの作成

AWS アカウント にサインアップしたら、AWS アカウントのルートユーザー をセキュリティで保護し、AWS IAM Identity Center を有効にして管理ユーザーを作成します。日常的なタスクには、管理ユーザーを使用し、ルートユーザーを使用しないようにします。

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウント のメールアドレスを入力して、アカウント所有者として AWS Management Console にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、「AWS サインイン User Guide」の「Signing in as the root user」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の「AWS アカウントのルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理ユーザーの作成

1. IAM Identity Center を有効にする

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。
2. IAM アイデンティティセンターで、管理ユーザーに管理アクセス権を付与します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリ を使用したユーザーアクセスの設定」を参照してください。

管理ユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM アイデンティティセンターのユーザーを使用してサインインする方法については、「AWS サインイン User Guide」の「Signing in to the AWS access portal」を参照してください。

ステップ 2: プログラム的なアクセス権を付与する

AWS Management Console の外部で AWS を操作するには、プログラマチックアクセス権が必要です。プログラマチックアクセス権を付与する方法は、AWS にアクセスしているユーザーのタイプによって異なります。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	To	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラマチックリクエストに署名します。	使用するインターフェイス用の手引きに従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「AWS IAM Identity Center を使用するための AWS CLI の設定」を参照してください。 • 既存の AWS KMS 小売り店キーを選択します。Category: spacingAWS SDK、ツール、および AWS API については、AWS SDK とツールリファレンスガイドの「IAM Identity Center

プログラマチックアクセス権を必要とするユーザー	To	方法
		認証」を参照してください。
IAM	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラムによるリクエストに署名します。	「IAM ユーザーガイド」の「AWS リソースでの一時的な認証情報の使用」の指示に従ってください。
IAM	(非推奨) 長期的な認証情報を使用して、AWS CLI、AWS SDK、AWS API へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手順に従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDK とツールについては、AWS SDK とツールリファレンスガイドの「長期認証情報を使用して認証する」を参照してください。 • AWS API については、IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」を参照してください。

AWS Step Functions の開始方法

Step Functions 入門チュートリアルシリーズへようこそ。

Step Functions は、一連のイベント駆動型ステップとしてアプリケーションワークフローを定義できるサーバーレスオーケストレーションサービスです。ワークフローの各ステップは状態と呼ばれます。ワークフローの定義には タスク、選択、並行、マッピング などの状態を使用するのが最も一般的です。Task 状態内では、Step Functions がサポートする AWS SDK 統合を使用でき、ワークフロー内の複数の AWS のサービス をオーケストレーションできます。

トピック

- 主要なコンセプト
- このシリーズのチュートリアル
- チュートリアル 1: ステートマシンのプロトタイプを作成する
- チュートリアル 2: Lambda 関数を使用して最初のサービス統合を定義する
- チュートリアル 3: ワークフローに if-else 条件を実装する
- チュートリアル 4: 並列で実行する複数のタスクを定義する
- チュートリアル 5: 項目のコレクションを同時に反復処理する
- チュートリアル 6: ワークフローを保存してステートマシンを実行する
- チュートリアル 7: 入力と出力を構成する
- チュートリアル 8: コンソールでエラーをデバッグする

主要なコンセプト

このセクションでは、Step Functions の重要な概念を紹介します。開始する前に、次の主要な概念を確認してください。

期間	説明
ワークフロー	一連のステップを説明して、多くのビジネスプロセスに適合させます。
Workflow Studio	ワークフローのプロトタイプ作成と構築を迅速に行えるようにする視覚的なワークフローデザイナー。詳細については「AWS Step Functions ワークフロースタジオ」を参照してください。

期間	説明
状態	ステートマシン内の個々のステップ。ステートマシンでさまざまな機能を実行します。詳細については「状態」を参照してください。
ステートマシン	StartAt、TimeoutSeconds 、Version などのフィールドを使用してワークフロー内の個々の状態またはステップを表す JSON テキストを使って定義されるワークフロー。詳細については「ステートマシン構造」を参照してください。
Amazon ステートメント言語	ステートマシンを定義するための、JSON ベースの構造化言語。これは、作業の実行 (Task 状態)、次に移行する状態の決定 (Choice 状態)、エラーで実行の停止 (Fail 状態) などを実行できる、状態のコレクションです。詳細については「Amazon ステートメント言語」を参照してください。
入力および出力の設定	ワークフローのそれぞれの状態で JSON データを入力として受け取ります。また、通常 JSON データを出力として次の状態に渡します。Step Functions には、状態間の入出力データフローを制御するための複数のフィルターが用意されています。詳細については「Step Functions の入出力処理」を参照してください。
サービス統合	Step Functions は AWS のサービス と統合し、ワークフローから各サービスの API アクションを直接呼び出すことができます。詳細については「AWS Step Functions 他のサービスとの併用」を参照してください。
サービス統合タイプ	Step Functions には以下のサービス統合タイプがあります。 • 最適化された統合 - ワークフローに特別な機能を提供するために Step Functions によってカスタマイズされています。例えば、Lambda 呼び出しは API 出力を、エスケープされた JSON 文字列から JSON オブジェクトに変換します。 • AWS SDK 統合 - AWS SDK を使った標準 API 呼び出しとまったく同じように動作します。200 を超える AWS のサービス のサービスをステートマシンから直接呼び出して、9,000 を超える API アクションにアクセスできます。 詳細については「AWS Step Functions 他のサービスとの併用」を参照してください。

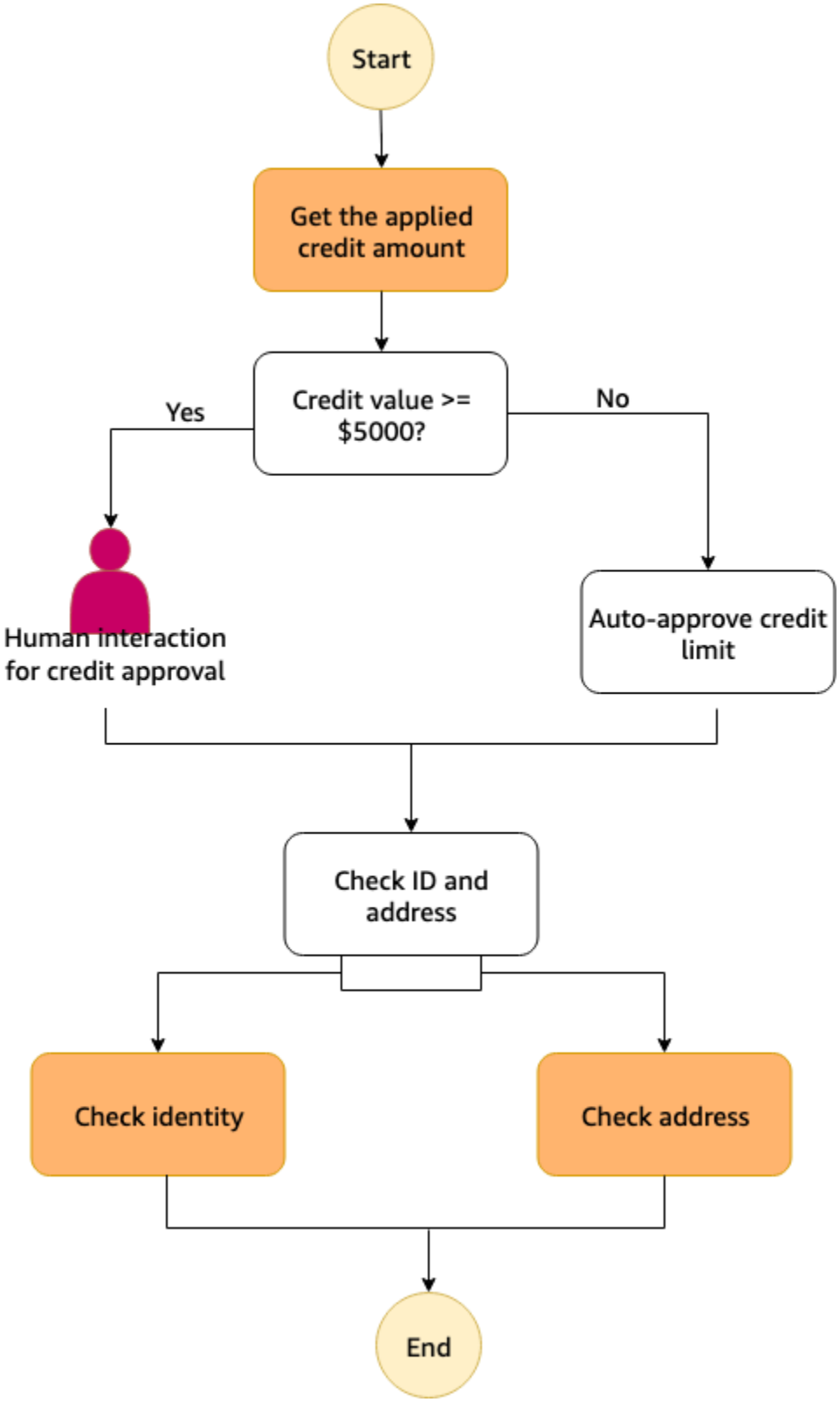
期間	説明
サービス統合パターン	ワークフロー内で、統合された AWS のサービス を呼び出すには、Step Functions が提供する以下のサービス統合パターンのいずれかを使用します。 • レスポンスをリクエスト (デフォルト) - サービスを呼び出し、Step Functions が HTTP レスポンスを取得した直後に次のステップに進むことができるようにします。 • ジョブの実行 (.sync) - サービスを呼び出し、ジョブが完了するまで Step Functions が待機するようにします。 • タスクトークンのコールバックまで待機する (.waitForTaskToken) - タスクトークンを使用してサービスを呼び出し、タスクトークンがコールバックとともに返されるまで Step Functions を待機させます。
実行	ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。詳細については「Step Functions で実行」を参照してください。

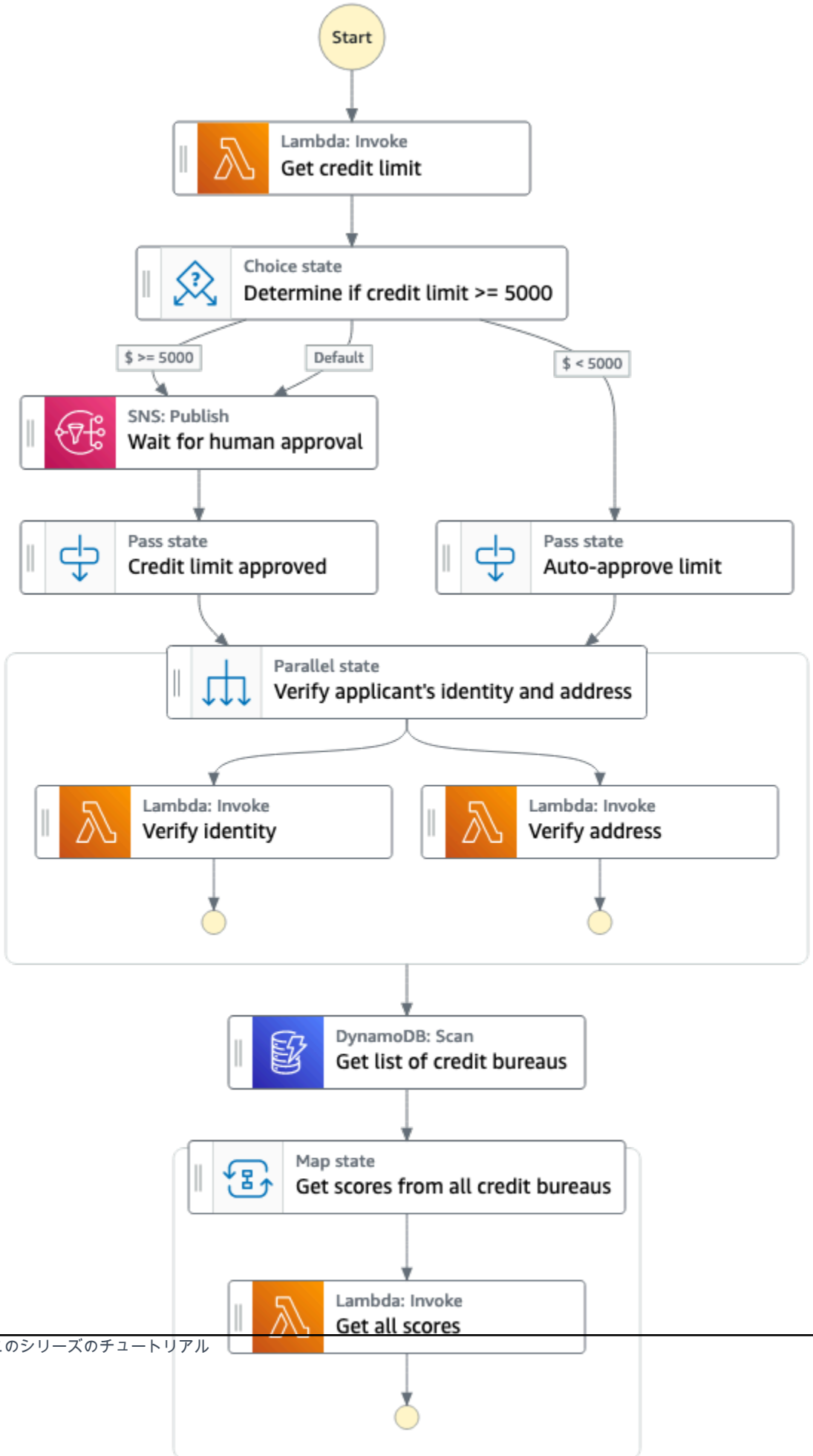
このシリーズのチュートリアル

この章の「はじめに」チュートリアルでは、クレジットカード申請を処理するための基本的なワークフローの作成方法を順を追って説明します。これらのチュートリアルでは、Step Functions でよく使用されるステートを使用する方法を学習します。作成したワークフローを、AWS Lambda や Amazon Simple Notification Service など、他の AWS のサービス と統合します。これらのチュートリアルを完了すると、クレジットカード申請の処理をシミュレートする簡単なワークフローが完成します。

> **Note**
>
> これらの「はじめに」チュートリアルではクレジットカード申請のワークフローについて説明していますが、Step Functions を使用すれば複数のタイプのワークフローを作成できます。例えば、データ処理、IT オートメーション、機械学習、メディア処理、注文処理などのワークフローを作成できます。

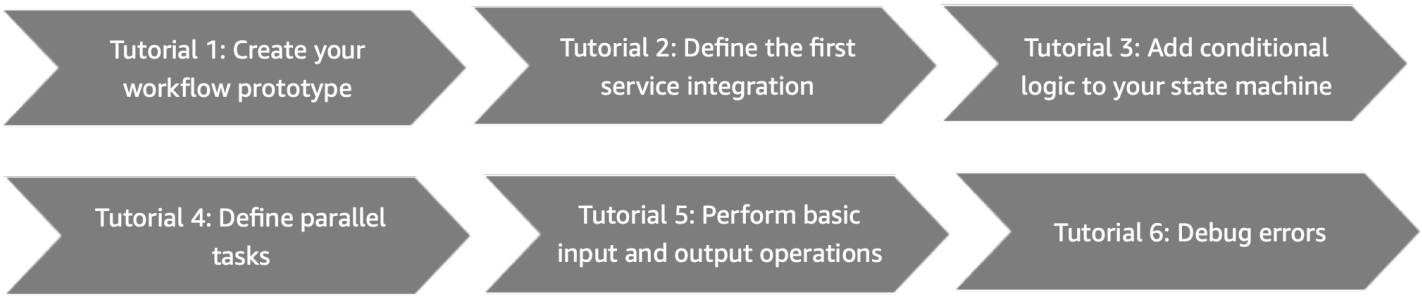
以下のイメージは、クレジットカード申請のワークフローと、Step Functions を使用してオーケストレーションを行った場合にどのように表示されるかを示しています。フローチャートの各ステップは、Step Functions ワークフローの状態と共に表されます。


Start
Get the applied credit amount
Credit value >= $5000?
Yes
No
Human interaction for credit approval
Auto-approve credit limit
Check ID and address
Check identity
Check address
End


Start
Lambda: Invoke
Get credit limit
Choice state
Determine if credit limit >= 5000
$ >= 5000
Default
$ < 5000
SNS: Publish
Wait for human approval
Pass state
Credit limit approved
Pass state
Auto-approve limit
Parallel state
Verify applicant's identity and address
Lambda: Invoke
Verify identity
Lambda: Invoke
Verify address
DynamoDB: Scan
Get list of credit bureaus
Map state
Get scores from all credit bureaus
Lambda: Invoke
Get all scores

> **Note**
>
> これらのチュートリアルをすべて順番に完了することをお勧めします。チュートリアルをすべて完了すると、Step Functions を本稼働ワークフローで使用するための重要な概念と機能の使い方を学ぶことができます。

次のロードマップは、Step Functions の Workflow Studio を使用してクレジットカード処理ワークフローを構築するために実行するステップを示しています。これらのステップは、そのステップを完了する方法についての説明を含む一連のチュートリアルとして提供されています。



開始するには、前提条件を満たしていることを確認する必要があります。

チュートリアル 1: ステートマシンのプロトタイプを作成する

このチュートリアルでは、Step Functions の Workflow Studio を使用して、クレジットカード処理ワークフロープロトタイプを作成します。必要な API アクションと状態を、それぞれ [アクション] タブと [フロー] タブから選択し、Workflow Studio のドラッグアンドドロップ機能を使用してワークフロープロトタイプを作成します。これ以降のチュートリアルでは、このワークフローで使用する AWS のサービス と、Step Functions の状態の設定方法を学習します。

ステートマシンのプロトタイプを作成するには

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選択します。これにより、デザインモード で Workflow Studio が開きます。
4. Workflow Studio で、[アクション] タブから [AWS Lambda 呼び出し] API アクションをドラッグし、[最初の状態をここにドラッグ] とラベル付けされた空の状態にドロップします。これを次のように設定します。

    - [設定] タブの [状態名] に **`Get credit limit`** と入力します。

5. [フロー] タブから [選択] 状態をドラッグアンドドロップして、[クレジット限度額を取得] 状態の下にドロップします。[選択] 状態の名前を **Credit applied >= 5000?** に変更します。
6. 次の状態を、[適用したクレジット >= 5000?] 状態の分岐をドラッグアンドドロップします。
   a. [Amazon SNS 発行] - [アクション] タブから、[Amazon SNS 発行] API アクションをドラッグアンドドロップします。この状態の名前を **Wait for human approval** に変更します。
   b. [パス]状態 - [フロー] タブから [パス] 状態をドラッグアンドドロップします この分岐の名前を **Auto-approve limit** に変更します。
7. [パス] 状態を [人による承認待ち] 状態の下にドラッグアンドドロップします。この [パス] 状態の名前を **Credit limit approved** に変更します。
8. 次のように [選択] 状態の後に [並列] 状態をドラッグアンドドロップします。
   a. [クレジット限度額承認済み] 状態の後に [並列] 状態をドロップします。
   b. [並列] 状態の名前を **Verify applicant's identity and address** に変更します。
   c. [並行] 状態の両方の分岐の下に、2 つの [AWS Lambda 呼び出し] API アクションをドラッグアンドドロップします。
   d. これらの状態の名前をそれぞれ、**Verify identity** と **Verify address** に変更します。
   e. [自動承認上限] 状態を選択し、[次の状態] で [申請者の ID と住所の確認] を選択します。
9. [DynamoDB Scan] 状態をドラッグし、[申請者の ID と住所の確認] 状態の下にドロップします。[DynamoDB Scan] の名前を **Get list of credit bureaus** に変更します。
10. [マップ] 状態を [信用調査機関のリストを取得] 状態の後にドラッグアンドドロップします。次のように [マップ] 状態を設定します。
    a. この名前を **Get scores from all credit bureaus** に変更します。
    b. 処理モード は、デフォルトの [インライン] を選択したままにします。
    c. [AWS Lambda 呼び出し] API アクションを [ここに状態をドロップ] とラベル付けされた空の状態にドラッグします。
    d. AWS Lambda 呼び出し 状態の名前を **Get all scores** に変更します。
11. 以降のアクションに進むには、このウィンドウを開いたままで次のチュートリアルに進みます。

次のステップ

次のチュートリアルでは、[クレジット限度額の取得] 状態で使用される Lambda 関数を統合する方法を学習します。

チュートリアル 2: Lambda 関数を使用して最初のサービス統合を定義する

このチュートリアルでは、ワークフローの最初のサービス統合を定義する方法を学習します。Lambda 関数を呼び出すには、[クレジット限度額を取得] という名前の `Task` 状態を使用します。`Task` 状態内では、Step Functions がサポートする AWS SDK 統合を使用できます。

ワークフローの最初のサービス統合を定義するには、最初に Lambda 関数を作成します。次に、ワークフローを更新して Lambda 関数とのサービス統合を指定します。このチュートリアルで使用する Lambda 関数は、申請者が申請したクレジット限度額を表す、ランダムに生成された整数を返します。

トピック

- ステップ 1: Lambda 関数を作成およびテストする
- ステップ 2: ワークフローの更新 - [クレジット限度額を取得] 状態を設定する
- 次のステップ

ステップ 1: Lambda 関数を作成およびテストする

関数のコードは、AWS Management Console または使い慣れたエディタで作成できます。次のステップでは、`RandomNumberforCredit` というタイトルの Node.js Lambda 関数を作成します。

> **Important**
>
> チュートリアル 1 で作成したワークフロープロトタイプが、このチュートリアルで作成する Lambda 関数と同じ AWS リージョン であることを確認してください。

1. 新しいタブまたはウィンドウで Lambda コンソールを開き、**`RandomNumberforCredit`** というタイトルの Node.js 16.x Lambda 関数を作成します。コンソールを使用した Lambda 関数の作成については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda の関数の作成」を参照してください。

2. [RandomNumberforCredit] ページで、[index.mjs] を選択し、[コードソース] 領域の既存のコードを次のコードに置き換えます。

```
export const handler = async function(event, context) {

    const credLimit = Math.floor(Math.random() * 10000);
    return (credLimit);

};
```

3. [関数の概要]セクションから、Lambda 関数の Amazon リソースネームをコピーし、そのテキストファイルに保存します。[クレジット限度額を取得] 状態のサービス統合を指定する場合は、関数 ARN が必要になります。ARN の例を次に示します。

```
arn:aws:lambda:us-east-2:123456789012:function:HelloWorld
```

4. [デプロイ] を選択して、次に [テスト] を選択して変更をデプロイし、Lambda 関数の出力を確認します。

ステップ 2: ワークフローの更新 - [クレジット限度額を取得] 状態を設定する

Step Functions コンソールで、ワークフローを更新して、ステップ 1 で作成した RandomNumberforCredit Lambda 関数 とのサービス統合を指定します。

1. チュートリアル 1 で作成したワークフロープロトタイプを含む、Step Functions コンソールウィンドウを開きます。
2. [クレジット限度額を取得] 状態を選択し、[設定] タブで次の操作を行います。
   a. [統合タイプ] は、デフォルトの [最適化] のままにします。

      Step Functions を使用すると、他の AWS のサービス と統合して、ワークフロー内でそれらを調整できます。サービス統合とタイプの詳細については、「AWS Step Functions 他のサービスとの併用」を参照してください。
   b. [関数名] には、ドロップダウンリストから [RandomNumberforCredit] Lambda 関数を選択します。
   c. 残りの項目は、デフォルトの選択のままにします。
3. 以降のアクションに進むには、このウィンドウを開いたままで次のチュートリアルに進みます。

> **Note**
>
> このチュートリアルでは、ワークフローの Task 状態内で Lambda 関数と統合する方法を学習しました。次の構文に示すように、サービス名と API コールを指定することにより、サポートされている他の AWS SDK 統合を Task 状態内で使用することもできます。
>
> ```
> arn:aws:states:::aws-sdk:serviceName:apiAction
> ```
>
> 詳細については「AWS Step Functions 他のサービスとの併用」を参照してください。

次のステップ

次のチュートリアルでは、ワークフローに条件付きロジックを実装します。Step Functions ステートマシンの条件付きロジックは、ごく一般的なプログラミング言語の if-else ステートメントと同様に動作します。ワークフローに条件ロジックを使用して、条件の情報に基づいて実行パスを決定します。

チュートリアル 3: ワークフローに if-else 条件を実装する

状態を使用することにより、ワークフローに if-else 条件を実装できます。指定された条件が「true」または「false」のどちらと評価されるかによって、ワークフローの実行パスを決定します。

このチュートリアルでは、チュートリアル 2 で使用した `RandomNumberforCredit` Lambda 関数によって返されたクレジットの申請額が、特定のしきい値を超えたかどうかを判断する条件付きロジックを追加します。金額が上限を超えた場合、アプリケーションでは、承認には人による操作が必要になります。それ以外の場合、アプリケーションによって自動承認され、次のステップに進みます。

タスクトークンが返されるまでワークフローの実行を一時停止することにより、人による操作ステップを模倣します。これを行うには、このチュートリアルで使用する AWS SDK 統合である Amazon Simple Notification Service にタスクトークンを渡します。ワークフローの実行は、`SendTaskSuccess` API コールでタスクトークンが返されるまで一時停止されます。タスクトークン使用の詳細については、「タスクトークンのコールバックまで待機する」を参照してください。

このチュートリアルでは、ワークフロープロトタイプで人による承認と自動承認のステップを定義済みなので、まず、コールバックトークンを受け取る Amazon SNS トピックを作成します。次に、

コールバック機能を実装する Lambda 関数を作成します。最後に、これらの AWS のサービス 統合の詳細を追加して、ワークフロープロトタイプを更新します。

トピック

- ステップ 1: コールバックトークンを受け取る Amazon SNS トピックを作成する
- ステップ 2: コールバックを処理する Lambda 関数を作成する
- ステップ 3: ワークフローの更新 - 選択状態に if-else 条件ロジックを追加する
- 次のステップ

ステップ 1: コールバックトークンを受け取る Amazon SNS トピックを作成する

人による操作ステップを実装するには、Amazon Simple Notification Service トピックに公開し、このトピックにコールバックタスクトークンを渡します。タスクトークンがペイロードとともに返されるまで、コールバックタスクはワークフローの実行を一時停止します。

1. Amazon SNS コンソールを開き、[標準] トピックタイプを作成します。トピックの作成の詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS トピックを作成する」を参照してください。
2. トピック名に **`TaskTokenTopic`** を指定します。
3. 必ずトピック ARN をコピーし、テキストファイルに保存してください。人による承認待ち状態のサービス統合を指定する場合には、トピック ARN が必要になります。トピック ARN の例を次に示します。

   ```
   arn:aws:sns:us-east-2:123456789012:TaskTokenTopic
   ```

4. トピックの E メールベースのサブスクリプションを作成し、次にサブスクリプションを確定します。トピックへのサブスクリプションの詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「トピックに対するサブスクリプションを作成する」を参照してください。

ステップ 2: コールバックを処理する Lambda 関数を作成する

コールバック機能を処理するには、Lambda 関数を定義し、この関数のトリガーとして、ステップ 1 で作成した Amazon SNS トピックを追加します。タスクトークンを使用して Amazon SNS トピックに発行すると、発行されたメッセージのペイロードで Lambda 関数が呼び出されます。

- ステップ 2.1: コールバックを処理する Lambda 関数を作成する
- ステップ 2.2: Amazon SNS トピックを Lambda 関数のトリガーとして追加する
- ステップ 2.3: Lambda 関数の IAM ロールに必要なアクセス許可を付与する

ステップ 2.1: コールバックを処理する Lambda 関数を作成する

この関数は、クレジット限度額承認リクエストを処理し、SendTaskSuccess API コールによって、リクエストの結果を成功として返します。この Lambda 関数は、Amazon SNS トピックから受け取ったタスクトークンも返します。

簡略化のため、人による操作ステップに使用される Lambda 関数は、すべてのタスクを自動的に承認し、SendTaskSuccess API コールでタスクトークンを返します。Lambda 関数には、**`callback-human-approval`** という名前を付けることができます。

1. 新しいタブまたはウィンドウで Lambda コンソールを開き、**`callback-human-approval`** というタイトルの Node.js 16.x Lambda 関数を作成します。コンソールを使用した Lambda 関数の作成については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda の関数の作成」を参照してください。
2. [callback-human-approval] ページで、[コードソース] 領域の既存のコードを、次のコードに置き換えます。

```
// Sample Lambda function that will automatically approve any task whenever a
 message is published to an Amazon SNS topic by Step Functions.

console.log('Loading function');
const AWS = require('aws-sdk');
const resultMessage = "Successful";

exports.handler = async (event, context) => {
    const stepfunctions = new AWS.StepFunctions();

    let message = JSON.parse(event.Records[0].Sns.Message);
    let taskToken = message.TaskToken;

    console.log('Message received from SNS:', message);
    console.log('Task token: ', taskToken);

    // Return task token to Step Functions

    let params = {
```

```
        output: JSON.stringify(resultMessage),
        taskToken: taskToken
    };

    console.log('JSON Returned to Step Functions: ', params);
    let myResult = await stepfunctions.sendTaskSuccess(params).promise();
    console.log('State machine - callback completed..');

    return myResult;

};
```

3. 以降のアクションに進むには、このウィンドウを開いたまま次のステップに進みます。

ステップ 2.2: Amazon SNS トピックを Lambda 関数のトリガーとして追加する

このチュートリアルのステップ 1 で作成した Amazon SNS トピックを、このチュートリアルのステップ 2.1 で作成した Lambda 関数のトリガーとして追加すると、Amazon SNS トピックに発行するたびに Lambda 関数がトリガーされます。タスクトークンを使用して Amazon SNS トピックに発行すると、発行されたメッセージのペイロードで Lambda 関数が呼び出されます。Lambda 関数のトリガー設定の詳細については、「AWS Lambda デベロッパーガイド」の「トリガーの設定」を参照してください。

1. `callback-human-approval` Lambda 関数の [関数の概要] セクションで、[トリガーを追加] を選択します。
2. トリガーのドロップダウンリストから、トリガーとして [SNS] を選択します。
3. [SNS トピック] には、このチュートリアルのステップ 1 で作成した Amazon SNS トピックの名前を入力し始め、表示されるドロップダウンリストから選択します。
4. [Add] (追加) を選択します。
5. 以降のアクションに進むには、このウィンドウを開いたまま次のステップに進みます。

ステップ 2.3: Lambda 関数の IAM ロールに必要なアクセス許可を付与する

`SendTaskSucess` API コールとともにタスクトークンを返すには、`callback-human-approval` Lambda 関数に Step Functions へのアクセス許可を付与する必要があります。

1. [callback-human-approval] ページで、[設定] タブを選択し、次に [アクセス許可] を選択します。

2. [実行ロール] で、[ロール名] を選択し、AWS Identity and Access Management コンソールの [ロール] ページに移動します。
3. 必要なアクセス許可を追加するには、[アクセス許可の追加] を選択し、次に [ポリシーをアタッチ] を選択します。
4. [検索] ボックスに、**`AWSStepFunctions`** と入力して Enter を押します。
5. [AWSStepFunctionsFullAccess] を選択し、下にスクロールして [ポリシーをアタッチ] を選択します。これにより、`callback-human-approval` Lambda 関数ロールに必要なアクセス許可を含むポリシーが追加されます。

ステップ 3: ワークフローの更新 - 選択状態に if-else 条件ロジックを追加する

Step Functions コンソールで、`Choice` 状態を使用してワークフローの条件ロジックを定義します。`RandomNumberforCredit` Lambda 関数によって返される出力が 5,000 未満の場合、リクエストされたクレジットは自動的に承認されます。返される出力が 5,000 以上の場合、ワークフローの実行は人による操作ステップに進み、クレジット限度額額を承認します。

`Choice` 状態では、比較演算子を使用して、入力変数を特定の値と比較します。ステートマシンの実行開始時に、入力変数を実行入力として指定することも、直前のステップの出力を現在のステップの入力として使用することもできる。デフォルトでは、ステップの出力は `Payload` と呼ばれる変数に保存されます。`Payload` 変数の値を `Choice` 状態内の比較に使用するには、次の手順に示す `$` 構文を使用しま。

ある状態から別の状態への情報の流れや、ワークフローでの入出力の指定方法については、「チュートリアル 7: 入力と出力を構成する」および「Step Functions の入出力処理」を参照してください。

> **Note**
>
> `Choice` 状態によって、比較のためにステートマシンの実行入力で指定された入力変数を使用する場合は、`$.variable_name` 構文を使用して比較を実行します。例えば、`myAge` などの変数を比較するには、`$.myAge` 構文を使用します。

このステップでは、`Choice` 状態が クレジット限度額を取得状態からの入力を受け取ることになるため、`$` 構文を使用して `Choice` 状態を設定します。`$.variable_name` 状態の設定で、`Choice` 構文を使用して直前のステップの出力を参照する場合、ステートマシンの実行結果がどのように異な

るかを調べるには、チュートリアル 8 の「無効なパス選択状態のエラーのデバッグ セクション」を参照してください。

Choice 状態を使用して if-else 条件ロジックを追加するには

1. チュートリアル 1: ステートマシンのプロトタイプを作成する で作成したワークフロープロトタイプを含む、Step Functions コンソールウィンドウを開きます。
2. [適用したクレジット > = 5000?] 状態を選択します。[設定] タブで、次のように条件付きロジックを指定します。

   a. [選択ルール] で、[ルール #1] タイルの [編集] アイコンを選択し、最初の選択ルールを定義します。

   b. [条件を追加] を選択します。

   c. [ルール #1 の条件] ダイアログボックスで、[変数] に **$** を入力します。

   d. [演算子] には [未満] を選択します。

   e. [値] で [定数] を選択し、[値] ドロップダウンリストの横にあるフィールドに **5000** を入力します。

   f. [条件を保存する] を選択します。

   g. [次の状態:] ドロップダウンリストで、[自動承認の上限] を選択します。

   h. [新しい選択ルールを追加] を選択し、サブステップ 2.b から 2.f を繰り返して、クレジット金額が 5000 以上になったら 2 つ目の選択ルールを定義します。[演算子] で、[以上] を選択します。

   i. [次の状態:] ドロップダウンリストには、[人による承認待ち] を選択します。

   j. [デフォルトのルール] ボックスで、[編集] アイコンを選択してデフォルトの選択ルールを定義し、[デフォルトの状態]ドロップダウンリストから [人による承認待ち] を選択します。デフォルトのルールは、[選択] 状態の条件がどれも「true」または「false」と評価されなかった場合に、遷移する次の状態を指定するために定義します。

3. 次のように、[人による承認待ち] 状態を設定します。

   a. [設定] タブの [トピック] に、Amazon SNS トピックの名前 [TaskTokenTopic] を入力し始め、ドロップダウンリストに表示される名前を選択します。

   b. [メッセージ] には、ドロップダウンリストから [メッセージを入力] を選択します。[メッセージ] フィールドで、Amazon SNS トピックに公開するメッセージを指定します。このチュートリアルでは、タスクトークンをメッセージとして公開します。

タスクトークンを使用すると、外部プロセスが完了してタスクトークンが返されるまで、[Standard] タイプの Step Functions ワークフローを一時停止できます。`.waitForTaskToken` サービス統合パターンを指定して、タスク状態をコールバックタスクとして指定すると、タスクの開始時にタスクトークンが生成され、コンテキストオブジェクトに配置されます。コンテキストオブジェクトは、実行中に使用できる内部の JSON 構造であり、ステートマシンとその実行に関する情報が含まれています。コンテキストオブジェクトの詳細については、「コンテキストオブジェクト」を参照してください。

c. 表示されたボックスで、メッセージとして次のように入力します。

```
{
  "TaskToken.$": "$$.Task.Token"
}
```

d. [コールバックを待つ] チェックボックスをオンにします。

e. 表示されたダイアログボックスで [完了] を選択します。

4. 以降のアクションに進むには、このウィンドウを開いたままで次のチュートリアルに進みます。

次のステップ

次のチュートリアルでは、複数のタスクを並行して実行する方法を学習します。

チュートリアル 4: 並列で実行する複数のタスクを定義する

ここまで、ワークフローを順番に実行する方法を学習してきました。一方で、`Parallel` 状態を使用すれば、2 つ以上のステップを並列で実行できます。`Parallel` 状態があると、インタープリタは各分岐を同時に実行します。

`Parallel` 状態内の分岐は、どちらも同じ入力を受け取りますが、各分岐はそれぞれ固有の入力部分を処理します。Step Functions では、各分岐の実行が完了するまで待機してから次のステップに進みます。

このチュートリアルでは、並列状態を使用して、申請者の ID と住所を同時に確認します。

トピック

- ステップ 1: Lambda 関数を作成して必要な確認を行う
- ステップ 2: ワークフローの更新 - 並列で実行するタスクを追加する

ステップ 1: Lambda 関数を作成して必要な確認を行う

このクレジットカード申請ワークフローでは、並列状態内で 2 つの Lambda 関数を呼び出して、申請者の ID と住所を確認します。これらの確認は、並列状態を使用して同時に実行します。ステートマシンが実行を完了するのは、並列分岐の両方が実行を完了した後に限ります。

[check-identity] と [check-address] の確認の Lambda 関数を作成するには

1. 新しいタブまたはウィンドウで Lambda コンソールを開き、`check-identity` と `check-address` いうタイトルの 2 つのNode.js 16.x Lambda 関数を作成します。コンソールを使用した Lambda 関数の作成については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda の関数の作成」を参照してください。
2. [check-identity] 関数ページを開き、[コードソース] 領域の既存のコードを次のコードに置き換えます。

```
const ssnRegex = /^\d{3}-?\d{2}-?\d{4}$/;
const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;

class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = "CustomValidationError";
    }
}

exports.handler = async (event) => {
    const {
        ssn,
        email
    } = event;
    console.log(`SSN: ${ssn} and email: ${email}`);

    const approved = ssnRegex.test(ssn) && emailRegex.test(email);

    if (!approved) {
        throw new ValidationError("Check Identity Validation Failed");
    }

    return {
        statusCode: 200,
        body: JSON.stringify({
            approved,
```

```
            message: `Identity validation ${approved ? 'passed' : 'failed'}`
        })
    }
};
```

3. [check-address] 関数ページを開き、[コードソース] 領域の既存のコードを次のコードに置き換えます。

```
class ValidationError extends Error {
  constructor(message) {
      super(message);
      this.name = "CustomAddressValidationError";
  }
}

exports.handler = async event => {
  const {
      street,
      city,
      state,
      zip
  } = event;
  console.log(`Address information: ${street}, ${city}, ${state} - ${zip}`);

  const approved = [street, city, state, zip].every(i => i?.trim().length > 0);

  if (!approved) {
      throw new ValidationError("Check Address Validation Failed");
  }

  return {
      statusCode: 200,
      body: JSON.stringify({
          approved,
          message: `Address validation ${ approved ? 'passed' : 'failed'}`
      })
  }
};
```

4. 両方の Lambda 関数で、[関数の概要]セクションから、それぞれの Amazon リソースネーム (ARN) をコピーし、テキストファイルに保存します。[申請者の ID と住所の確認] 状態のサービス統合を指定する場合は、関数の ARN が必要になります。ARN の例を次に示します。

```
arn:aws:lambda:us-east-2:123456789012:function:HelloWorld
```

ステップ 2: ワークフローの更新 - 並列で実行するタスクを追加する

Step Functions コンソールで、ワークフローを更新して、ステップ 1 で作成した [check-identity] Lambda 関数と [check-address] Lambda 関数とのサービス統合を指定します。

ワークフローに並列タスクを追加するには

1. チュートリアル 1: ステートマシンのプロトタイプを作成する で作成したワークフロープロトタイプを含む、Step Functions コンソールウィンドウを開きます。
2. [ID の検証] 状態を選択し、[構成]タブで次の操作を行います。

   a. [統合タイプ] は、デフォルトの [最適化] のままにします。

   > **Note**
   >
   > Step Functions を使用すると、他の AWS のサービス と統合して、ワークフロー内でそれらを調整できます。サービス統合とタイプの詳細については、「AWS Step Functions 他のサービスとの併用」を参照してください。

   b. [関数名] には、ドロップダウンリストから [check-identity] Lambda 関数を選択します。

   c. [ペイロード] には、[ペイロードを入力] を選択し、サンプルのペイロードを次のものに置き換えます。

   ```
   {
       "email": "janedoe@example.com",
       "ssn": "012-00-0000"
   }
   ```

3. [住所の確認] 状態を選択し、[構成]タブで次の操作を行います。

   a. [統合タイプ] は、デフォルトの [最適化] のままにします。

   b. [関数名] には、ドロップダウンリストから [check-address] Lambda 関数を選択します。

   c. [ペイロード] には、[ペイロードを入力] を選択し、サンプルのペイロードを次のものに置き換えます。

```
{
    "street": "123 Any St",
    "city": "Any Town",
    "state": "AT",
    "zip": "01000"
}
```

4. [次へ] をクリックします。

チュートリアル 5: 項目のコレクションを同時に反復処理する

前回のチュートリアルでは、Parallel 状態を使用して、ステップの個別の分岐を並行して実行する方法を学習しました。Map 状態を使用すると、データセット内の各項目に対して一連のワークフローステップを実行できます。Map 状態の反復は並列で実行されるため、データセットを迅速に処理できます。

Map 状態をワークフローに含めることにより、インラインモードと分散モードの 2 つの マップステート処理モード のどちらかを使用して、データ処理などのタスクを実行できます。Map 状態を設定するには、ItemProcessor を定義します。これには Map 状態の処理モードと、その定義を指定する JSON オブジェクトが含まれます。このチュートリアルでは、Map 状態をデフォルトのインラインモードで実行します。このモードでは、最大 40 回の同時反復がサポートされています。分散モードで Map 状態を実行すると、最大 10,000 の並列の子ワークフローの実行がサポートされます。

ワークフロー実行が Map 状態に入力されると、ステート入力で指定された JSON 配列を反復処理します。各配列項目に対し、その Map 状態を含むワークフローのコンテキストで、対応する反復処理が実行されます。すべての反復が完了すると、Map 状態は ItemProcessor によって処理された各項目の出力を含む配列を返します。

このチュートリアルでは、Map 状態をインラインモードで使用し、一連の信用調査機関を反復処理して、申請者のクレジットスコアを取得する方法を学習します。これを行うには、まず Amazon DynamoDB のテーブルに保存されているすべての信用調査機関の名前を取得し、次に Map 状態を使用して信用調査機関のリストをループ処理して、各機関から報告された申請者のクレジットスコアを取得します。

トピック

- ステップ 1: すべての信用調査機関の名前を保存する DynamoDB テーブルを作成する
- ステップ 2: ステートマシンの更新 - DynamoDB テーブルから結果を取得する

- ステップ 3: すべての信用調査機関のクレジットスコアを返す Lambda 関数を作成する
- ステップ 4: ステートマシンの更新 - マップステートを追加してクレジットスコアを反復取得する

ステップ 1: すべての信用調査機関の名前を保存する DynamoDB テーブルを作成する

このステップでは、DynamoDB コンソールを使用して **GetCreditBureau** という名前のテーブルを作成します。このテーブルでは、文字列属性の Name をパーティションキーとして使用します。このテーブルには、申請者のクレジットスコアを取得するすべての信用調査機関の名前を格納します。

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールのナビゲーションペインで、[テーブル] を選択し、次に [テーブルを作成] を選択します。
3. テーブルの詳細を次のように入力します。
   a. [Table name] (テーブル名) に「**GetCreditBureau**」と入力します。
   b. [パーティションキー] に「**Name**」と入力します。
   c. デフォルトの選択を維持して、[テーブルを作成] を選択します。
4. テーブルを作成したら、[テーブル]リストで [GetCreditBureau] テーブルを選択します。
5. [アクション] を選択し、次に [項目を作成] を選択します。
6. [値] には、信用調査機関の名前を入力します。例えば、**CredTrack** です。
7. [項目を作成] を選択します。
8. このプロセスを繰り返して、他の信用調査機関の名前の項目を作成します。例えば、**KapFinn** と **CapTrust** です。

ステップ 2: ステートマシンの更新 - DynamoDB テーブルから結果を取得する

Step Functions コンソールで `Task` 状態を追加し、AWS SDK 統合を使用して、ステップ 1 で作成した DynamoDB テーブルから信用調査機関の名前を取得します。このステップの出力は、このチュートリアルのワークフローに追加する `Map` の入力として使用します。

1. [CreditCardWorkflow] ステートマシンを開き、これを更新します。

2. [信用調査機関のリストを取得] を選択します。
3. API パラメータには、[テーブル名] の値に **GetCreditBureau** を指定します。

ステップ 3: すべての信用調査機関のクレジットスコアを返す Lambda 関数を作成する

このステップでは、すべての信用調査機関の名前を入力として受け取り、これらの各信用調査機関の申請者のクレジットスコアを返す Lambda 関数を作成します。この Lambda 関数は、このチュートリアルのステップ 4 でワークフローに追加する `Map` 状態から呼び出されます。

1. Node.js 16.x Lambda 関数を作成し、これに **get-credit-score** という名前を付けます。
2. [get-credit-score] というタイトルのページで、次のコードを [コードソース] 領域に貼り付けます。

```
function getScore(arr) {
  let temp;
  let i = Math.floor((Math.random() * arr.length));
  temp = arr[i];
  console.log(i);
  console.log(temp);
  return temp;
}

const arrScores = [700, 820, 640, 460, 726, 850, 694, 721, 556];

exports.handler = (event, context, callback) => {
  let creditScore = getScore(arrScores);
  callback(null, "Credit score pulled is: " + creditScore + ".");
};
```

3. Lambda 関数をデプロイします。

ステップ 4: ステートマシンの更新 - マップステートを追加してクレジットスコアを反復取得する

Step Functions コンソールで、[get-credit-score] Lambda 関数を呼び出して、[信用調査機関のリストを取得] 状態によって返されたすべての信用調査機関の申請者のクレジットスコアを確認する `Map` 状態を追加します。

1. [CreditCardWorkflow] ステートマシンを開き、これを更新します。
2. [すべての信用調査機関からスコアを取得] 状態を選択します。
3. [設定] タブで、[項目配列へのパスを指定] を選択し、[**`$.Items`**] と入力します。
4. Map 状態内のステップで、[すべてのスコアを取得] を選択します。
5. [設定] タブで、[統合タイプ] に [最適化] が選択されていることを確認します。
6. [関数] 名に [get-credit-score] Lambda 関数の名前を入力し始め、表示されるドロップダウンリストから選択します。
7. [ペイロード] には、[ペイロードなし] を選択します。

チュートリアル 6: ワークフローを保存してステートマシンを実行する

ワークフロープロトタイプで使用しているすべてのリソースを設定したので、Step Functions ステートマシンとして保存して実行を開始できます。 AWS のサービス

トピック

- ステップ 1: 自動生成されたステートマシン定義を確認してステートマシンを保存する
- ステップ 2: 残りの IAM ポリシーを追加する
- ステップ 3: ステートマシンを実行する

ステップ 1: 自動生成されたステートマシン定義を確認してステートマシンを保存する

[フロー] タブから状態をキャンバスにドラッグアンドドロップすると、ワークフロープロトタイプが作成されます。Step Functions は、ワークフローの Amazon ステートメント言語 (ASL) の定義をリアルタイムで自動的に作成します。この定義は、コードエディタ で必要に応じて編集できます。

ASL 定義を確認してステートマシンを保存するには

1. (オプション) Inspector で [定義] を選択すると、ステートマシン Amazon ステートメント言語 (ASL) の定義が表示されます。この定義は、[アクション] タブと [フロー] タブ、[Inspector] パネルでの選択に基づいて自動的に生成されます。

> **Tip**
>
> 定義を編集するには、ページ上部の [コード] を選択してコードエディタを開きます。このチュートリアルでは、自動生成された定義のまま進めます。

2. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択しますMyStateMachine。次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**`CreditCardWorkflow`**」と入力します。

3. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

   > **Note**
   >
   > (オプション) Step Functions は、`RandomNumberforCredit` Lambda 関数を呼び出して Amazon SNS トピックに発行するために必要な最小特権を持つ、ステートマシンの実行ロールを自動的に作成します。

   ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

4. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

   > **Note**
   >
   > Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 2: 残りの IAM ポリシーを追加する

Step Functions は、`Parallel` 状態で使用される Lambda 関数を呼び出すアクセス許可を自動生成しないため、必要なポリシーを追加する必要があります。

残りのポリシーを追加するには

1. CreditCardWorkflowページで、ステートマシンの IAM ロールを選択して IAM コンソールに移動します。このページで、残りの Lambda 関数に必要な権限を追加します。
2. [アクセス許可を追加]、[ポリシーをアタッチ] の順に選択します。
3. [検索] ボックスに、**`AWSLambdaRole`** と入力して Enter を押します。
4. AWSLambdaRoleを選択し、[ポリシーをアタッチ] を選択します。これで、このポリシーがステートマシンの実行ロールに追加されました。このポリシーにより、ステートマシンで任意の Lambda 関数を呼び出すことができます。

ステップ 3: ステートマシンを実行する

ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。

ステートマシンを実行するには

1. CreditCardWorkflowページで [実行開始] を選択します。

   [実行を開始] ダイアログが表示されます。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

> **Note**
>
> このステートマシンを実行するために、入力を指定する必要はありません。ただし、必要に応じて、他のステートマシンの [実行を開始] ダイアログボックスの [入力] 領域で実行入力を指定できます。ステートマシンに実行入力を提供する方法の例については、「ステップ 4: AWS Step Functions Workflow Studio の使い方を学ぶ」チュートリアルの新しい実行を開始するを参照してください。

b. [実行のスタート] を選択します。

3. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

チュートリアル 7: 入力と出力を構成する

Step Functions を実行すると、JSON テキストを入力として受け取り、その入力をワークフローの最初の状態に渡します。ワークフローのそれぞれの状態で JSON データを入力として受け取ります。また、通常 JSON データを出力として次の状態に渡します。デフォルトでは、ワークフロー内の 1 つ以上の状態に対して入力や出力を設定していない限り、データはワークフローの 1 つの状態から次の状態に渡されます。Step Functions でワークフローを効率的に設計し、実装するには、状態間の情報の流れを理解して、このデータのフィルタリングし、操作する方法を学習することが重要です。

Step Functions には、状態間の入出力データフローを制御するための複数のフィルターが用意されています。ワークフローでは、次のフィルターを使用できます。

> **Note**
>
> ユースケースによっては、これらのフィルターをワークフローにすべて適用する必要がない場合があります。

InputPath

全体の入力ペイロードのうち、タスクの入力としてどの部分を使用するかを選択します。このフィールドを指定すると、Step Functions は、最初にこのフィールドを適用します。

#####

タスクを起動する前に、入力をどのように表示するかを指定します。Parameters フィールドを使用すると、AWS Lambda 関数などのAWS のサービス の統合の入力として渡されるキーと値のペアのコレクションを作成できます。これらの値は、静的にすることも、状態入力またはワークフローコンテキストオブジェクトから動的に選択することもできます。

ResultSelector

タスクの出力から何を選択するかを決定します。ResultSelector フィールドを使用すると、状態の結果を置き換えるキーと値のペアのコレクションを作成し、そのコレクションを ResultPath に渡すことができます。

ResultPath

タスクの出力をどこに置くかを決めます。ResultPath を使用して、状態の出力が、入力のコピー、生成された結果、またはその両方の組み合わせなのかを判断します。

OutputPath

次の状態に何を送るかを決めます。OutputPath を使用すると、不要な情報をフィルタリングして、JSON データのうちの必要な部分のみを渡すことができます。

> **Tip**
>
> Parameters と ResultSelector のフィルターは JSON を構築することにより機能し、InputPath と OutputPath フィルターは JSON データオブジェクト内の特定のノードをフィルタリングすることにより機能します。ResultPath フィルターは出力を追加できるフィールドを作成することにより機能します。

このチュートリアルでは、次のタスクを実行する方法を説明しています。

- フィルターを使用して、 InputPath 未加工の入力の特定の部分を選択します
- パラメータフィルタを使用して選択した入力を操作する
- ResultSelector、ResultPath、OutputPath フィルターを使用して出力を設定する

ワークフローの入力と出力の設定の詳細については、「Step Functions の入出力処理」を参照してください。

フィルターを使用して、 InputPath 未加工の入力の特定の部分を選択します

InputPath フィルターを使用して、入力ペイロードの特定の部分を選択します。

InputPath を指定しない場合、その値はデフォルトで $ になり、状態のタスクは特定の部分ではなく、未加工の入力全体を参照することになります。

InputPath フィルターの使用方法を理解するには、次のステップを実行します。

- ステップ 1: ステートマシンを作成する
- ステップ 2: ステートマシンを実行する
- ステップ 3: InputPath フィルターを使用して実行入力の特定の部分を選択する

ステップ 1: ステートマシンを作成する

> **Important**
>
> ステートマシンが、前に作成した Lambda AWS 関数と同じアカウントとリージョンにあることを確認します。

1. チュートリアル 4 で説明した Parallel 状態の例を使用して、新しいステートマシンを作成します。ワークフロープロトタイプは、次のプロトタイプのようになることを確認してください。
2. check-identity と check-address Lambda 関数の統合を構成します。Lambda 関数の作成と、ステートマシンでの使用については、「ステップ 1: Lambda 関数を作成して必要な確認を行う」と「ステップ 2: ワークフローの更新 - 並列で実行するタスクを追加する」を参照してください。
3. [ペイロード] は、デフォルトの [状態入力をペイロードとして使用] を選択したままにしてください。
4. [次へ] を選択し、チュートリアル 5 の ステップ 1: ステートマシンを保存する のステップ 1～3 を実行して、新しいステートマシンを作成します。このチュートリアルでは、ステートマシンに **WorkflowInputOutput** という名前を付けます。

ステップ 2: ステートマシンを実行する

1. WorkflowInputOutputページで [実行開始] を選択します。
2. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

3. [入力] 領域で、実行入力として次の JSON データを追加します。

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    }
  }
}
```

4. [実行のスタート] を選択します。
5. `check-identity` Lambda 関数と `check-address` Lambda 関数が、必要な ID と住所の確認を実行するために使用する必要がある実行入力の部分が指定されていないため、ステートマシンの実行結果はエラーになります。
6. このチュートリアルのステップ 3 に進み、エラーを修正してください。

ステップ 3: **InputPath** フィルターを使用して実行入力の特定の部分を選択する

1. [実行の詳細] ページで、[ステートマシンの編集] を選択します。
2. ステップ 2: ステートマシンを実行する に記載されている実行入力に従って申請者の ID を確認するには、[ID の検証] タスク定義を次のように編集します。

```
...
{
    "StartAt": "Verify identity",
    "States": {
      "Verify identity": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "InputPath": "$.data.identity",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:check-
identity:$LATEST"
        },
        "End": true
      }
    }
  }
  ...
```

その結果、次の JSON データが `check-identity` 関数の入力として使用できるようになります。

```
{
   "email": "jdoe@example.com",
   "ssn": "123-45-6789"
 }
```

3. 実行入力に記載されている申請者の住所を確認するには、`Verify address` タスク定義を次のように編集します。

```
...
{
    "StartAt": "Verify address",
    "States": {
      "Verify address": {
```

```
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "InputPath": "$.data.address",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:us-east-1:123456789012:function:check-
address:$LATEST"
        },
        "End": true
      }
    }
  }
  ...
```

その結果、次の JSON データが check-address 関数の入力として使用できるようになります。

```
{
    "street": "123 Main St",
    "city": "Columbus",
    "state": "OH",
    "zip": "43219"
}
```

4. [実行のスタート] を選択します。これで、ステートマシンの実行が正常に完了しました。

パラメータフィルタを使用して選択した入力を操作する

InputPath フィルターによって、指定した未加工の JSON 入力を制限しやすくなりますが、Parameters フィルターを使用すると、キーと値のペアのコレクションを入力として渡すことができます。これらのキーと値のペアは、ステートマシンの定義で定義した静的な値か、InputPath を使用して未加工の入力から選択された値のいずれかになります。

ワークフローでは、Parameters は InputPath の後で適用されます。Parameters は、基礎となるタスクが入力ペイロードをどのように受け入れるかを指定するのに役立ちます。例えば、check-address Lambda 関数が JSON データの代わりに文字列パラメータを入力として受け取る場合、Parameters フィルターを使用して入力を変換できます。

次の例では、Parameters フィルターは ステップ 3: InputPath フィルターを使用して実行入力の特定の部分を選択する の InputPath を使用して選択した入力を受け取り、その組み込み関数

States.Format を入力アイテムに適用して、addressString という文字列を作成します。組み込み関数は、特定の入力に対して基本的なデータ処理オペレーションを実行するのに役立ちます。詳細については「組み込み関数」を参照してください。

```
"Parameters": {
    "addressString.$": "States.Format('{}. {}, {} - {}', $.street, $.city, $.state,
 $.zip)"
  }
```

その結果、次の文字列が作成され、入力として check-address Lambda 関数に提供されます。

```
{
    "addressString": "123 Main St. Columbus, OH - 43219"
}
```

ResultSelector、ResultPath、OutputPath フィルターを使用して出力を設定する

WorkflowInputOutput ステートマシンで check-address Lambda 関数が呼び出されると、この関数は住所の確認を実行した後に出力ペイロードを返します。[実行の詳細] ページで [住所の確認] ステップを選択し、ステップの詳細 ペインの [タスク結果] 内の出力ペイロードを確認します。

```
{
    "ExecutedVersion": "$LATEST",
    "Payload": {
      "statusCode": 200,
      "body": "{\"approved\":true,\"message\":\"identity validation passed\"}"
    },
    "SdkHttpMetadata": {
      "AllHttpHeaders": {
        "X-Amz-Executed-Version": [
          "$LATEST"
        ],
        ...
        ...
    "StatusCode": 200
  }
```

ResultSelector の使用

ID と住所の確認結果をワークフローの次の状態に送る必要がある場合は、出力 JSON で [Payload.body] ノードを選択すると、ResultSelector フィルターの StringToJson 組み込み関数を使用して、必要に応じてデータをフォーマットできます。

ResultSelector は、タスク出力から何が必要かを選択します。次の例では、ResultSelector は $.Payload.body の文字列を取得し、States.StringToJson 組み込み関数を適用して文字列を JSON に変換して、結果の JSON を ID ノード内に置きます。

```
"ResultSelector": {
    "identity.$": "States.StringToJson($.Payload.body)"
  }
```

その結果、次の JSON データが作成されます。

```
{
    "identity": {
      "approved": true,
      "message": "Identity validation passed"
    }
  }
```

これらの入力フィルターと出力フィルターを操作すると、無効な JSON パス式を指定したことによりランタイムエラーが発生することもあります。詳細については、「」を参照してください。

ResultPath の使用

ResultPath フィールドを使用すると、最初の入力ペイロード内の場所を指定して、状態のタスク処理結果を保存できます。ResultPath を指定しない場合、その値はデフォルトで $ になり、最初の入力ペイロードは未加工のタスク結果に置き換えられます。ResultPath に null を指定すると、未処理の結果は破棄され、最初の入力ペイロードが有効な出力になります。

ResultSelector フィールドを使用して作成した JSON データに ResultPath フィールドを適用すると、次の例に示すように、タスクの結果が入力ペイロードの結果ノード内に追加されます。

```
{
  "data": {
    "firstname": "Jane",
```

```
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    },
  "results": {
    "identity": {
      "approved": true
    }
  }
}
```

OutputPath の使用

ResultPath を適用した後の状態出力の一部を選択して、次の状態に渡すことができます。これにより、不要な情報をフィルタして、必要な一部の JSON のみを渡すことができるようになります。

次の例では、OutputPath フィールドは状態出力を、次の結果ノード「"OutputPath": "$.results"」内に保存します。したがって、状態の最終出力は次のようになります。これを次の状態に渡すことができます。

```
{
    "addressResult": {
      "approved": true,
      "message": "address validation passed"
    },
    "identityResult": {
      "approved": true,
      "message": "identity validation passed"
    }
}
```

コンソール機能を使用して入出力データフローを視覚化する

Step Functions コンソールのデータフローシミュレーターまたは実行の詳細ページの詳細表示オプションを使用して、ワークフローの状態間の入出力データフローを視覚化できます。

チュートリアル 8: コンソールでエラーをデバッグする

Step Functions を操作していると、次のような理由でランタイムエラーが発生することがあります。

- Choice 状態の [変数] フィールドの JSON パスが無効です。
- ステートマシンの定義に問題 (例えば、Choice 状態に一致するルールが定義されていない) があります。
- JSON パス式が無効です (フィルターを適用して入出力を操作する際)。
- Lambda 関数の例外が原因でタスクが失敗しました。
- IAM アクセス許可エラー。

このチュートリアルでは、Step Functions コンソールを使用してこれらのエラーの一部をデバッグする方法を学習します。詳細については「Step Functions のエラー処理」を参照してください。

トピック

- 無効なパス選択状態のエラーのデバッグ
- 入力フィルターと出力フィルターの適用中の JSON パス式エラーのデバッグ

無効なパス選択状態のエラーのデバッグ

Choice 状態の [変数] フィールドに正しくない、または解決できない JSON パスを指定した、Choice 状態にマッチングルールを定義しなかった場合、ワークフローの実行中にエラーが表示されます。

無効なパスのエラーを説明するため、このチュートリアルではワークフローの Choice 状態エラーを紹介します。CreditCardWorkflow ステートマシンを使用し、その定義を編集してエラーを発生させます。

1. Step Functions コンソールを開き、[CreditCardWorkflow] を選択します。
2. [編集] を選択して、ステートマシン定義を編集します。次のコードで強調表示されている変更を、ステートマシン定義に加えます。

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Get credit limit",
  "States": {
```

```
    "Get credit limit": {
      ...
      ...
    },
    "Credit applied >= 5000?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.Payload",
          "NumericLessThan": 5000,
          "Next": "Auto-approve limit"
        },
        {
          "Variable": "$.Payload",
          "NumericGreaterThanEquals": 5000,
          "Next": "Wait for human approval"
        }
        ],
        "Default": "Wait for human approval"
    },
    ...
    ...
  }
}
```

3. [保存] を選択し、次に [このまま保存する] を選択します。
4. ステートマシンを実行します。
5. ステートマシン実行の [実行の詳細] ページで、次のいずれかを実行します。
   a. エラーメッセージの [原因] を選択すると、実行が失敗した理由が表示されます。
   b. エラーメッセージの [ステップの詳細を表示] を選択すると、エラーの原因となったステップが表示されます。
6. [ステップの詳細] セクションの [入力と出力] タブで、[アドバンスドビュー] トグルボタンを選択すると、選択したステートの入出力データの転送パスが表示されます。
7. [グラフビュー] で、[適用したクレジット >= 5000?] が選択されていることを確認し、次の操作を実行します。
   a. [入力] ボックスの状態の入力値を表示します。
   b. [定義] タブを選択し、[変数] フィールドに指定されている JSON パスを確認します。

[適用したクレジット >= 5000?] 状態の入力値は数値ですが、入力値の JSON パスは `$.Payload` と指定されています。この JSON パスは、ステートマシンの実行中には存在しないため、`Choice` 状態はこの JSON パスを解決できません。

8. ステートマシンを編集して [変数] フィールドの値を **$** に指定します。

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Get credit limit",
  "States": {
    "Get credit limit": {
      ...
      ...
    },
    "Credit applied >= 5000?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$",
          "NumericLessThan": 5000,
          "Next": "Auto-approve limit"
        },
        {
          "Variable": "$",
          "NumericGreaterThanEquals": 5000,
          "Next": "Wait for human approval"
        }
        ],
        "Default": "Wait for human approval"
    },
    ...
    ...
  }
}
```

入力フィルターと出力フィルターの適用中の JSON パス式エラーのデバッグ

入力フィルターと出力フィルターを操作すると、無効な JSON パス式を指定したことによりランタイムエラーが発生することがあります。

次の例では、チュートリアル 5 で作成した WorkflowInputOutput ステートマシンを使用し、ResultSelector フィルターを使用してタスク出力の一部を選択するシナリオを示しています。

1. ResultSelector フィルターを適用して、[ID の確認] ステップのタスク出力の一部を選択します。これを行うには、次のようにステートマシン定義を編集します。

```
{
  "StartAt": "Verify identity",
  "States": {
    "Verify identity": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:check-
identity",
        "Payload": {
          "email": "jdoe@example.com",
          "ssn": "123-45-6789"
        }
      },
      ...
      ...
      "ResultSelector": {
        "identity.$": "$.Payload.body.message"
 }",
      "End": true
    }
  }
}
```

2. ステートマシンを実行します。
3. ステートマシン実行の [実行の詳細] ページで、次の手順を実行します。
   a. エラーメッセージの [原因] を選択すると、実行が失敗した理由が表示されます。
   b. エラーメッセージの [ステップの詳細を表示] を選択すると、エラーの原因となったステップが表示されます。
4. エラーメッセージでは、$.Payload.body ノードの内容がエスケープされた JSON 文字列であることに注意してください。このエラーは、JSON パス表記を使用して文字列を参照できないために発生しました。

5. $.Payload.body.message ノードを参照するには、次の操作を行います。

   a. States.StringToJSON 組み込み関数を使用して、最初に文字列を JSON 形式に変換します。

   b. 組み込み関数内の $.Payload.body.message ノードの JSON パスを指定します。

```
"ResultSelector": {
 "identity.$":"States.StringToJson($.Payload.body.message)"
}
```

6. ステートマシンを再度実行します。

ユースケース

AWS Step Functions では、ビジネス要件をアプリケーションにすばやく変換するのに役立つビジュアルワークフローを構築できます。Step Functions は、状態、チェックポイント、および再起動を管理し、エラーや例外を自動的に処理する組み込み機能を提供します。Step Functions が提供する機能への理解を深めるため、次のユースケースを読み取ります。

トピック

- データ処理
- Machine Learning
- マイクロサービスのオーケストレーション
- IT およびセキュリティのオートメーション

データ処理

多様化が進むソースからのデータのボリュームが増加するにつれて、組織はこのデータを処理に向けて迅速に移行して、情報に基づいたビジネス上の決定を迅速に下す必要があると認識しています。大規模なデータを処理するには、モバイルデバイス、アプリケーション、衛星、マーケティングと販売、運用データ保存、インフラストラクチャなどから受け取る情報管理のためにリソースを伸縮自在にプロビジョンする必要があります。

Step Functions は、データ処理ワークフローを正常に管理するために必要なスケーラビリティ、信頼性、および可用性を提供します。水平方向にスケールし、耐障害性ワークフローを提供するため、Step Functions で何百万もの同時実行を管理できます。Step Functions ## 状態タイプのような並列実行を使用してデータを高速処理またはその ##### 状態タイプを使った動的並列処理。ワークフローの一部として、##### 状態を使って、Amazon S3 バケットのような静的データストア内のオブジェクトを反復処理します。Step Functions を使用すると、複雑なプロセスの管理をする必要はなく、実行が失敗しても簡単に再試行したり、エラーを処理する特定の方法を選択できます。

データ処理のニーズに応じて、Step Functions は　バッチ処理用の AWS Batch、ビッグデータ処理用の Amazon EMR、データ準備用の AWS Glue、データ分析用の Athena、およびコンピューティング用の AWS Lambda など AWS によって提供されている他のデータ処理サービスと直接統合します。

顧客が Step Functions を使用して達成するデータ処理ワークフローのタイプの例を次に示します。

ファイル、動画、およびイメージ処理

- 動画ファイルを収集し、携帯電話、ラップトップ、テレビなど、表示されるデバイスに最適な他のサイズや解像度に変換します。
- ユーザーがアップロードした大量の写真のコレクションを撮り、サムネイルやさまざまな解像度の画像に変換して、ユーザーのウェブサイトに表示できます。
- CSV ファイルなどの半構造化データを取得し、それを請求書などの非構造化データと結合して、ビジネスステークホルダーに毎月送信されるビジネスレポートを作成します。
- 衛星から収集された地球観測データを取得し、それを互いに合う形式に変換し、地球上で収集された他のデータソースを追加してインサイトを高めます。
- 製品のさまざまな輸送モードから輸送ログを取得し、モンテカルロシミュレーションを使用して最適化を求め、商品の出荷に関して頼りにされている組織や人にレポートを送り返します。

座標抽出、変換、ロード (ETL) ジョブ:

- AWS Glue を使用して、一連のデータ準備手順を通じて、レコードとマーケティングメトリクスデータセットを組み合わせ、組織全体で使えるビジネスインテリジェンスレポートを作成します。
- ビッグデータ処理のために Amazon EMR クラスターを作成、スタート、終了します。

Batch 処理およびハイパフォーマンスコンピューティング (HPC) ワークロード:

- 生の全ゲノム配列をバリアントコールで処理するゲノム二次解析パイプラインを構築します。生ファイルをリファレンスシーケンスに整列させ、動的並列処理を使用して、指定された染色体リストのバリアントを呼び出します。
- さまざまな電気化学物質を使用してさまざまなレイアウトをシミュレーションして、次のモバイルデバイスやその他の電子機器の生産効率を見出します。さまざまなシミュレーションを通じてワークロードの大規模なバッチ処理を実行し、最適な設計を実現します。

Machine Learning

機械学習により、組織は収集したデータを迅速に分析してパターンを特定し、最小限の人間の介入で意思決定を行うことができます。機械学習は、トレーニングデータと呼ばれる初期データセットから始まります。このトレーニングデータは、機械学習モデルの予測精度を高めるのに役立ち、このモデルが学習する基礎となります。モデルがビジネスニーズを満たせる精度があると判断されると、本番環境にデプロイされます。AWS Step Functions データサイエンスソフトウェア開発キット (SDK) は、Amazon SageMaker と Step Functions を使用してデータを前処理し、モデルをトレーニングし、パブリッシュするワークフローを簡単に作成できるオープンソースライブラリです。

既存のデータセットを前処理することは、組織が頻繁にトレーニングデータを作成する方法です。このメソッドは、イメージ内のオブジェクトにラベルを付けたり、テキストに注釈を付けたり、オーディオを処理したりするなど、情報を追加します。データを前処理するには、AWS Glue を使用するか、Jupyter Notebook アプリケーションを実行する SageMaker ノートブックインスタンスを作成することもできます。データの準備ができたら、簡単にアクセスできるように Amazon S3 にアップロードできます。機械学習モデルのトレーニング時に、各モデルのパラメータを調整して、デプロイの準備が整うまで精度を向上できます。

Step Functions を使うと、SageMaker 上でエンドツーエンドの機械学習ワークフローをオーケストレートできます。これらのワークフローには、データの前処理、後処理、特徴エンジニアリング、データ検証、およびモデル評価が含まれます。モデルを本番環境に導入したら、新しいアプローチを改良してテストし、ビジネス成果を継続的に改善できます。本番環境に対応したワークフローを Python で直接作成できるだけでなく、 Step Functions データサイエンス SDK を使って、新しいオプションでそのワークフロー経験をコピーし、洗練されたワークフローを実稼働環境に配置することもできます。

お客様が Step Functions を使用する機械学習ワークフローには、次のようなものがあります。

不正検出

- 信用詐欺などの不正取引を特定し、防止します。
- トレーニング済みの機械学習モデルを使用して、アカウントの乗っ取りを検出して防止します。
- 偽アカウントの作成など、プロモーション上の不正使用を特定して、すぐにアクションを起こすことができます。

パーソナライゼーションとレコメンデーション

- 関心を引き付けると予測される内容に基づいて、ターゲットを絞った顧客に製品をレコメンデーションします。
- 顧客がアカウントを無料利用枠から有料サブスクリプションにアップグレードするかどうかを予測します。

データエンリッチメント

- データエンリッチメントを前処理の一部として使用して、より正確な機械学習モデルのためのより良いトレーニングデータを提供します。
- テキストとオーディオの抜粋に注釈を付けて、皮肉やスラングなどの構文情報を追加します。

- イメージ内の追加のオブジェクトにラベル付けをし、オブジェクトがリンゴ、バスケットボール、岩、動物であるかどうかなど、モデルの学習に必要な重要な情報を提供します。

マイクロサービスのオーケストレーション

マイクロサービスアーキテクチャは、アプリケーションを疎結合サービスに分割します。スケーラビリティの向上、耐障害性の向上、市場投入までの時間の短縮などの利点があります。各マイクロサービスは独立しているため、アプリケーション全体をスケールすることなく、単一のサービスまたは関数を簡単にスケールアップできます。個々のサービスは疎結合されているため、独立したチームはアプリケーション全体を理解する必要なく、単一のビジネスプロセスに集中できます。また、マイクロサービスでは、ビジネスニーズに合った個々のコンポーネントを選択できるため、ワークフロー全体を書き直すことなく柔軟に選択を変更できます。様々なチームが、選択したプログラミング言語とフレームワークを使用してマイクロサービスで作業できます。また、このマイクロサービスは、アプリケーションプログラムインターフェイス (API) を介してアプリケーション内の他のものと引き続き通信できます。

Step Functions には、マイクロサービスのワークフローを管理する方法がいくつか用意されています。長時間稼働するワークフローでは、コンテナで実行されているアプリケーションをオーケストレートする AWS Fargate 統合機能があるスタンダードワークフローを使用できます。即時対応が必要な、短期間の大量のワークフローの場合、同期 Express ワークフローが理想的です。このワークフローは、ウェブベースまたはモバイルアプリケーションに使用できます。これらのアプリケーションでは、ワークフローが短いことが多いため、応答を返す前に一連のステップを完了する必要があります。Amazon API Gateway から同期 Express ワークフローを直接トリガーでき、ワークフローが完了するかタイムアウトするまで接続は開いたままになります。ただちに応答を必要としない短期間のワークフローの場合、Step Functions は非同期　Express ワークフローを提供します。

Step Functions を使用するいくつかの API オーケストレーションの例を次に示します。

同期ワークフローまたはリアルタイムワークフロー

- 従業員の姓の更新など、レコード内の値を変更し、変更をすぐに画面に表示します。
- 項目の追加、削除、数量の変更など、チェックアウト中に注文を更新し、すぐに更新を顧客に反映します。
- クイック処理ジョブを実行し、結果をすぐにリクエスタに戻します。

コンテナオーケストレーション

- Amazon Elastic Kubernetes Service を使って Kubernetes でジョブを実行するか、Fargate を使用して Amazon Elastic Container Service (ECS) でジョブを実行し、同じワークフローの一部として Amazon SNS を使った通知を送信など、他の AWS のサービスと統合します。

IT およびセキュリティのオートメーション

IT オートメーションは、ソフトウェアのアップグレードとパッチ適用、脆弱性に取り組むためセキュリティ更新プログラムのデプロイ、インフラストラクチャの選択、データの同期、サポートチケットのルーティングなど、ますます複雑で時間のかかる操作の管理に役立ちます。反復的で時間のかかるタスクのオートメーションにより、組織は日常業務を迅速かつ一貫性を持たせながらラージスケールで完了する許可が与えられます。これにより、特徴開発、複雑なサポート要求、イノベーションなどの戦略的な作業に集中しながら、これらの増大する要求に応えることができます。

Step Functions では、マニュアルによる介入を必要とせずに、ビジネスのニーズに合わせて自動的にスケールするワークフローを作成できます。ワークフローでエラーが発生した場合は、マニュアルによる介入が必要ないことがよくあります。Step Functions を使用すると、自動的に、失敗したタスクの再試行そしてワークフローのエラーを管理できるエクスポネンシャルバックオフを実行できます。

ワークフローが進行する前に、人間の介入が必要になる状況が発生する可能性があります。例えば、大幅なクレジット増加を承認するには、人間の承認が必要かもしれません。これを管理するには、Step Functions で分岐ロジックを定義して、定義した量を超えるリクエストのみが人間の承認を必要とし、他のすべてのリクエストは自動的に完了するようにします。人による承認が必要な場合、Step Functions を使用すると、特定のステップでワークフローを一時停止し、応答を待ち、応答を受信した後にワークフローを続行できます。

顧客が Step Functions を使用するオートメーションワークフローのタイプには、次のようなものがあります。

IT オートメーション

- SSH ポートを開く、ディスク容量が少ない、Amazon S3 バケットに公開アクセスが付与された場合などのインシデントを自動修復します。
- AWS CloudFormation StackSets のデプロイをオートメーションにする

セキュリティオートメーション

- ユーザーおよびユーザーアクセスキーが公開されているシナリオへの応答を自動化します。

- アクションを特定の ARN に制限したり、他のアクションを適用するなど、定義されたポリシーアクションに従って、セキュリティインシデントレスポンスを自動修正します。
- 受信後数秒以内にフィッシングメールを従業員に警告します。

[Human Approval] (ヒューマン承認)

- 機械学習モデルのトレーニングをオートメーションにし、受け取った応答に基づいてモデルを自動的にデプロイまたは拒否する前に、データサイエンティストによるモデルのマニュアル承認を要求します。
- センチメント分析に基づいて受け取った顧客からのフィードバックのルーティングを自動化して、否定的なセンチメントを持つものはすぐにエスカレーションしてマニュアルレビューを行います。

Step Functions 仕組み

このセクションでは、AWS Step Functions に慣れ親しんで理解するための重要な概念について説明します。

トピック

- 標準ワークフロー対 Express ワークフロー
- 状態
- マップステート処理モード
- 分散マップ状態の許容される失敗しきい値
- Transitions
- ステートマシンデータ
- Step Functions の入出力処理
- データフローシミュレータ
- バージョニングとエイリアスによる継続的デプロイの管理
- Step Functions で実行
- Step Functions のエラー処理
- 他のサービスから AWS Step Functions の呼び出し
- Step Functions 読み取り整合性
- Step Functions でのタグ付け

標準ワークフロー対 Express ワークフロー

ステートマシンを作成するときは、[標準] または [Express] のいずれかの [タイプ] を選択します。ステートマシンのデフォルトタイプは標準です。タイプ が 標準のステートマシンは標準ワークフローと呼ばれ、タイプが Express のステートマシンは Express ワークフローと呼ばれます。

標準と Express のどちらのワークフローでも、Amazon ステートメント言語 を使用してステートマシンを定義します。ステートマシンの実行の動作は、選択した[タイプ]によって異なります。

> **Important**
>
> ステートマシンが作成された後は、選択した [タイプ] を変更することはできません。

> **Note**
>
> Step Functions のコンソールの外部 (任意のエディタなど) でステートマシンを定義する場合は、ステートマシン定義を拡張子.asl.json を付けて保存する必要があります。

Standard ワークフローは、長時間実行され (最長 1 年)、耐久性が高く、監査可能なワークフローに最適です。実行完了後 90 日以内であれば Step Functions API を使用して完全な実行履歴を取得できます。標準ワークフローは 一度だけのモデルを採用しており、ASL で `Retry` 動作を指定しない限り、タスクと状態が複数回実行されることはありません。これにより、Amazon EMR クラスターのスタートや支払い処理など、非べき等アクションのオーケストレーションに適した標準ワークフローになります。標準ワークフローの実行は、処理された状態遷移の数に応じて課金されます。

Express ワークフローは、IoT データの取り込み、ストリーミングデータ処理と変換、モバイルアプリケーションのバックエンドなど、大容量のイベント処理ワークロードに最適です。これらのワークフローは最大 5 分間実行できます。Express ワークフローは、少なくとも 1 回は実行されるモデルですが、複数回実行される可能性もあります。これにより Express ワークフローは、入力データの変換や PUT アクションによる Amazon DynamoDB での保存など、べき等アクションの調整に最適になります。Express ワークフローの実行は、実行回数、実行時間、消費されたメモリによって、実行中に課金されます。

標準ワークフローと Express ワークフローは、Amazon API Gateway (大規模なフルマネージド型) からの HTTP リクエスト、IoT ルール、および、Amazon EventBridge 内の 140 を超えるその他のイベントソースなどのイベントに応答して自動的に開始できます。

> **Tip**
>
> Express ワークフローの例を AWS アカウント にデプロイするには、「AWS Step Functions ワークショップ」の「モジュール 7 – API Gateway、パラレルステート、Express ワークフロー」を参照してください。

標準ワークフローと Express ワークフローの実行のコンソールエクスペリエンスについては、「コンソールでの Standard ワークフローと Express ワークフローの実行」を参照してください。

標準ワークフローと Express ワークフロー

	標準ワークフロー	エクスプレスワークフロー:同期および非同期
最大期間	1 年	5 分
サポートされている実行開始レート	サポートされている実行開始レートに関連するクォータについては、「API アクションのスロットリングに関連するクォータ」を参照してください。	サポートされている実行開始レートに関連するクォータについては、「API アクションのスロットリングに関連するクォータ」を参照してください。
サポートされている状態遷移レート	サポートされている状態遷移レートに関連するクォータについては、「状態のスロットリングに関連するクォータ」を参照してください。	無制限
料金表	状態遷移回数による価格設定。状態遷移は、実行のステップが完了するたびにカウントされます。	実行回数、実行時間、およびメモリ消費量によって価格設定されます。
実行履歴	実行は、Step Functions API を使用して一覧表示および記述できます。実行は、コンソールから視覚的にデバッグできます。ステートマシンでのログを有効にすることで、CloudWatch Logs で実行を検査することもできます。 コンソールでの標準ワークフロー実行のデバッグの詳細については、「コンソールでの Standard ワークフローと Express ワークフローの実	実行履歴は無制限に、つまり 5 分以内に生成できる限り多くの実行履歴エントリが保持されます。 ステートマシンでログ記録を有効にすることで、実行を CloudWatch Logs または Step Functions コンソールで検査できます。 コンソールでの Express ワークフロー実行のデバッグの詳細については、「コンソール

	標準ワークフロー	エクスプレスワークフロー:同期および非同期
	行」と「実行の表示とデバッグ」を参照してください。	での Standard ワークフローと Express ワークフローの実行」と「実行の表示とデバッグ」を参照してください。
実行セマンティクス	一度だけのワークフロー実行。	非同期 Express ワークフロー:ワークフローの実行が最低 1 回行われます。 同期 Express ワークフロー:最大 1 回のワークフロー実行。
サービス統合	すべてのサービス統合とパターンをサポートします。	すべてのサービス統合をサポートします。 **Note** Express ワークフローはジョブ実行 (`.sync`) やコールバック (`.waitForTaskToken`) サービス統合パターンをサポートしていません。
Step Functions アクティビティ	Step Functions アクティビティをサポートします。	Step Functions アクティビティをサポートしません。

同期および非同期 Express ワークフロー

選択できる Express ワークフローには、非同期 Express ワークフローと同期 Express ワークフローの 2 種類があります。

- 非同期 Express ワークフローは、ワークフローが開始されたことの確認を返しますが、ワークフローの完了を待機しません。結果を得るには、サービスの [CloudWatch Logs] ポーリングを行う必要があります。非同期 Express ワークフローは、メッセージングサービスや、他のサービスが依存しないデータ処理など、即時のレスポンス出力が必要ではない場合に使用できます。非同期 Express ワークフローは、イベントに応えて、Step Functions 内のネストされたワークフローまたは `StartExecution` API コールを使用して開始できます。
- 同期 Express ワークフローはワークフローを開始して、それが完了するまで待機してから、結果を返します。同期 Express ワークフローはマイクロサービスのオーケストレーションに使用できます。同期 Express ワークフローを使用すると、エラー処理、再試行、並列タスクの実行のための追加のコードを開発しなくても、アプリケーションを開発できます。同期 Express ワークフローは Amazon API Gateway、AWS Lambda、または `StartSyncExecution` API コールを使って呼び出すことができます。

> **Note**
>
> Step Functions Express ワークフローをコンソールから同期的に実行した場合、`StartSyncExecution` リクエストは 60 秒後に経過します。Express ワークフローを最長 5 分間同期的に実行するには、Step Functions コンソールの代わりに AWS SDK または AWS Command Line Interface (AWS CLI) を使用して `StartSyncExecution` リクエストを行います。

同期 Express 実行 API コールは、既存のアカウント容量制限には影響しません。Step Functions は、オンデマンドで容量を提供し、持続的なワークロードを使って、自動的に拡張します。容量が利用可能になるまで、ワークロードの急増をスロットリングできます。

実行の保証

標準ワークフロー	非同期 Express ワークフロー	同期 Express ワークフロー	
一度だけのワークフロー実行	最低 1 回のワークフロー実行	最大 1 回のワークフロー実行	

標準ワークフロー	非同期 Express ワークフロー	同期 Express ワークフロー	
実行状態は状態遷移の間も内部的に持続します。	実行状態は状態遷移の間は持続しません。	実行状態は状態遷移の間は持続しません。	
現在実行中のワークフローと同じ名前で実行を開始すると、自動的にべき等性の応答が返されます。新しいワークフローは開始せず、現在実行中のワークフローが完了すると例外がスローされます。	べき等性は自動的には管理されません。同じ名前で複数のワークフローを開始すると、同時に実行されます。ステートマシンのロジックがべき等でないと、内部ワークフローの状態が失われる可能性があります。	べき等性は自動的には管理されません。Step Functions は実行が開始されると待機し、完了時にステートマシンの結果を返します。例外が発生しても、ワークフローは再開されません。	

標準ワークフロー	非同期 Express ワークフロー	同期 Express ワークフロー	
実行履歴データは 90 日後に削除されました。ワークフロー名は古い実行データを削除した後も再利用できます。 コンプライアンス、組織、または規制の要件を満たすために、クォータリクエストを送信することによって実行履歴の保持期間を 30 日に短縮できます。そのためには、AWS Support Center Console を使用して新しいケースを作成してください。	実行履歴は Step Functions ではキャプチャされません。ログ記録は Amazon CloudWatch Logs を通じて有効にする必要があります。	実行履歴は Step Functions ではキャプチャされません。ログ記録は Amazon CloudWatch Logs を通じて有効にする必要があります。	

Express ワークフローによるコスト最適化

Step Functions は、ステートマシンの構築に使用するワークフロータイプに基づいて、標準ワークフローと Express ワークフローの価格を決定します。サーバーレスワークフローのコストを最適化するには、以下の推奨事項のどちらかまたは両方に従ってください。

トピック

- ヒント #1: 標準ワークフローでの Express ワークフローのネスト
- ヒント #2: Express ワークフローへの標準ワークフローの変換

Standard または Express ワークフロータイプの選択が請求にどのように影響するかについては、「AWS Step Functions の料金」を参照してください。

ヒント #1: 標準ワークフローでの Express ワークフローのネスト

Step Functions は一定の期間とステップ数のあるワークフローを実行します。ワークフローによっては、短時間待機した後に実行を完了する場合があります。また、長時間実行されるワークフローとイベントレートの高いワークフローの両方を組み合わせなければならない場合があります。Step Functions を使用すると、小さくて単純な複数のワークフローから大規模で複雑なワークフローを構築できます。

例えば、注文処理ワークフローを構築する場合、同一ではないすべてのアクションを標準ワークフローに含めることができます。これには、人が操作する注文の承認や支払いの処理などのアクションが含まれる場合があります。その後、支払い通知の送信や商品在庫の更新など、一連の独立したアクションを Express ワークフローにまとめることができます。この Express ワークフローは標準ワークフロー内にネストできます。この例で、標準ワークフローは親ステートマシンと呼ばれます。ネストされた Express ワークフローは子ステートマシンと呼ばれます。

ヒント #2: Express ワークフローへの標準ワークフローの変換

既存の標準ワークフローが以下の要件を満たしていれば、Express ワークフローに変換できます。

- ワークフローは 5 分以内に実行を完了する必要があります。
- このワークフローは、少なくとも 1 回の実行モデルに準拠しています。つまり、ワークフローの各ステップは 1 回以上実行される可能性があるということです。
- ワークフローは `.waitForTaskToken` または `.sync` のサービス統合パターンを使用しません。

> **Important**
>
> Express ワークフローは Amazon CloudWatch Logs を使用して実行履歴を記録します。CloudWatch Logs を使用すると、追加のコストが発生します。

コンソールを使用して標準ワークフローを Express ワークフローに変換するには

1. Step Functions コンソールを開きます。
2. [ステートマシン] ページで、標準タイプのステートマシンを選択して開きます。

> **Tip**
>
> [すべてのタイプ] ドロップダウンリストから [標準] を選択すると、ステートマシンのリストがフィルタリングされ、標準ワークフローのみが表示されます。

3. [新規にコピー] を選択します。

   デザインモード で Workflow Studio が開き、選択したステートマシンのワークフローが表示されます。

4. (オプション) ワークフローのデザインを更新します。

5. ステートマシンの名前を指定します。これを行うには、MyStateMachine のデフォルトステートマシン名の横にある編集アイコンを選択します。次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

6. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   [タイプ] では必ず [Express] を選択してください。[ステートマシンの設定] のその他のデフォルト設定をすべてそのまま使用します。

> **Note**
>
> 以前 AWS CDK または AWS SAM で定義された標準ワークフローを変換する場合は、`Type` の値と `Resource` の名前を変更する必要があります。

7. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ワークフローのコスト最適化を管理する際のベストプラクティスとガイドラインについて詳しくは、「Building cost-effective AWS Step Functions workflows」を参照してください。

状態

個別の状態では、入力に基づいて決定を行い、それらの入力からアクションを実行して、出力を他の状態に渡すことができます。AWS Step Functions で、Amazon States Language (ASL) のワークフローを定義します。Step Functions コンソールは、アプリケーションロジックを視覚化するために、そのステートマシンをグラフィカルに表示します。

> **Note**
>
> Step Functions のコンソールの外部 (任意のエディタなど) でステートマシンを定義する場合は、ステートマシン定義を拡張子.asl.json を付けて保存する必要があります。

状態はステートマシンの要素です。状態は名前で参照されます。任意の文字列を指定できますが、ステートマシン全体の範囲で一意である必要があります。

状態はステートマシン内でさまざまな関数を実行できます。

- ステートマシンで何らかの作業をする (Task 状態)。
- 実行の選択肢間で選択する (選択状態)
- 失敗または成功で実行を停止する (失敗または成功状態)
- 入力を出力に渡す、または一部の固定データをワークフローに挿入する (パス状態)
- 一定時間または指定された日時まで遅延を設ける (待機状態)
- 実行の並列ブランチを開始する (並行状態)
- ステップを動的に反復する (マップ状態)

以下は、AWS Lambda 関数を実行する `HelloWorld` という名前の状態の例です。

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
  "Next": "AfterHelloWorldState",
  "Comment": "Run the HelloWorld Lambda function"
```

```
}
```

状態は多くの一般的な特徴を共有しています。

- その状態のタイプを示す Type フィールドがあります。
- 状態に関する、人間が読んで理解できるコメントまたは説明を保持するオプションの Comment フィールド。
- 各状態 (Succeed または Fail 状態を除く) には Next フィールドが必要です。または、代わりに End フィールドを指定して終了状態にできます。

> **Note**
>
> Choice 状態には 1 つ以上の Next がある場合がありますが、各 選択ルール内には 1 つのみです。Choice 状態では End を使用できません。

特定の状態タイプには追加フィールドが必要です。または共通フィールドの使用方法を再定義します。

標準ワークフローが作成されて実行されると、Step Functions コンソールの [実行の詳細] ページを表示することにより、各状態、入出力、いつ、どのくらいの時間アクティブだったかの情報にアクセスできます。詳細については「Step Functions コンソールでの実行の表示とデバッグ」を参照してください。

Express ワークフローを作成して実行した後、ログ記録が有効になっている場合は、Step Functions コンソール、または Amazon CloudWatch Logs にある、実行に関する情報にアクセスできます。詳細については「Step Functions コンソールでの実行の表示とデバッグ」を参照してください。

トピック

- Amazon ステートメント言語
- パス
- タスク
- 選択
- 待機
- 成功
- 失敗
- 並行

- マッピング

Amazon ステートメント言語

Amazon ステートメント言語は JSON ベースの構造化言語で、ステートマシンおよび作業を実行できる状態のコレクション (Task 状態) の定義、次に移行する状態の決定 (Choice 状態)、エラーによる実行の停止 (Fail 状態) などに使用されます。

詳細については、Amazon ステートメント言語の仕様と、Amazon ステートメント言語コードを検証するツールである Statelint を参照してください。

Amazon ステートメント言語を使って Step Functions コンソール上でステートマシンを作成するには、開始方法を参照してください。

> **Note**
>
> Step Functions のコンソールの外部 (任意のエディタなど) でステートマシンを定義する場合は、ステートマシン定義を拡張子.asl.json を付けて保存する必要があります。

Amazon ステートメント言語の仕様の例

```
{
  "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
    }
  }
}
```

トピック

- ステートマシン構造
- 組み込み関数
- [Common State] (共通状態) フィールド

ステートマシン構造

ステートマシンは、次のフィールドを含む構造を表す JSON テキストを使用して定義されます。

Comment (オプション)

ステートマシンの人間が読み取れる説明。

StartAt (必須)

いずれかの状態オブジェクトの名前と完全に一致する必要がある (大文字と小文字が区別されます) 文字列。

TimeoutSeconds (オプション)

ステートマシンを実行できる最大秒数。指定された時間より長く実行されると、States.Timeout エラー名 で実行が失敗します。

Version (オプション)

ステートマシンで使用される のバージョンです (デフォルトは「1.0」)。

States (必須)

オブジェクトには、コンマで区切られた一連の状態が含まれています。

States フィールドには状態が含まれています。

```
{
    "State1" : {
    },

    "State2" : {
    },
    ...
}
```

ステートマシンは、それに含まれている状態と、状態間の関係によって定義されます。

次に例を示します。

```
{
  "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
```

```
    }
  }
}
```

このステートマシンの実行が開始されると、システムは StartAt フィールドで参照されている状態で開始されます ("HelloWorld")。この状態に "End": true フィールドがある場合、実行が停止して結果が返されます。それ以外の場合、システムは "Next": フィールドを探し、次はその状態で続行します。このプロセスは、システムが終了状態 ("Type": "Succeed"、"Type": "Fail"、または "End": true の状態) に到達するまで繰り返されます。そうでない場合は、ランタイムエラーが発生します。

ステートマシン内の状態には次のルールが適用されます。

- 状態は、囲みブロック内の任意の順序で発生しますが、リストされた順序は実行順序に影響を与えません。状態自体の内容によってこの順序が決まります。
- ステートマシン内では、最上位の構造で StartAt フィールドの値により指定される start 状態として指定された状態が 1 つだけ存在します。この状態は、実行が開始されたときに最初に実行される状態です。
- End フィールドが true の状態は、end (または terminal) 状態とみなされます。ステートマシンのロジックによっては (ステートマシンに実行のブランチが複数あるかどうかなど)、複数の end 状態が存在する可能性があります。
- ステートマシンが 1 つの状態だけで構成される場合、start 状態と end 状態の両方になることができます。

組み込み関数

Amazon States Language には、Task 状態を使用せずに基本的なデータ処理操作を実行するのに役立つ、組み込み関数 がいくつか用意されています。組み込み関数はプログラミング言語の関数に似たコンストラクトです。これらを使用すると、ペイロードビルダーが Task 状態の Resource フィールドに対して送受信されるデータを処理するために使用できます。

Amazon States Language では、実行するデータ処理タスクの種類に基づいて、組み込み関数は次のカテゴリに分類されます。

- 配列の組み込み関数
- データのエンコードとデコードのための組み込み関数
- ハッシュ計算用組み込み関数

- JSON データ操作用の組み込み関数
- 数学演算用組み込み関数
- String 演算の組み込み関数
- 一意の ID を生成するための組み込み関数
- 汎用操作用の組み込み関数

> **Note**
>
> - 組み込み関数を使用するには、次の例に示すとおり、ステートマシン定義のキー値に　.$ を指定する必要があります。
>
> ```
> "KeyId.$": "States.Array($.Id)"
> ```
>
> - ワークフローの 1 つのフィールドには、最大 10 個の組み込み関数をネストできます。次の例は、9 つのネストされた組み込み関数を含む *myArn* という名前のフィールドを示しています。
>
> ```
> "myArn.$": "States.Format('{}.{}.{}',
>  States.ArrayGetItem(States.StringSplit(States.ArrayGetItem(States.StringSplit($.ImageRe
>  '/'), 2), '.'), 0),
>  States.ArrayGetItem(States.StringSplit(States.ArrayGetItem(States.StringSplit($.ImageRe
>  '/'), 2), '.'), 1))"
> ```

> **Tip**
>
> ローカル開発環境で Step Functions を使用する場合は、ワークフローにすべての組み込み関数を含めることができるように、バージョン 1.12.0 以降を使用していることを確認してください。

組み込み関数をサポートするフィールド

次の表は、どのフィールドが各状態で組み込み関数をサポートするかを示します。

組み込み関数をサポートするフィールド

State

	パス	タスク	選択	待機	成功	失敗	並行	マップ
InputPath								
パラメータ	✓	✓					✓	✓
ResultSelector		✓					✓	✓
ResultPath								
OutputPath								
変数								
<ComparisonOperator(比較演算子)>Path								
TimeoutSecondsPath								
HeartbeatsecondsPath								
認証情報		✓						

配列の組み込み関数

配列操作には以下の組み込み関数を使用します。

States.Array

States.Array 組み込み関数は 0 以上の引数を取ります。インタープリタは、指定された順序で引数の値を含む JSON 配列を返します。例として、次の入力があるとします。

```
{
  "Id": 123456
}
```

以下を使用できます

```
"BuildId.$": "States.Array($.Id)"
```

これに対して が返されます。

```
“BuildId”: [123456]
```

States.ArrayPartition

States.ArrayPartition 組み込み関数を使用して大きな配列を分割します。この組み込み関数を使用しデータをスライスしてから、ペイロードを小さなチャンクに分けて送信することもできます。

この組み込み関数は 2 つの引数を取ります。1 番目の引数は配列で、2 番目の引数はチャンクサイズを定義します。インタープリタは入力配列をチャンクサイズで指定されたサイズの複数の配列にチャンクします。配列に残っている項目の数がチャンクサイズよりも小さい場合、最後の配列チャンクの長さは前の配列チャンクの長さよりも短くなることがあります。

入力の検証

- 関数の 1 番目の引数の入力値として配列を指定する必要があります。
- チャンクサイズ値を表す 2 番目の引数には、0 以外の正の整数を指定する必要があります。

  2 番目の引数に整数以外の値を指定すると、Step Functions はその値を最も近い整数に四捨五入します。

- 入力配列は Step Functions のペイロードサイズ制限である 256 KB を超えることはできません。

例えば、次の入力配列があるとします。

```
{"inputArray": [1,2,3,4,5,6,7,8,9] }
```

States.ArrayPartition 関数を使用して、配列を次の 4 つの値のチャンクに分割できます。

```
"inputArray.$": "States.ArrayPartition($.inputArray,4)"
```

この場合、次のような配列チャンクが返されます。

```
{"inputArray": [ [1,2,3,4], [5,6,7,8], [9]] }
```

前の例では、States.ArrayPartition 関数は 3 つの配列を出力します。最初の 2 つの配列にはそれぞれ、チャンクサイズで定義される 4 つの値が含まれます。3 番目の配列には残りの値が含まれ、定義されたチャンクサイズよりも小さくなっています。

States.ArrayContains

States.ArrayContains 組み込み関数を使用して、配列に特定の値が存在するかどうかを判断します。例えば、この関数を使用して Map 状態反復でエラーが発生したかどうかを検出できます。

この組み込み関数は 2 つの引数を取ります。1 番目の引数は配列で、2 番目の引数は配列内で検索される値です。

入力の検証

- 関数の 1 番目の引数の入力値として配列を指定する必要があります。
- 2 番目の引数には有効な JSON オブジェクトを指定する必要があります。
- 入力配列は Step Functions のペイロードサイズ制限である 256 KB を超えることはできません。

例えば、次の入力配列があるとします。

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9],
```

```
   "lookingFor": 5
}
```

States.ArrayContains 関数を使用して、inputArray 内の lookingFor 値を検索できます。

```
"contains.$": "States.ArrayContains($.inputArray, $.lookingFor)"
```

lookingFor に格納されている値は inputArray に含まれているため、States.ArrayContains は次の結果を返します。

```
{"contains": true }
```

States.ArrayRange

States.ArrayRange 組み込み関数を使用して、特定の範囲の要素を含む新しい配列を作成します。新しい配列には最大 1000 個の要素を含めることができます。

この関数は 3 つの引数を取ります。1 番目の引数は新しい配列の 1 番目の要素、2 番目の引数は新しい配列の最後の要素、3 番目の引数は新しい配列の要素間の増分値です。

入力の検証

- すべての引数に整数値を指定する必要があります。

  いずれかの引数に整数以外の値を指定すると、Step Functions はその値を最も近い整数に四捨五入します。
- 3 つ目の引数には 0 以外の値を指定する必要があります。
- 新しく生成された配列には 1000 個を超える項目を含めることはできません。

例えば、States.ArrayRange 関数を次のように使用すると、最初の値が 1、最後の値が 9 の配列が作成され、最初の値と最後の値の間の値は、項目ごとに 2 ずつ増加します。

```
"array.$": "States.ArrayRange(1, 9, 2)"
```

この場合、次のような配列が返されます。

```
{"array": [1,3,5,7,9] }
```

States.ArrayGetItem

この組み込み関数は、指定されたインデックスの値を返します。この関数は 2 つの引数を取ります。1 番目の引数は値の配列で、2 番目の引数は返される値の配列インデックスです。

例えば、次の inputArray 値および index 値を使用します。

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9],
   "index": 5
}
```

これらの値から、States.ArrayGetItem 関数を使用して配列内の 5 番目の位置の index の値を返すことができます。

```
"item.$": "States.ArrayGetItem($.inputArray, $.index)"
```

この例では、States.ArrayGetItem は次の結果を返します。

```
{ "item": 6 }
```

States.ArrayLength

States.ArrayLength 組み込み関数は配列の長さを返します。1 つの引数を持ち、それは長さを返す配列です。

例えば、次の入力配列があるとします。

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9]
}
```

States.ArrayLength を使用して以下の inputArray の長さを返すことができます。

```
"length.$": "States.ArrayLength($.inputArray)"
```

この例では、States.ArrayLength は配列の長さを表す次の JSON オブジェクトを返します。

```
{ "length": 9 }
```

States.ArrayUnique

States.ArrayUnique 組み込み関数は配列から重複する値を削除し、一意の要素のみを含む配列を返します。この関数は 1 つの配列を取りますが、これは唯一の引数であるためソートされません。

例えば、次の inputArray には重複する一連の値が含まれています。

```
{"inputArray": [1,2,3,3,3,3,3,3,4] }
```

States.ArrayUnique 関数を使用して、重複する値を削除する配列を指定できます。

```
"array.$": "States.ArrayUnique($.inputArray)"
```

States.ArrayUnique 関数は、重複する値をすべて削除して、一位の要素のみを含む次の配列を返します。

```
{"array": [1,2,3,4] }
```

データのエンコードとデコードのための組み込み関数

以下の組み込み関数を使用して、Base64 エンコーディングスキームに基づいてデータをエンコードまたはデコードします。

States.Base64Encode

States.Base64Encode 組み込み関数を使用して、MIME Base64 エンコーディングスキームに基づいてデータをエンコードします。この関数を使うと、AWS Lambda 関数を使わずにデータを他の AWS サービスに渡すことができます。

この関数は、最大 10,000 文字のデータ文字列を取り、唯一の引数としてエンコードします。

例えば、次の input 文字列を考えてみます。

```
{"input": "Data to encode" }
```

States.Base64Encode 関数を使用して、input 文字列を MIME Base64 文字列としてエンコードできます。

```
"base64.$": "States.Base64Encode($.input)"
```

States.Base64Encode 関数は応答として以下のエンコード済みデータを返します。

```
{"base64": "RGF0YSB0byBlbmNvZGU=" }
```

States.Base64Decode

States.Base64Decode 組み込み関数を使用して、MIME Base64 デコーディングスキームに基づいてデータをデコードします。この関数を使うと、Lambda 関数を使わずにデータを他の AWS サービスに渡すことができます。

この関数は、最大 10,000 文字の Base64 エンコード済みデータ文字列を取り、唯一の引数としてデコードします。

例として、次の入力があるとします。

```
{"base64": "RGF0YSB0byBlbmNvZGU=" }
```

States.Base64Decode 関数を使用して base64 文字列を人間が読める文字列にデコードできます。

```
"data.$": "States.Base64Decode($.base64)"
```

States.Base64Decode function は応答として次のデコード済みデータを返します。

```
{"data": "Decoded data" }
```

ハッシュ計算用組み込み関数

States.Hash

States.Hash 組み込み関数を使用して、特定の入力のハッシュ値を計算します。この関数を使うと、Lambda 関数を使わずにデータを他の AWS サービスに渡すことができます。

この関数は 2 つの引数を取ります。最初の引数は、ハッシュ値を計算するデータです。2 番目の引数は、ハッシュ計算を行うために使用するハッシュアルゴリズムです。指定するデータは 10,000 文字以下のオブジェクト文字列である必要があります。

指定するハッシュアルゴリズムは、以下のいずれかのアルゴリズムにすることができます。

- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512

例えば、この関数を使用して、指定した Algorithm を使用する Data 文字列のハッシュ値を計算できます。

```
{
   "Data": "input data",
   "Algorithm": "SHA-1"
}
```

States.Hash 関数を使用してハッシュ値を計算できます。

```
"output.$": "States.Hash($.Data, $.Algorithm)"
```

States.Hash 関数は応答として以下のハッシュ値を返します。

```
{"output": "aaff4a450a104cd177d28d18d7485e8cae074b7" }
```

JSON データ操作用の組み込み関数

これらの関数を使用して JSON オブジェクトに対して基本的なデータ処理操作を実行します。

States.JsonMerge

States.JsonMerge 組み込み関数を使用して 2 つの JSON オブジェクトを単一オブジェクトにマージします。この関数は 3 つの引数を取ります。最初の 2 つの引数はマージする JSON オブジェクトです。3 番目の引数は false のブール値です。このブール値により、ディープマージモードが有効かどうか決定されます。

現在、Step Functions はシャローマージモードのみをサポートしているため、ブーリアン値を false として指定する必要があります。シャローモードでは、両方の JSON オブジェクトに同じキーが存在する場合、後者のオブジェクトのキーが最初のオブジェクトの同じキーよりも優先されます。さらに、JSON オブジェクト内にネストされたオブジェクトは、シャローマージを使用してもマージされません。

例えば、States.JsonMerge 関数を使用して、キー a を共有する次の JSON オブジェクトをマージできます。

```
{
   "json1": { "a": {"a1": 1, "a2": 2}, "b": 2 },
   "json2": { "a": {"a3": 1, "a4": 2}, "c": 3 }
}
```

json1 オブジェクトと json2 オブジェクトを States.JsonMerge 関数の入力として指定して、それらを結合できます。

```
"output.$": "States.JsonMerge($.json1, $.json2, false)"
```

States.JsonMerge は結果として以下のマージされた JSON オブジェクトを返します。マージされた JSON オブジェクト output では、json2 オブジェクトのキー a により json1 オブジェクトキー a が置き換えられます。また、シャローモードはネストされたオブジェクトのマージをサポートしていないため、json1 オブジェクトのキー a 内のネストされたオブジェクトは破棄されます。

```
{
   "output": {
      "a": {"a3": 1, "a4": 2},
      "b": 2,
      "c": 3
   }
}
```

States.StringToJson

States.StringToJson 関数は、エスケープ処理された JSON 文字列への参照パスを唯一の引数として取ります。

インタープリタは JSON パーサを適用して、入力のパース済み JSON 形式を返します。例えば、この関数を使用して次の入力文字列をエスケープ処理できます。

```
{
 "escapedJsonString": "{\"foo\": \"bar\"}"
}
```

States.StringToJson 関数を使用して、escapedJsonString を入力引数として指定します。

```
States.StringToJson($.escapedJsonString)
```

States.StringToJson 関数は、次の結果を返します。

```
{ "foo": "bar" }
```

States.JsonToString

States.JsonToString 関数は引数を 1 つのみ取ります。それはエスケープ処理されていない文字列として返される JSON データを含むパスです。インタープリタは、パスで指定されたデータを表す JSON テキストを含む文字列を返します。例えば、エスケープ処理された値を含む次の JSON パスを指定できます。

```
{
  "unescapedJson": {
     "foo": "bar"
  }
}
```

unescapedJson 内に含まれるデータを States.JsonToString 関数に提供します。

```
States.JsonToString($.unescapedJson)
```

States.JsonToString 関数は以下の応答を返します。

```
{\"foo\": \"bar\"}
```

数学演算用組み込み関数

これらの関数を使用して数学演算を実行します。

States.MathRandom

States.MathRandom 組み込み関数を使用して、指定した開始番号 (含む) と終了番号 (含まない) の間の乱数を返します。

この関数を使うと、特定のタスクを 2 つ以上のリソースに配布することができます。

この関数は 3 つの引数を取ります。1 番目の引数は開始番号、2 番目の引数は終了番号、最後の引数はシード値を制御します。シード値の引数はオプションです。この関数を同じシード値で使用すると、同じ数が返されます。

> **Important**
>
> States.MathRandom 関数は暗号的に安全な乱数を返さないため、高いセキュリティが要求されるアプリケーションには使用しないことをお勧めします。

入力の検証

- 開始番号と終了番号の引数には整数値を指定する必要があります。

  開始番号または終了番号の引数に整数以外の値を指定すると、Step Functions はその値を最も近い整数に四捨五入します。

例えば、1 から 999 までの乱数を生成するには、次の入力値を使用できます。

```
{
   "start": 1,
   "end": 999
}
```

乱数を生成するには、States.MathRandom 関数に start と end の値を指定します。

```
"random.$": "States.MathRandom($.start, $.end)"
```

States.MathRandom 関数は応答として次の乱数を返します。

```
{"random": 456 }
```

States.MathAdd

States.MathAdd 組み込み関数を使用して、2 つの数の加算結果を返します。例えば、この関数を使用すると、Lambda 関数を呼び出さずにループ内の値をインクリメントできます。

入力の検証

- すべての引数に整数値を指定する必要があります。

  いずれかまたは両方の引数に整数以外の値を指定すると、Step Functions はその値を最も近い整数に四捨五入します。
- -2147483648 ~ 2147483647 までの範囲の整数値を指定する必要があります。

例えば、次の値を使用して 111 から 1 を減算することができます。

```
{
   "value1": 111,
   "step": -1
}
```

次に、States.MathAdd 関数を使用して value1 を初期値、value1 の増分値を step に定義します。

```
"value1.$": "States.MathAdd($.value1, $.step)"
```

States.MathAdd 関数は応答として次の数を返します。

```
{"value1": 110 }
```

String 演算の組み込み関数

States.StringSplit

States.StringSplit 組み込み関数を使用して、文字列を値の配列に分割します。この関数は 2 つの引数を取ります。1 番目の引数は文字列で、2 番目の引数は関数が文字列を分割するために使用する区切り文字です。

Example - 単一文字列を使用して入力文字列を分割

この例では、States.StringSplit を使用して、カンマで区切られた一連の値を含む、以下の inputString を分割します。

```
{
```

```
    "inputString": "1,2,3,4,5",
    "splitter": ","
}
```

States.StringSplit を使用して、inputString を最初の引数、区切り文字 splitter を 2 番目の引数として定義します。

```
"array.$": "States.StringSplit($.inputString, $.splitter)"
```

States.StringSplit 関数は結果として次の文字列配列を返します。

```
{"array":  ["1","2","3","4","5"] }
```

Example - 複数の区切り文字を使用して入力文字列を分割

この例では、States.StringSplit を使用して、複数の区切り文字を含む以下の inputString を分割します。

```
{
  "inputString": "This.is+a,test=string",
  "splitter": ".+,="
}
```

States.StringSplit 関数を次のように使用します。

```
{
  "myStringArray.$": "States.StringSplit($.inputString, $.splitter)"
}
```

States.StringSplit 関数は結果として次の文字列配列を返します。

```
{"myStringArray": [
  "This",
  "is",
  "a",
  "test",
  "string"
]}
```

一意の ID を生成するための組み込み関数

States.UUID

States.UUID 組み込み関数を使用して、乱数を使用して生成されたバージョン 4 汎用一意識別子 (v4 UUID) を返します。例えば、この関数を使用して、UUID パラメータを必要とする他の AWS サービスやリソースを呼び出したり、DynamoDB テーブルに項目を挿入したりできます。

States.UUID 関数は引数を指定せずに呼び出されます。

```
"uuid.$": "States.UUID()"
```

この関数は、次の例のようにランダムに生成された UUID を返します。

```
{"uuid": "ca4c1140-dcc1-40cd-ad05-7b4aa23df4a8" }
```

汎用操作用の組み込み関数

States.Format

States.Format 組み込み関数を使用して、リテラル値と補間値の両方から文字列を作成します。この関数は 1 つ以上の引数を取ります。最初の引数の値は文字列である必要があり、文字シーケンス {} のインスタンスを 0 個以上含めることができます。組み込み関数の呼び出しには、{} の出現回数と同じだけの引数が残っていなければなりません。インタープリタは、組み込み関数の呼び出しの際に各 {} を位置対応引数の値に置き換えて、最初の引数で定義された文字列を返します。

例えば、次のような個人の name の入力と、その人の名前を挿入する template 文を使用できます。

```
{
 "name": "Arnav",
 "template": "Hello, my name is {}."
}
```

States.Format 関数を使用して、template 文字列と、{} 文字の代わりに挿入する文字列を指定します。

```
States.Format('Hello, my name is {}.', $.name)
```

または

```
States.Format($.template, $.name)
```

前述のいずれかを入力すると、States.Format 関数は応答として完成した文字列を返します。

```
Hello, my name is Arnav.
```

組み込み関数の予約文字

次の文字は組み込み関数用に予約されており、値に表示したい場合は、バックスラッシュ ('\') を使ってエスケープする必要があります。'{}、および\。

文字 \ をエスケープ文字として機能させずに値の一部として表示する必要があれば、最初にバックスラッシュをエスケープ文字にする必要があります。組み込み関数では、以下のエスケープ文字シーケンスが使用されます。

- リテラル文字列 \' は ' を表します。
- リテラル文字列 \{ は { を表します。
- リテラル文字列 \} は } を表します。
- リテラル文字列 \\ は \ を表します。

JSON では、文字列リテラル値に含まれるバックスラッシュを別のバックスラッシュでエスケープする必要があります。JSON の同等のリストは次のとおりです。

- エスケープされた文字列 \\\' は \' を表します。
- エスケープされた文字列 \\\{ は \{ を表します。
- エスケープされた文字列 \\\} は \} を表します。
- エスケープされた文字列 \\\\ は \\ を表します。

> **Note**
>
> オープンエスケープバックスラッシュ \ が組み込み関数の呼び出し文字列で見つかると、インタープリタはランタイムエラーを返します。

[Common State] (共通状態) フィールド

Type (必須)

状態のタイプ。

Next

現在の状態が終了するときに実行される次の状態の名前。Choice など、一部の状態では複数の移行状態が許可されます。

現在の状態がワークフローの最後の状態、成功 または 失敗 またはなどのターミナルステートである場合は、Next フィールドを指定する必要はありません。

End

true に設定されている場合は、この状態を終了状態として指定します (実行が終了されます)。ステートマシンごとに、任意の数の終了状態が存在します。状態では Next または End のどちらか 1 つのみを使用できます。Choice などの一部の状態タイプや、成功 や 失敗 などのターミナルの状態は、End フィールドをサポートしていないか、または使用していません。

Comment (オプション)

状態に関する、人間が読み取れる説明を保持します。

InputPath (オプション)

状態の入力の一部を選択して状態の処理タスクに渡すパス。省略した場合、入力全体を指定する値 $ が設定されます。詳細については、入力および出力処理を参照してください。

OutputPath (オプション)

状態の出力の一部を選択して次の状態に渡すパス。省略した場合、出力全体を指定する値 $ が設定されます。詳細については、入力および出力処理を参照してください。

パス

Pass 状態 ("Type": "Pass") は、何も作業せずに入力を出力に渡します。Pass 状態は、ステートマシンを構築およびデバッグする際に便利です。

Pass 状態を使用して、フィルターを使用する JSON 状態入力を変換し、変換されたデータをワークフローの次の状態に渡すこともできます。入力変換の詳細については、「InputPath、パラメータ、ResultSelector」を参照してください。

[common state fields] (共通状態フィールド) に加えて、Pass 状態では次のフィールドを使用できます。

Result (オプション)

次の状態に渡される仮想タスクの出力を参照します。ResultPath フィールドをステートマシン定義に含めると、Result は ResultPath で指定されたとおりに配置され、次の状態に渡されます。

ResultPath (オプション)

Result で指定された仮想タスクの出力を配置する場所 (入力に対して) を指定します。その後、入力は OutputPath フィールド (ある場合) に従ってフィルタリングされてから状態の出力に使用されます。詳細については、入力および出力処理を参照してください。

Parameters (オプション)

入力として渡されるキーと値のペアの集合を作成します。Parameters を静的な値として指定することも、パスを使用して入力から選択することもできます。詳細については「InputPath、パラメータ、 ResultSelector」を参照してください。

パス状態の例

以下の Pass 状態の例では、テストなどの目的で、ステートマシンにいくつかの固定データを挿入します。

```
"No-op": {
  "Type": "Pass",
  "Result": {
    "x-datum": 0.381018,
    "y-datum": 622.2269926397355
  },
  "ResultPath": "$.coords",
  "End": true
}
```

この状態の入力は以下であるとします。

```
{
  "georefOf": "Home"
}
```

出力は次のようになります。

```
{
  "georefOf": "Home",
  "coords": {
    "x-datum": 0.381018,
    "y-datum": 622.2269926397355
  }
}
```

タスク

Task 状態 ("Type": "Task") は、ステートマシンによって実行される単一の作業単位を表します。タスクは、アクティビティや AWS Lambda 関数を使用して、サポート対象 AWS のサービス と統合して、または Stripe のようなサードパーティーの API を呼び出すことによって、作業を実行します。

Amazon States Language は、ステートのタイプを Task に設定し、タスクにアクティビティ、Lambda 関数、またはサードパーティー API エンドポイントの Amazon リソースネーム (ARN) を指定することによりタスクを表します。次のタスク状態定義は、*HelloFunction* という名前の Lambda 関数を呼び出します。

```
"Lambda Invoke": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Parameters": {
    "Payload.$": "$",
    "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:HelloFunction:
$LATEST"
  },
  "End": true
}
```

このトピックの内容

- タスクタイプ
- タスク状態フィールド
- タスク状態定義の例
- アクティビティ

タスクタイプ

Step Functions では、Task 状態定義内で指定できる次のタスクタイプをサポートしています。

- アクティビティ
- Lambda 関数
- サポート対象の AWS のサービス
- HTTP タスク

タスクタイプを指定するには、タスク状態定義の `Resource` フィールドに ARN を指定します。次の例は、`Resource` フィールドの構文を示しています。サードパーティー API を呼び出すタスクタイプを除くすべてのタスクタイプは、次の構文を使用します。HTTP 構文の詳細については、「サードパーティーの API を呼び出す」を参照してください。

タスク状態定義で、次の構文のイタリック体のテキストを、AWS リソース固有の情報に置き換えます。

```
arn:partition:service:region:account:task_type:name
```

次のリストでは、この構文の個々のコンポーネントについて説明します。

- `partition` は使用する AWS Step Functions のパーティションです。最も一般的に使用されるのは `aws` です。
- `service` はタスクの実行に使用した AWS のサービス を示しており、次のいずれかの値を取ることができます。
  - `states`アクティビティに。
  - Lambda 関数 (`lambda`) を実行します。Amazon SNS または Amazon DynamoDB などの他の AWS のサービス と統合する場合は、`sns` または `dynamodb` を使用してください。
- `region` は、Step Functions アクティビティ、ステートマシンタイプ、Lambda 関数、または他の AWS リソースが作成された AWS リージョンコードです。
- `account` はリソースを定義した AWS アカウント ID です。
- `task_type` は実行するタスクのタイプです。これには、次のいずれかの値を指定できます。
  - `activity` – アクティビティ。
  - `function` - Lambda 関数。
  - *`servicename`* - サポートされた接続サービスの名前 (Step Functions 用統合最適化 を参照)。

- `name` はメンバーリソース名 (アクティビティ名、 Lambda 関数名、またはサービス API アクション) です。

> **Note**
>
> Step Functions は、パーティションまたはリージョン間の ARN の参照をサポートしていません。例えば、`aws-cn` は `aws` パーティション内のタスクを呼び出すことはできません。その逆も同様です。

次のセクションでは、各タスクタイプの詳細を示します。

アクティビティ

アクティビティは、お客様によって実装およびホストされ、特定のタスクを実行するワーカー (プロセスまたはスレッド) を示します。標準ワークフローでのみサポートされ、Express ワークフローではサポートされません。

アクティビティ `Resource` ARN は以下の構文を使用します。

```
arn:partition:states:region:account:activity:name
```

> **Note**
>
> アクティビティは、最初に使用する前に Step Functions (CreateActivity、API アクション、または Step Functions コンソールを使用) で作成しておく必要があります。

アクティビティの作成とワーカーの実装の詳細については、アクティビティを参照してください。

Lambda 関数

Lambda タスクでは AWS Lambda を使用して関数を実行します。Lambda 関数を指定するには、`Resource` フィールドの Lambda 関数の ARN を使用します。

Lambda 関数の指定に使用する統合のタイプ (最適化された統合または AWS SDK 統合) によって、Lambda 関数の `Resource` フィールドの構文は異なります。

次の `Resource` フィールド構文は、Lambda 関数と最適化された統合の例です。

```
"arn:aws:states:::lambda:invoke"
```

次の Resource フィールド構文は、Lambda 関数との AWS SDK 統合の例です。

```
"arn:aws:states:::aws-sdk:lambda:invoke"
```

次の Task 状態定義は、*HelloWorld* という名前の Lambda 関数と最適化された統合の例を示しています。

```
"LambdaState": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "OutputPath": "$.Payload",
  "Parameters": {
    "Payload.$": "$",
    "FunctionName": "arn:aws:lambda:us-east-1:function:HelloWorld:$LATEST"
  },
  "Next": "NextState"
}
```

Resource フィールドに指定された Lambda 関数が完了すると、その出力が Next フィールド (「NextState」) で指定された状態に送信されます。

サポート対象の AWS のサービス

接続されたリソースをリファレンスすると、Step Functions によって、サポートされたサービスの API アクションが直接呼び出されます。Resource フィールドでサービスとアクションを指定します。

接続されたサービス Resource ARN では以下の構文を使用します。

```
arn:partition:states:region:account:servicename:APIname
```

> **Note**
>
> 接続されたリソースへの同期接続を作成するには、.sync を ARN の *APIname* エントリに付加します。詳細については、「他の サービスでの使用」を参照してください。

例:

```
{
 "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
    }
  }
}
```

タスク状態フィールド

[common state fields] (共通状態フィールド) に加えて、Task 状態には次のフィールドがあります。

Resource (必須)

URI (特に、実行する特定のタスクを一意に識別する ARN)。

Parameters (オプション)

接続されたリソースの API アクションに情報を渡すのに使用します。パラメータには静的な JSON と JsonPath を組み合わせて使用できます。詳細については、「サービス API にパラメータを渡す」を参照してください。

Credentials (オプション)

指定した Resource を呼び出す前にステートマシンの実行ロールが継承する必要があるターゲットロールを指定します。または、実行入力に基づいて実行時に IAM ロール ARN を解決する JSONPath 値または組み込み関数を指定することもできます。JSONPath 値を指定する場合は、$. 表記をプレフィックスとして付ける必要があります。

このフィールドを Task 状態で使用した例については、「タスク状態の認証情報フィールドの例」を参照してください。このフィールドを使用してステートマシンからクロスアカウント

AWS リソースにアクセスする例については、「チュートリアル: AWS クロスアカウントリソースへのアクセス」を参照してください。

> **Note**
>
> このフィールドは、Lambda 関数およびサポート対象 AWS サービスを使用する タスクタイプ によってサポートされています。

`ResultPath` (オプション)

`Resource` で指定されたタスクを実行した結果を配置する場所 (入力内) を指定します。その後、入力は `OutputPath` フィールド (ある場合) に従ってフィルタリングされてから状態の出力に使用されます。詳細については、入力および出力処理を参照してください。

`ResultSelector` (オプション)

値が静的であるか、結果から選択されたキーバリューのペアの集合を渡します。詳細については、「ResultSelector」を参照してください。

`Retry` (オプション)

Retrier と呼ばれるオブジェクトの配列。状態でランタイムエラーが発生した場合の再試行ポリシーを定義します。詳細については、「Retry と Catch を使用するステートマシンの例」を参照してください。

`Catch` (オプション)

Catcher と呼ばれるオブジェクトの配列で、フォールバック状態を定義します。この状態は、状態にランタイムエラーが発生し、その再試行ポリシーが使い果たされたか定義されていない場合に実行されます。詳細については、フォールバック状態を参照してください。

`TimeoutSeconds` (オプション)

アクティビティまたはタスクが States.Timeout エラーでタイムアウトして失敗するまでに実行できる最大時間を指定します。タイムアウトの値はゼロ以外の正の整数にする必要があります。デフォルト値は 99999999 です。

タスクが開始されるとタイムアウトのカウントが開始します (例: `ActivityStarted` または `LambdaFunctionStarted` イベントが実行イベント履歴に口グインされる際)。アクティビティにおいては、`GetActivityTask` がトークンを受信し、`ActivityStarted` が実行イベント履歴に口グインされると、カウントが開始します。

タスクが開始されると、Step Functions は指定された TimeoutSeconds 期間内にタスクまたはアクティビティワーカーからの成功または失敗の応答を待ちます。タスクまたはアクティビティのワーカーがこの時間内の応答に失敗した場合、Step Functions はワークフロー実行を失敗としてマークします。

TimeoutSecondsPath (オプション)

リファレンスパスを使用して状態の入力からタイムアウト値を動的に指定したい場合は、TimeoutSecondsPath を使用してください。解決されると、リファレンスパスは、値が正の整数のフィールドを選択する必要があります。

> **Note**
>
> Task 状態には TimeoutSeconds と TimeoutSecondsPath の両方を含めることはできません。

HeartbeatSeconds (オプション)

タスクの実行中にアクティビティワーカーが送信するハートビートシグナルの頻度を決定します。ハートビートは、タスクがまだ実行中で、完了するまでにさらに時間がかかることを示します。ハートビートは、アクティビティやタスクが TimeoutSeconds 期間内にタイムアウトするのを防ぎます。

HeartbeatSeconds は TimeoutSeconds フィールド値より小さい 0 以外の正の整数値でなければなりません。デフォルト値は 99999999 です。タスクからのハートビートの間隔が指定された秒数よりも長くなる場合、タスク状態は States.Timeout エラーで失敗します。

アクティビティにおいては、GetActivityTask がトークンを受信し、ActivityStarted が実行イベント履歴にログインされると、カウントが開始します。

HeartbeatSecondsPath (オプション)

リファレンスパスを使用して状態の入力からハートビート値を動的に指定したい場合は、HeartbeatSecondsPath を使用します。解決されると、リファレンスパスは、値が正の整数のフィールドを選択する必要があります。

> **Note**
>
> Task 状態には HeartbeatSeconds と HeartbeatSecondsPath の両方を含めることはできません。

Task 状態は、その状態で実行が終了する場合は End フィールドが true に設定されている必要があります。または、Next フィールドに、Task 状態が完了した際に実行される状態を指定する必要があります。

タスク状態定義の例

次の例は、要件に基づいてタスク状態定義を指定する方法を示しています。

- タスク状態のタイムアウトとハートビート間隔
  - 静的タイムアウトとハートビート通知の例
  - 動的タスクタイムアウトとハートビート通知の例
- 認証情報フィールドの使用
  - ハードコーディングされた IAM ロール ARN の指定
  - IAM ロール ARN として JSONPath を指定
  - 組み込み関数を IAM ロール ARN として指定する

タスク状態のタイムアウトとハートビート間隔

長時間実行されるアクティビティでは、タイムアウト値とハートビート間隔を設定することをお勧めします。これは、タイムアウトとハートビート値を指定するか、動的に設定することによって実行できます。

静的タイムアウトとハートビート通知の例

HelloWorld が完了すると、次の状態 (ここでは NextState と呼ばれます) が実行されます。

このタスクが 300 秒以内に完了しなかった場合、または 60 秒間隔のハートビート通知を送信しなかった場合、タスクは failed とマークされます。

```
"ActivityState": {
  "Type": "Task",
  "Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld",
  "TimeoutSeconds": 300,
  "HeartbeatSeconds": 60,
  "Next": "NextState"
}
```

動的タスクタイムアウトとハートビート通知の例

この例では、AWS Glue ジョブが完了すると、次の状態が実行されます。

このタスクが、AWS Glue ジョブによって動的に設定された間隔内に完了しなかった場合、タスクは `failed` としてマークされます。

```
"GlueJobTask": {
  "Type": "Task",
  "Resource": "arn:aws:states:::glue:startJobRun.sync",
  "Parameters": {
    "JobName": "myGlueJob"
  },
  "TimeoutSecondsPath": "$.params.maxTime",

  "Next": "NextState"
}
```

タスク状態の認証情報フィールドの例

ハードコーディングされた IAM ロール ARN の指定

次の例では、ステートマシンの実行ロールが Echo という名前のクロスアカウント Lambda 関数にアクセスするために継承する必要がある、ターゲット IAM ロールを指定します。この例では、ターゲットロールの ARN はハードコードされた値として指定されています。

```
{
  "StartAt": "Cross-account call",
  "States": {
    "Cross-account call": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn": "arn:aws:iam::111122223333:role/LambdaRole"
      },
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:Echo"
      },
      "End": true
    }
  }
}
```

IAM ロール ARN として JSONPath を指定

次の例では、実行時に IAM ロール ARN を解決する JSONPath 値を指定します。

```
{
  "StartAt": "Lambda",
  "States": {
    "Lambda": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn.$": "$.roleArn"
      },
      ...
    }
  }
}
```

組み込み関数を IAM ロール ARN として指定する

次の例では、実行時に IAM ロール ARN を解決する、`States.Format` 組み込み関数を使用しています。

```
{
  "StartAt": "Lambda",
  "States": {
    "Lambda": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn.$": "States.Format('arn:aws:iam::{}:role/ROLENAME', $.accountId)"
      },
      ...
    }
  }
}
```

アクティビティ

アクティビティは作業がワーカーによって実行されるステートマシン内のタスク設定を有効にする AWS Step Functions 機能です。このタスクは、Amazon Elastic Compute Cloud (Amazon EC2)、Amazon Elastic Container Service (Amazon ECS)、モバイルデバイスと、基本的にはどこでも、ホストできます。

概要

AWS Step Functions では、アクティビティはどこかで実行されているコード (アクティビティワーカーとも呼ばれる) を、ステートマシンの特定のタスクと関連付ける方法です。Step Functions コンソールを使用して、または CreateActivity を呼び出すことでアクティビティを作成できます。これによって、タスクの状態の Amazon リソースネーム (ARN) が提供されます。この ARN を使用して、アクティビティワーカーでの作業のタスク状態をポーリングします。

> **Note**
>
> アクティビティはバージョニングされず、下位互換性があることが予期されます。アクティビティ定義に下位互換性のない変更を加える必要がある場合は、一意の名前を使用して、Step Functions で新しいアクティビティを作成する必要があります。

アクティビティワーカーは、Amazon EC2 インスタンス、AWS Lambda 関数、モバイルデバイスで実行されているアプリケーションです。任意の場所にホストされている、HTTP 接続を行うことができる任意のアプリケーションです。Step Functions がアクティビティタスクの状態に達すると、ワークフローはアクティビティワーカーのタスクのポーリングを待機します。アクティビティワーカーは、GetActivityTask を使用して Step Functions をポーリングし、関連するアクティビティの ARN を送信します。GetActivityTask は input (タスクの JSON 入力文字列) および taskToken (タスクの一意の識別子) を含むレスポンスを返します。アクティビティワーカーは作業の完了後に SendTaskSuccess または SendTaskFailure を使用して成功または失敗のレポートを提供します。これら 2 つの呼び出しは、GetActivityTask によって提供される taskToken を使用して結果をタスクと関連付けます。

アクティビティタスクに関連する API

Step Functions では、アクティビティの作成と一覧表示、タスクのリクエスト、ワーカーの結果に基づくステートマシンのフロー管理のための API を提供しています。

以下は、アクティビティに関連する Step Functions API です。

- CreateActivity
- GetActivityTask
- ListActivities
- SendTaskFailure
- SendTaskHeartbeat

- SendTaskSuccess

> **Note**
>
> GetActivityTask を使用してアクティビティタスクをポーリングすると、一部の実装でレイテンシーが発生する可能性があります。アクティビティタスクのポーリング時のレイテンシーを回避する を参照してください。

アクティビティタスクの完了を待機中

タスク定義で TimeoutSeconds を設定して、ステートが待機する時間を設定します。タスクをアクティブおよび待機中にしておくためには、TimeoutSeconds で設定した時間内に SendTaskHeartbeat を使用して、アクティビティワーカーから定期的にハートビートを送信します。長いタイムアウト期間を設定し、能動的にハートビートを送信することで、Step Functions のアクティビティは実行の完了を最大で 1 年間待機できます。

例えば、長いプロセスの結果を待機するワークフローが必要な場合は、以下の操作を実行します。

1. コンソールを使用して、または CreateActivity を呼び出してアクティビティを作成します。アクティビティ ARN を書き留めます。
2. ステートマシンの定義でアクティビティのタスク状態の ARN を参照し、TimeoutSeconds を設定します。
3. GetActivityTask を使用して作業をポーリングするアクティビティワーカーを実装し、そのアクティビティ ARN を参照します。
4. ステートマシンのタスク定義の HeartbeatSeconds で設定した時間内に定期的に SendTaskHeartbeat を使用して、タスクがタイムアウトしないようにします。
5. ステートマシンの実行を開始します。
6. アクティビティワーカーのプロセスを開始します。

実行はアクティビティタスクの状態で一時停止して、アクティビティワーカーのタスクのポーリングを待機します。taskToken がアクティビティワーカーに提供されると、ワークフローはステータス提供のため SendTaskSuccess または SendTaskFailure を待機します。TimeoutSeconds で設定された時間より前に、これらのいずれかまたは SendTaskHeartbeat 呼び出しを実行が受け取らない場合、実行は失敗し、実行履歴に ExecutionTimedOut イベントが含まれることになります。

次のステップ

アクティビティワーカーを使用するステートマシンの作成の詳細については、以下を参照してください。

- Step Functions を使用してアクティビティステートマシンを作成する
- Ruby のサンプルアクティビティワーカー

Ruby のサンプルアクティビティワーカー

以下は、ベストプラクティスの使用方法およびお客様独自のアクティビティワーカーを実装する方法を示すため、AWS SDK for Ruby を使用するアクティビティワーカーの例です。

このコードには、ポーラーおよびアクティビティワーカー用に構成可能な数のスレッドを含む、コンシューマー/プロデューサーパターンが実装されています。ポーラースレッドは、アクティビティタスクを常にロングポーリングします。アクティビティタスクが取得されると、アクティビティスレッドの区切られたブロックキューを通じて渡され、選択されます。

- AWS SDK for Ruby の詳細については、AWS SDK for Ruby API リファレンスを参照してください。
- このコードと関連リソースをダウンロードするには、GitHub リポジトリで step-functions-ruby-activity-worker を参照してください。

次の Ruby コードは、この例の Ruby アクティビティワーカーのメインエントリポイントです。

```
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'] }
end
```

コードにはデフォルトが含まれていますが、アクティビティを参照するために変更し、実際の実装に合わせることができます。このコードは、実際の実装ロジックを入力として取得するため、特定のアクティビティと認証情報を参照し、スレッドの数とハートビートの遅延を設定できます。詳細およびコードのダウンロードについては、Step Functions Ruby アクティビティワーカーを参照してください。

項目	説明
`require_relative`	次の例のアクティビティワーカーコードへの相対パス。
`region`	アクティビティの AWS リージョン。
`workers_count`	アクティビティワーカーのスレッドの数。ほとんどの実装では、10 ~ 20 のスレッドで十分です。アクティビティの処理に時間がかかればかかるほど、必要なスレッドが多くなります。試算するには、1 秒あたりのプロセスアクティビティの数と、99 パーセンタイルアクティビティ処理レイテンシー (秒単位) を乗算してください。
`pollers_count`	ポーラーのスレッドの数。ほとんどの実装では、10 ~ 20 のスレッドで十分です。
`heartbeat_delay`	ハートビート間の遅延 (秒)。
`input`	アクティビティの実装ロジック。

以下は、コード内で `../lib/step_functions/activity` とともに参照される Ruby アクティビティワーカーです。

```
require 'set'
require 'json'
require 'thread'
require 'logger'
require 'aws-sdk'

module Validate
  def self.positive(value)
    raise ArgumentError, 'Argument has to be 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 has 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
```

選択

Choice 状態 ("Type": "Choice") はステートマシンに条件付きロジックを追加します。

共通状態フィールドの大半に加えて、Choice 状態には次の追加フィールドが含まれます。

Choices (必須)

ステートマシンが次に移行する状態を決定する選択ルールの配列。選択ルールで比較演算子を使用して、入力変数を特定の値と比較します。例えば、選択ルールを使用すると、入力変数が 100 より大きいか小さいかを比較できます。

Choice 状態が実行されると、各選択ルールは true または false と評価されます。この評価の結果に基づいて、Step Functions はワークフローの次の状態に移行します。

Choice 状態では、少なくとも 1 つのルールを定義する必要があります。

Default (オプション、推奨)

Choices のいずれの移行も実行されない場合の移行先の状態の名前。

> **Important**
>
> Choice 状態では End フィールドはサポートされません。また、Next は Choices フィールド内でのみ使用されます。

> **Tip**
>
> Choice ステートを使用するワークフローの例を AWS アカウント にデプロイするには、AWS Step Functions ワークショップの「モジュール 5- Choice ステートおよびマップステート」を参照してください。

選択ルール

Choice 状態には値が空でない配列の Choices フィールドが必要です。この配列の各要素は選択ルールというオブジェクトで、以下の内容が含まれています。

- 比較 – 比較する入力変数、比較のタイプ、および可変を比較する値に指定する 2 つのフィールド。選択ルールは、2 つの可変の比較をサポートします。選択ルール内で、可変値は、サポートされている比較演算子の名前に Path を加えて、状態入力の別の値と比較できます。比較対象の Variable フィールドとパスフィールドの値は、有効な参照パスでなければなりません。
- **Next** フィールド – このフィールドの値はステートマシンの状態名と一致する必要があります。

次の例では、数値が 1 と等しいかどうかを確認します。

```
{
  "Variable": "$.foo",
  "NumericEquals": 1,
  "Next": "FirstMatchState"
}
```

次の例では、文字列が MyString と等しいかどうかを確認します。

```
{
  "Variable": "$.foo",
  "StringEquals": "MyString",
  "Next": "FirstMatchState"
}
```

次の例では、文字列が MyStringABC を超過しているかどうかを確認します。

```
{
  "Variable": "$.foo",
  "StringGreaterThan": "MyStringABC",
```

```
  "Next": "FirstMatchState"
}
```

次の例では、文字列が null かどうかを確認します。

```
{
 "Variable": "$.possiblyNullValue",
 "IsNull": true
}
```

次の例は、IsPresent 選択ルール が先行しているため、$.keyThatMightNotExist が 存在する場合は、StringEquals ルールのみが評価される方法を示しています。

```
"And": [
 {
 "Variable": "$.keyThatMightNotExist",
 "IsPresent": true
 },
 {
 "Variable": "$.keyThatMightNotExist",
 "StringEquals": "foo"
 }
]
```

次の例では、ワイルドカードを含むパターンが一致するかどうかを確認します。

```
{
 "Variable": "$.foo",
 "StringMatches": "log-*.txt"
}
```

次の例では、タイムスタンプが 2001-01-01T12:00:00Z と等しいかどうかを確認します。

```
{
  "Variable": "$.foo",
  "TimestampEquals": "2001-01-01T12:00:00Z",
  "Next": "FirstMatchState"
}
```

次の例では、可変を状態の入力の別の値と比較します。

```
{
 "Variable": "$.foo",
 "StringEqualsPath": "$.bar"
}
```

Step Functions は Choices フィールドにリストされた順序で各選択ルールを検討します。次に、最初の選択ルールの Next フィールドで指定された状態に遷移します。ここで、変数は比較演算子に従って値と一致します。

次の比較演算子がサポートされています。

- And
- BooleanEquals,BooleanEqualsPath
- IsBoolean
- IsNull
- IsNumeric
- IsPresent
- IsString
- IsTimestamp
- Not
- NumericEquals,NumericEqualsPath
- NumericGreaterThan,NumericGreaterThanPath
- NumericGreaterThanEquals,NumericGreaterThanEqualsPath
- NumericLessThan,NumericLessThanPath
- NumericLessThanEquals,NumericLessThanEqualsPath
- Or
- StringEquals,StringEqualsPath
- StringGreaterThan,StringGreaterThanPath
- StringGreaterThanEquals,StringGreaterThanEqualsPath
- StringLessThan,StringLessThanPath
- StringLessThanEquals,StringLessThanEqualsPath
- StringMatches
- TimestampEquals,TimestampEqualsPath

- TimestampGreaterThan,TimestampGreaterThanPath
- TimestampGreaterThanEquals,TimestampGreaterThanEqualsPath
- TimestampLessThan,TimestampLessThanPath
- TimestampLessThanEquals,TimestampLessThanEqualsPath

これらの演算子のそれぞれで、対応する値が適切なタイプ (文字列、数値、ブール値、またはタイムスタンプ) である必要があります。Step Functions では、数値フィールドと文字列値の一致を試みません。ただし、タイムスタンプフィールドは論理的に文字列であるため、タイムスタンプとみなされるフィールドを StringEquals コンパレータで一致させることはできます。

> **Note**
>
> 相互運用性のため、数値比較は IEEE 754-2008binary64 データタイプで表される大きさまたは精度を外れる値では機能しないと考えてください。特に、$[-2^{53}+1, 2^{53}-1]$ の範囲外の整数では想定した形での比較が失敗する可能性があります。
> タイムスタンプ (例えば、2016-08-18T17:33:00Z) は、RFC3339 プロファイル ISO 8601 に準拠している必要があり、さらに制限があります。
>
> - 大文字の T で日付部分と時刻部分を区切る必要があります。
> - 大文字の Z で数値タイムゾーンオフセットが存在しないことを示す必要があります。
>
> 文字列比較の動作を理解するには、Java compareTo のドキュメントを参照してください。
> And および Or 演算子は選択ルールの空ではない配列の値であり、それ自体に Next フィールドを含まない必要があります。同様に、Not 演算子の値は単一の選択ルールである必要があり、Next フィールドを含めることはできません。
> And、Not、および Or を使用して、複雑なネスト化された選択ルールを作成できます。ただし、Next フィールドは最上位の選択ルールにのみ使用できます。
> 1 つ以上のワイルドカード (「*」) を使用したパターンに対する文字列比較は、StringMatches 比較演算子を使用して実行できます。ワイルドカード文字は、スタンダード \\ (Ex: “*”) を使用してエスケープされます。「*」以外の文字は、マッチング中に特別な意味を持ちません。

選択状態の例

以下は、Choice 状態および移行先の他の状態の例です。

> **Note**
>
> $.type フィールドを指定する必要があります。状態の入力に $.type フィールドが含まれていない場合は、実行が失敗し、実行履歴にエラーが表示されます。文字列をリテラル値と一致する StringEquals フィールドに指定することしかできません。例えば、"StringEquals": "Buy" です。

```
"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!"
}
```

この例では、以下の入力値を使用してステートマシンが開始されます。

```
{
  "type": "Private",
  "value": 22
}
```

Step Functions は、`value` フィールドに基づいて、`ValueInTwenties` 状態に移行します。

`Choice` 状態の `Choices` と一致するものがない場合、`Default` フィールドに指定されている状態が代わりに実行されます。`Default` 状態が指定されていない場合は、エラーで失敗します。

待機

`Wait` 状態 (`"Type": "Wait"`) は、ステートマシンの続行を指定された時間遅延させます。相対時間 (状態が開始してからの秒数で指定) と絶対時間 (タイムスタンプで指定) のいずれかを選択できます。

共通状態フィールドに加えて、`Wait` 状態には次のいずれかのフィールドがあります。

Seconds

Next フィールドで指定された状態の開始を待機する時間 (秒)。時間は 0 ~ 99999999 までの正の整数値として指定する必要があります。

Timestamp

Next フィールドで指定された状態の開始を待機する絶対時間。

タイムスタンプは、ISO 8601 の RFC3339 プロファイルに従う必要があります。ただし、大文字の T によって日付部分と時刻部分を区切り、大文字の Z によって数値のタイムゾーンオフセットが存在しないことを示す必要があるという追加の制限があります。例えば、2024-08-18T17:33:00Z などです。

> **Note**
>
> 現在、待機時間をタイムスタンプとして指定すると、Step Functions は時間値を最大秒と見なし、ミリ秒を切り捨てます。

SecondsPath

Next フィールドで指定された状態の開始まで待機する時間 (秒)。状態の入力データから取得されたパスを使用して指定します。

このフィールドには整数値を指定する必要があります。

TimestampPath

Next フィールドで指定された状態の開始まで待機する絶対時間。状態の入力データから取得されたパスを使用して指定します。

> **Note**
>
> Seconds、Timestamp、SecondsPath、または TimestampPath のいずれかちょうど 1 つを指定する必要があります。また、標準ワークフローとエクスプレスワークフローに指定できる最大待機時間は、それぞれ 1 年と 5 分です。

待機状態の例

例えば、次の Wait 状態はステートマシンに 10 秒の遅延を導入します。

```
"wait_ten_seconds": {
  "Type": "Wait",
  "Seconds": 10,
  "Next": "NextState"
}
```

次の例では、Wait ステートは絶対時間 2024 年 3 月 14 日午前 1:59 (UTC) まで待機します。

```
"wait_until" : {
  "Type": "Wait",
  "Timestamp": "2024-03-14T01:59:00Z",
  "Next": "NextState"
}
```

待機時間をハードコードする必要はありません。例えば、以下の入力データがあるとします。

```
{
  "expirydate": "2024-03-14T01:59:00Z"
}
```

リファレンスパス???を使用して入力データから選択することで、値「expirydate」を入力データから選択できます。

```
"wait_until" : {
    "Type": "Wait",
    "TimestampPath": "$.expirydate",
    "Next": "NextState"
}
```

成功

Succeed 状態 ("Type": "Succeed") は実行を正常に停止します。Succeed 状態は、実行を停止するだけの Choice 状態ブランチのターゲットに便利です。

Succeed 状態は終了状態であるため、次の例に示すように、End フィールドがなく、Next フィールドは必要ありません。

```
"SuccessState": {
  "Type": "Succeed"
}
```

失敗

Fail 状態 ("Type": "Fail") は、ステートマシンの実行を停止し、Catch ブロックでキャッチされるのでなければ、障害としてマークします。

Fail 状態では、Type 共通状態フィールド Comment のセットから および フィールドの使用のみが許可されます。また、Fail 状態では以下のフィールドを使用できます。

Cause (オプション)

エラーの原因を説明するカスタム文字列。このフィールドは、運用または診断目的で指定できます。

CausePath (オプション)

リファレンスパスを使用して、状態入力からエラーの原因に関する詳細な説明を動的に指定する場合は、CausePath を使用します。解決されると、リファレンスパスは文字列値を含むフィールドを選択する必要があります。

文字列を返す組み込み関数を使用して CausePath を指定することもできます。これらの組み込み関数
は、States.Format、States.JsonToString、States.ArrayGetItem、States.Base64Encode、States.Base64
です。

> **Important**
>
> - 失敗状態の定義では、CausePath または Cause のいずれかを指定できますが、両方を指定することはできません。
> - 情報セキュリティのベストプラクティスとして、原因の説明から機密情報や内部システムの詳細を削除することをお勧めします。

Error (オプション)

Retry または Catch フィールドを使用してエラー処理を実行する際に指定できるエラー名。運用または診断目的で使用できるエラー名も提供できます。

ErrorPath (オプション)

リファレンスパスを使用して状態入力からエラー名を動的に指定する場合は、ErrorPath を使用してください。解決されると、リファレンスパスは文字列値を含むフィールドを選択する必要があります。

文字列を返す組み込み関数を使用して ErrorPath を指定することもできます。これらの組み込み関数は、States.Format、States.JsonToString、States.ArrayGetItem、States.Base64Encode、States.Base64 です。

> **Important**
>
> - 失敗状態の定義では、ErrorPath または Error のいずれかを指定できますが、両方を指定することはできません。
> - 情報セキュリティのベストプラクティスとして、エラー名から機密情報や内部システムの詳細を削除することをお勧めします。

Fail 状態は常にステートマシンを終了するため、Next フィールドはなく、End フィールドも不要です。

失敗状態の例

以下の失敗状態の定義例では、Error および Cause フィールド値を指定しています。

```
"FailState": {
  "Type": "Fail",
  "Cause": "Invalid response.",
  "Error": "ErrorA"
}
```

以下の失敗状態の定義例では、参照パスを動的に使用して Error および Cause フィールド値を解決しています。

```
"FailState": {
  "Type": "Fail",
  "CausePath": "$.Cause",
  "ErrorPath": "$.Error"
```

```
}
```

以下の失敗状態の定義例では、States.Format 組み込み関数関数を使用して Error および Cause フィールド値を動的に指定しています。

```
"FailState": {
  "Type": "Fail",
  "CausePath": "States.Format('This is a custom error message for {}, caused by {}.',
 $.Error, $.Cause)",
  "ErrorPath": "States.Format('{}', $.Error)"
}
```

並行

Parallel 状態 ("Type": "Parallel") はステートマシンで実行する個別のブランチを追加するために使用できます。

[common state fields] (共通状態フィールド) に加えて、Parallel 状態には次の追加のフィールドがあります。

Branches (必須)

ステートマシンで並列して実行する状態を指定するオブジェクトの配列。このような各ステートマシンオブジェクトには States および StartAt というフィールドが必要です。これらの意味はステートマシンの最上位にあるものとまったく同様です。

ResultPath (オプション)

ブランチの出力を配置する (入力内の) 場所を指定します。その後、入力は OutputPath フィールド (ある場合) に従ってフィルタリングされてから状態の出力に使用されます。詳細については、入力および出力処理を参照してください。

ResultSelector (オプション)

値が静的であるか、結果から選択されたキーバリューのペアの集合を渡します。詳細については、「ResultSelector」を参照してください。

Retry (オプション)

Retrier と呼ばれるオブジェクトの配列。状態でランタイムエラーが発生した場合の再試行ポリシーを定義します。詳細については、「Retry と Catch を使用するステートマシンの例」を参照してください。

Catch (オプション)

Catcher と呼ばれるオブジェクトの配列。状態でランタイムエラーが発生し、再試行ポリシーがすでに試された後または定義されていない場合に実行されるフォールバック状態を定義します。詳細については、フォールバック状態を参照してください。

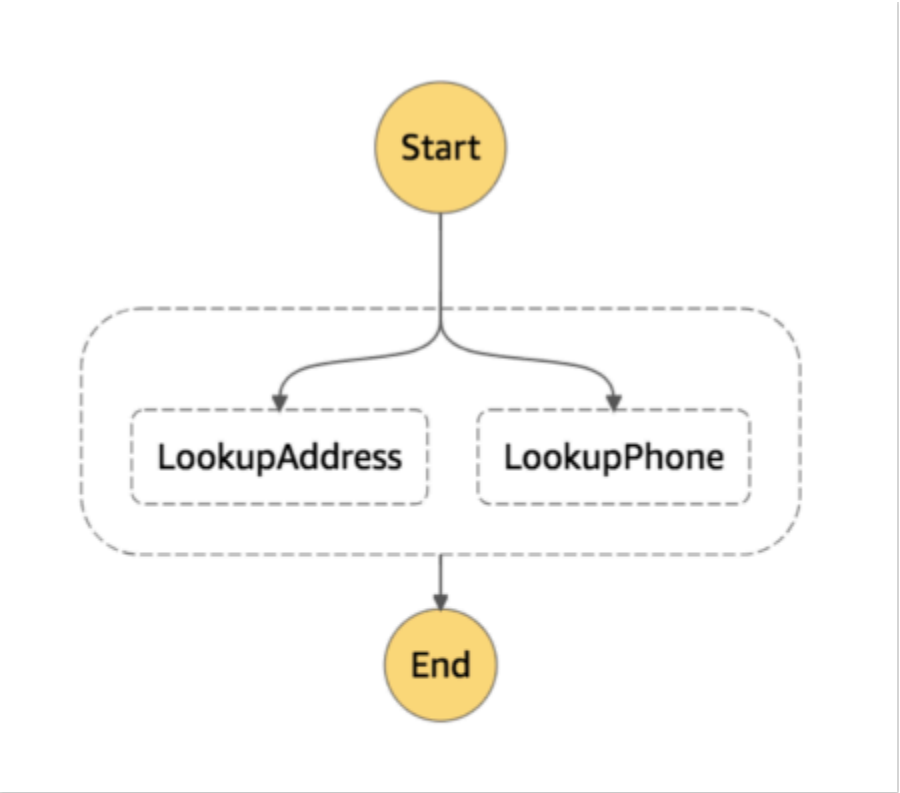
Parallel 状態により、 は各ブランチ AWS Step Functions を実行し、そのブランチの StartAtフィールドで指定された状態から、可能な限り同時に実行し、すべてのブランチが終了する (終了状態に到達する) まで待ってからParallel状態の Nextフィールドを処理します。

並行状態の例

```
{
  "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
           }
         }
       }
```

```
      ]
    }
  }
}
```

この例では、LookupAddress および LookupPhone ブランチは並行して実行されます。Step Functions コンソールでの視覚的なワークフローの外観を次に示します。



各ブランチは自己完結型である必要があります。Parallel 状態の 1 つのブランチの状態にはそのブランチ外のフィールドをターゲットにする Next フィールドがあってはなりません。また、そのブランチ外の他の状態からこのブランチに移行することもできません。

並行状態の入出力処理

Parallel 状態は各ブランチに自身の入力データ (InputPath フィールドによって変更される場合があります) のコピーを提供します。そのブランチからの出力を含む各ブランチの 1 つの要素から成る配列である出力が生成されます。すべての要素が同じタイプである必要はありません。出力配列は、通常の方法で ResultPath を使用して入力データに挿入し、全部を Parallel 状態の出力として送信できます (入力および出力処理を参照)。

```
{
  "Comment": "Parallel Example.",
  "StartAt": "FunWithMath",
  "States": {
    "FunWithMath": {
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Add",
          "States": {
            "Add": {
              "Type": "Task",
              "Resource": "arn:aws:states:us-east-1:123456789012:activity:Add",
              "End": true
            }
          }
        },
        {
          "StartAt": "Subtract",
          "States": {
            "Subtract": {
              "Type": "Task",
              "Resource": "arn:aws:states:us-east-1:123456789012:activity:Subtract",
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

FunWithMath 状態に入力として配列 [3, 2] が提供される場合、Add および Subtract 状態は両方ともその配列を入力として受信します。Add と Subtract タスクの出力は、配列要素 3 と 2 の和と差、つまり 5 と 1 になり、Parallel 状態の出力は配列になります。

```
[ 5, 1 ]
```

> **Tip**
>
> ステートマシンで使用している並列またはマップステートが配列の配列を返す場合は、ResultSelector フィールドを使用して配列をフラットな配列に変換できます。詳細については、「配列の配列の平坦化」を参照してください。

エラー処理

処理されないエラーまたは Fail 状態に移行したためにブランチが失敗すると、Parallel 状態全体が失敗とみなされ、そのすべてのブランチが停止します。エラーが Parallel 状態自体で処理されない場合、Step Functions は実行をエラーで停止します。

> **Note**
>
> 並列状態が失敗しても、呼び出された Lambda 関数が実行し続けられるため、タスクトークンを処理するアクティビティワーカーが停止することはありません。
>
> - 長時間稼働のアクティビティを停止するには、ハートビートを使用して、ブランチが Step Functions によって停止されたかどうかを検出し、タスクを処理しているワーカーを停止します。SendTaskHeartbeat、SendTaskSuccess、または SendTaskFailure を呼び出すと、状態が失敗した場合にエラーがスローされます。ハートビートエラーを参照してください。
> - Lambda 関数の実行は停止できません。フォールバックを実装している場合は、Lambda 関数終了後にクリーンアップが行われるように Wait 状態を使用します。

マッピング

Map 状態を使用して、データセット内の各アイテムに対して一連のワークフローステップを実行します。Map 状態の反復は並列で実行されるため、データセットを迅速に処理できます。Map 状態で

は、JSON 配列、Amazon S3 オブジェクトのリスト、CSV ファイルなど、さまざまな入力タイプを使用できます。

Step Functions には、ワークフローで Map 状態を使用するための処理モードとして、インラインモードと分散モードの 2 種類の処理モードがあります。

これらのモード、およびそのいずれかのモードで Map 状態を使用する方法の詳細については、次のトピックを参照してください。

- マップステート処理モード
  - インラインモードでのマップステートの使用
  - 分散マップ状態の使用

> **Tip**
>
> Map ステートを使用するワークフローの例を AWS アカウント にデプロイするには、AWS Step Functions ワークショップの「モジュール 5- Choice ステートおよびマップステート」を参照してください。

マップステート処理モード

Step Functions には、データセット内の項目の処理方法に応じて、Map 状態に対して次の処理モードが用意されています。

- インライン - 制限付き同時実行モード。このモードでは、Map 状態のそれぞれの反復は、Map 状態を含むワークフローのコンテキストで実行されます。Step Functions は、これらの反復の実行履歴を親ワークフローの実行履歴に追加します。デフォルトでは、Map 状態はインラインモードで実行されます。

  このモードでは、Map 状態は JSON 配列のみを入力として受け入れます。また、このモードでは最大 40 回の同時反復がサポートされています。

  詳細については「インラインモードでのマップステートの使用」を参照してください。
- 分散 - 高い並列性を持つモード。このモードでは、Map 状態は各反復を子ワークフロー実行として実行します。これにより、最大 10,000 件という高い並列性を持つの並列子ワークフロー実行が可能になります。それぞれの子ワークフローの実行には、親ワークフローとは別の実行履歴があります。

このモードでは、Map 状態は JSON 配列または CSV ファイルなどの Amazon S3 データソースを入力として受け入れることができます。

詳細については「分散マップ状態の使用」を参照してください。

使用するモードは、データセット内の項目の処理方法によって異なります。ワークフローの実行履歴が 25,000 エントリを超えない場合や、同時反復が 40 回以上必要ない場合は、Map 状態をインラインモードで使用します。

次の条件を満たす大規模な並列ワークロードをオーケストレーションする必要がある場合は、並列モードモードの Map 状態を使用します。

- データセットのサイズが 256 KB を超えています。
- ワークフローの実行イベント履歴が 25,000 エントリを超えています。
- 40 回以上の反復処理の同時実行が必要です。

トピック

- インラインモードと分散モードの違い
- インラインモードでのマップステートの使用
- マップステートを分散モードで使用して大規模な並行ワークロードをオーケストレーションする

インラインモードと分散モードの違い

次の表は、インラインモードと分散モードの違いを中心に示しています。

インラインモード	分散モード
Supported data sources	
ワークフローの前のステップから渡された JSON 配列を入力として受け取ります。	入力として次のデータソースを受け取ります。 • ワークフローの前のステップから渡された JSON 配列 • 配列が含まれる Amazon S3 バケット内の JSON ファイル

インラインモード	分散モード
	• Amazon S3 バケット内の CSV ファイル • Amazon S3 オブジェクトリスト • Amazon S3 インベントリ
Map iterations	
このモードでは、Map 状態のそれぞれの反復は、Map 状態を含むワークフローのコンテキストで実行されます。Step Functions は、これらの反復の実行履歴を親ワークフローの実行履歴に追加します。	このモードでは、Map 状態は各反復を子ワークフロー実行として実行します。これにより、最大 10,000 件という高い並列性を持つの並列子ワークフロー実行が可能になります。それぞれの子ワークフローの実行には、親ワークフローとは別の実行履歴があります。
Maximum concurrency for parallel iterations	
最大 40 回の反復を可能な限り同時に実行できます。	子ワークフローを最大 10,000 回並列で実行して、数百万件のデータ項目を一度に処理できます。
Input payload and event history sizes	
入力ペイロードサイズを 256 KB、実行イベント履歴に 25,000 エントリという制限を適用します。	Map 状態は、Amazon S3 データソースから直接入力を読み取ることができるため、ペイロードサイズの制限を解決できます。 このモードでは、Map 状態よって開始された子ワークフローの実行が、親ワークフローの実行履歴とは別の独自の実行履歴を保持するため、実行履歴の制限を解決することもできます。
Monitoring and observability	

インラインモード	分散モード
ワークフローの実行履歴は、コンソールから、または GetExecutionHistory API アクションを呼び出して確認できます。 また、CloudWatch と X-Ray を使用して実行履歴を表示できます。	分散モードで Map 状態を実行すると、Step Functions はマップ実行リソースを作成します。マップ実行とは、分散マップ状態によって開始する一連の子ワークフロー実行を指します。マップ実行は、Step Functions コンソールで表示できます。DescribeMapRun API アクションを呼び出すこともできます。また、マップ実行は CloudWatch にメトリクスを送信します。 詳細については「分散マップ状態実行のマップ実行の検証」を参照してください。

インラインモードでのマップステートの使用

デフォルトでは、Map 状態は [インライン] モードで実行されます。インラインモードでは、マップステートは JSON 配列のみを入力として受け入れます。ワークフローの前のステップから、この配列を受け取ります。このモードでは、Map 状態のそれぞれの反復は、Map 状態を含むワークフローのコンテキストで実行されます。Step Functions は、これらの反復の実行履歴を親ワークフローの実行履歴に追加します。

このモードでは、Map 状態は最大 40 回の同時反復をサポートします。

[インライン] に設定された Map ステートはインラインマップステートと呼ばれます。ワークフローの実行履歴が 25,000 エントリを超えない場合や、同時反復が 40 回以上必要ない場合は、Map 状態をインラインモードで使用します。

インラインマップステートの使用の概要については、チュートリアル インラインマップステートを使用したアクションの反復 を参照してください。

目次

- このトピックの主要概念
- インラインマップステートのフィールド
- 廃止されたフィールド
- インラインマップステートの例

- ItemSelector によるインラインマップステートの例
- インライン Map 状態の入出力処理

このトピックの主要概念

インラインモード

Map 状態の制限付き同時実行モード。このモードでは、Map 状態のそれぞれの反復は、Map 状態を含むワークフローのコンテキストで実行されます。Step Functions は、これらの反復の実行履歴を親ワークフローの実行履歴に追加します。Map 状態はデフォルトではインラインモードで実行されます。

このモードは JSON 配列のみを入力として受け入れ、最大 40 回の同時反復をサポートします。

インラインマップステート

[インライン] モードに設定された Map 状態。

マップワークフロー

Map各反復で状態が実行する一連のステップ。

マップステートの繰り返し

Map 状態内で定義されたワークフローの繰り返し。

インラインマップステートのフィールド

ワークフローでインラインマップステートを使用するには、これらのフィールドを 1 つ以上指定します。これらのフィールドは、共通状態フィールドに加えて指定します。

`Type` (必須)

状態のタイプ (Map など) を設定します。

`ItemProcessor` (必須)

Map 状態処理モードと定義を指定する次の JSON オブジェクトが含まれます。

定義には、各配列項目を処理する際に繰り返す一連の手順が含まれています。

- `ProcessorConfig` - Map 状態の処理モードを指定するオプションの JSON オブジェクト。このオブジェクトには `Mode` サブフィールドが含まれます。このフィールドのデフォルトは `INLINE` で、インラインモードの Map 状態を使用します。

このモードでは、反復処理が失敗すると Map 状態も失敗します。Map 状態に障害が発生すると、すべての反復処理が停止します。

- StartAt - ワークフローの最初の状態を示す文字列を指定します。この文字列は、大文字と小文字が区別され、いずれかの状態オブジェクトの名前と完全に一致する必要があります。この状態は、データセット内の各アイテムで最初に実行されます。Map 状態に提供した実行入力は、まず StartAt 状態に渡されます。
- States – カンマで区切られた一連の状態を含む JSON オブジェクト。このオブジェクトでは、Map workflowを定義します。

> **Note**
>
> - ItemProcessor フィールド内の状態は相互にしか遷移できません。フィールド外の状態は、ItemProcessor フィールド内の状態に遷移することはできません。
> - ItemProcessor フィールドは、現在廃止された Iterator フィールドに置き換わります。Iterator フィールドを使用する Map 状態は引き続き含めることができますが、このフィールドを ItemProcessor に置き換えることを強くお勧めします。
>
>   Step Functions Local は現在 ItemProcessor フィールドをサポートしていません。Iterator フィールドは、Step Functions Local で使用することをお勧めします。

ItemsPath (オプション)

JsonPath 構文を使用してリファレンスパスを指定します。このパスは、状態入力内の項目の配列を含む JSON ノードを選択します。詳細については「ItemsPath」を参照してください。

ItemSelector (オプション)

各 Map 状態反復に渡される前に、入力配列項目の値をオーバーライドします。

このフィールドでは、キーと値のペアのコレクションを含む有効な JSON を指定します。これらのペアには以下のいずれかを含めることができます。

- ステートマシンの定義で定義する静的値。
- パスを使用して状態入力から選択された値。
- コンテキストオブジェクトからアクセスされる値。

詳細については「ItemSelector」を参照してください。

ItemSelector フィールドは、現在廃止された Parameters フィールドに置き換わります。Parameters フィールドを使用する Map 状態は引き続き含めることができますが、このフィールドを ItemSelector に置き換えることを強くお勧めします。

MaxConcurrency (オプション)

並行して実行できる Map 状態の反復数の上限を示す整数値を指定します。例えば、MaxConcurrency 値が 10 の場合、Map 状態の反復の同時実行数は一度に 10 回に制限されます。

> **Note**
>
> 同時反復は制限される場合があります。この状況が起こったときは、前の反復が完了するまで一部の反復は開始されません。入力配列の項目が 40 を超えると、この問題が発生する可能性が高くなります。
> 同時実行性を高めるには、分散マップ状態の使用 を検討してください。

デフォルト値は 0 で、同時実行数に制限はありません。Step Functions は可能な限り同時に反復を呼び出します。

MaxConcurrency 値が 1 の場合、配列要素ごとに 1 回ずつ ItemProcessor を呼び出します。配列内の項目は、入力に現れる順に処理されます。Step Functions は、前の反復が完了するまで新しい反復を開始しません。

MaxConcurrencyPath (オプション)

リファレンスパスを使用して状態の入力からタイムアウト値を動的に最大同時実行値を指定する場合は、MaxConcurrencyPath を使用してください。解決されると、リファレンスパスは、値が負でない整数のフィールドを選択する必要があります。

> **Note**
>
> Map 状態には MaxConcurrency と MaxConcurrencyPath の両方を含めることはできません。

ResultPath (オプション)

Map 状態の反復の出力を入力のどこに保存するかを指定します。次に、マップステートは OutputPath フィールドの指定に従って入力をフィルタリングします (指定されている場合)。次

に、フィルター処理された入力を状態の出力として使用します。詳細については、入力および出力処理を参照してください。

`ResultSelector` (オプション)

値が静的であるか結果から選択された、キーバリューのペアの集合を渡します。詳細については「ResultSelector」を参照してください。

> **Tip**
>
> ステートマシンで使用している並列またはマップステートが配列の配列を返す場合は、ResultSelector フィールドを使用して配列をフラットな配列に変換できます。詳細については「配列の配列の平坦化」を参照してください。

`Retry` (オプション)

Retrier と呼ばれるオブジェクトの配列。再試行ポリシーを定義します。状態はランタイムエラーが発生すると再試行ポリシーを使用します。詳細については「Retry と Catch を使用するステートマシンの例」を参照してください。

> **Note**
>
> インライン状態に Retrier を定義すると、リトライポリシーは、失敗した反復だけではなくすべての `Map` 状態の反復に適用されます。例えば、`Map` 状態の反復が 2 回成功し、反復が 1 回失敗したとします。`Map` 状態の `Retry` フィールドを定義した場合、リトライポリシーは、失敗した反復だけでなく、3 回の `Map` 状態反復すべてに適用されます。

`Catch` (オプション)

Catcher と呼ばれるオブジェクトの配列で、フォールバック状態を定義します。状態は、ランタイムエラーが発生し、リトライポリシーが設定されていないか、リトライポリシーが使い果たされた場合に、catcher を実行します。詳細については、フォールバック状態を参照してください。

廃止されたフィールド

> **Note**
>
> 次のフィールドを使用する Map 状態は引き続き含めることができますが、この Iterator を ItemProcessor に Parameters を ItemSelector に置き換えることを強くお勧めします。

`Iterator`

配列の各要素を処理する一連のステップを定義する JSON オブジェクトを指定します。

`Parameters`

キーには次のいずれかを含めることができる、キーと値のペアの集合を指定します。

- ステートマシンの定義で定義する静的値。
- パスを使用して入力から選択された値。

インラインマップステートの例

[インライン] モードで実行されている Map 状態の次の入力データについて考えてみます。

```
{
  "ship-date": "2016-03-14T01:59:00Z",
  "detail": {
    "delivery-partner": "UQS",
    "shipped": [
      { "prod": "R31", "dest-code": 9511, "quantity": 1344 },
      { "prod": "S39", "dest-code": 9511, "quantity": 40 },
      { "prod": "R31", "dest-code": 9833, "quantity": 12 },
      { "prod": "R40", "dest-code": 9860, "quantity": 887 },
      { "prod": "R40", "dest-code": 9511, "quantity": 1220 }
    ]
  }
}
```

前の入力を指定すると、次の例の Map 状態は、shipped フィールドの配列の各項目に対して ship-val という名前の AWS Lambda 関数を 1 回呼び出します。

```
"Validate All": {
```

```
    "Type": "Map",
    "InputPath": "$.detail",
    "ItemProcessor": {
        "ProcessorConfig": {
            "Mode": "INLINE"
        },
        "StartAt": "Validate",
        "States": {
            "Validate": {
                "Type": "Task",
                "Resource": "arn:aws:states:::lambda:invoke",
                "OutputPath": "$.Payload",
                "Parameters": {
                    "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:ship-val:$LATEST"
                },
                "End": true
            }
        }
    },
    "End": true,
    "ResultPath": "$.detail.shipped",
    "ItemsPath": "$.shipped"
}
```

Map 状態の各反復は、ItemsPath フィールドで選択された、配列にある項目を ship-val Lambda 関数への入力として送信します。以下の値は、Lambda 関数の呼び出しに対して Map 状態が送信する入力の例です。

```
{
  "prod": "R31",
  "dest-code": 9511,
  "quantity": 1344
}
```

完了すると、Map 状態の出力は JSON 配列となります。配列の各項目は反復の出力です。この場合、この配列には ship-val Lambda 関数の出力が含まれます。

ItemSelector によるインラインマップステートの例

前の例の ship-val Lambda 関数でも、配送の運送会社に関する情報も必要であるとします。この情報は、繰り返しのたびに配列内の項目に追加されます。Map 状態の現在の反復に固有である情報

に加えて、入力からの情報を含めることもできます。次の例の ItemSelector フィールドに注目してください。

```
"Validate-All": {
  "Type": "Map",
  "InputPath": "$.detail",
  "ItemsPath": "$.shipped",
  "MaxConcurrency": 0,
  "ResultPath": "$.detail.shipped",
  "ItemSelector": {
    "parcel.$": "$$.Map.Item.Value",
    "courier.$": "$.delivery-partner"
  },
  "ItemProcessor": {
    "StartAt": "Validate",
    "States": {
      "Validate": {
        "Type": "Task",
 "Resource": "arn:aws:lambda:us-east-1:123456789012:function:ship-val",
        "End": true
      }
    }
  },
  "End": true
}
```

ItemSelector ブロックは反復への入力を JSON ノードに置き換えます。このノードには、コンテキストオブジェクトからの現在のアイテムデータと、Map 状態入力の delivery-partner フィールドからの運送会社に関する情報の両方が含まれます。以下に示すのは、1 回の反復処理への入力の例です。Map 状態は、この入力を ship-val Lambda 関数の呼び出しに渡します。

```
{
  "parcel": {
    "prod": "R31",
    "dest-code": 9511,
    "quantity": 1344
   },
   "courier": "UQS"
}
```

前述のインラインマップステートの例では、ResultPath フィールドは入力と同じ形式で出力を生成します。ただし、各要素が各反復の ship-val Lambda 呼び出しの出力である配列で detail.shipped フィールドが上書きされます。

インラインマップステートとそのフィールドの使用方法の詳細については、以下を参照してください。

- インラインマップステートを使用したアクションの反復
- Step Functions の入出力処理
- ItemsPath
- マップ状態のコンテキストオブジェクトデータ

インライン **Map** 状態の入出力処理

特定の Map 状態について、InputPath は状態の入力のサブセットを選択します。

Map 状態の入力には JSON 配列が含まれている必要があります。Map 状態は配列内の項目ごとに ItemProcessor セクションを 1 回実行します。ItemsPath フィールドを指定すると、Map 状態は反復処理の対象となる配列を検索する入力内の場所を選択します。場所を指定しない場合、ItemsPath の値は $ となり、ItemProcessor セクションはこの配列が唯一の入力であると見なします。ItemsPath フィールドを指定する場合、その値はリファレンスパスである必要があります。Map 状態は、InputPath を適用した後に、このパスを有効な入力に適用します。ItemsPath は、値が JSON 配列であるフィールドを識別する必要があります。

各反復への入力は、デフォルトでは、ItemsPath 値で識別される配列フィールドの単一の要素です。この値は、ItemSelector フィールドで上書きできます。

完了すると、Map 状態の出力は JSON 配列となります。配列の各項目は反復の出力です。

インラインマップステートの入力と出力の詳細については、以下を参照してください。

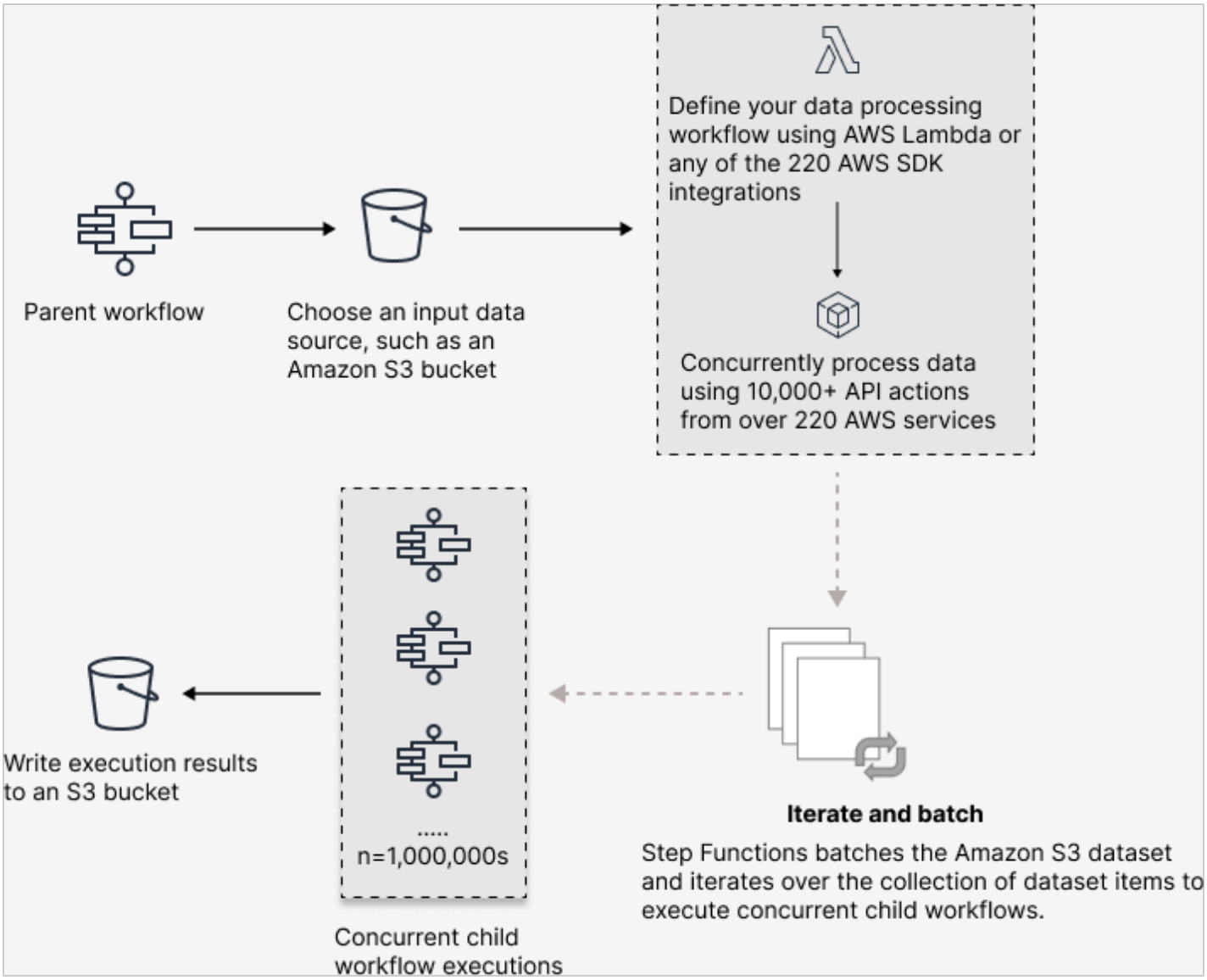
- インラインマップステートを使用したアクションの反復
- ItemSelector によるインラインマップステートの例
- Step Functions の入出力処理
- マップ状態のコンテキストオブジェクトデータ
- マップ状態を使用してデータを動的に処理する

マップステートを分散モードで使用して大規模な並行ワークロードをオーケストレーションする

Step Functions を使用すると、大規模な並行ワークロードをオーケストレーションして、半構造化データのオンデマンド処理などのタスクを実行できます。これらの並行ワークロードにより、Amazon S3 に保存されている大規模なデータソースを同時に処理できます。例えば、大量のデータを含む 1 つの JSON または CSV ファイルを処理できます。あるいは、大量の Amazon S3 オブジェクトを処理する場合もあります。

ワークフローに大規模な並行ワークロードを設定するには、Map 状態を分散モードに含めます。マップステートは、データセット内のアイテムを同時に処理します。[分散] に設定された Map ステートは、分散マップ状態と呼ばれます。分散モードでは、Map 状態によって高度な同時処理が可能になります。分散モードでは、Map 状態はデータセット内の項目を子ワークフロー実行と呼ばれる反復処理を行います。並行して実行できる子ワークフローの実行数を指定できます。それぞれの子ワークフローの実行には、親ワークフローとは別の実行履歴があります。指定しない場合、Step Functions は 10,000 件の子ワークフローを並列で実行します。

次の図は、ワークフローに大規模な並行ワークロードを設定する方法を示しています。



このトピックの内容

- 重要な用語
- 分散マップ状態の定義の例
- 分散マップを実行するアクセス許可
- 分散マップ状態のフィールド
- 次のステップ

重要な用語

分散モード

マップステートの処理モード。このモードでは、Map 状態の各反復処理が子ワークフロー実行として実行されるため、高い同時実行性が可能になります。子ワークフローの実行にはそれぞれ独自の実行履歴があり、親ワークフローの実行履歴とは別のものです。このモードは、大規模な Amazon S3 のデータソースからの入力の読み取りをサポートします。

分散マップ状態

[分散] 処理モードに設定されたマップステート。

マップワークフロー

Map 状態が実行する一連のステップ。

親ワークフロー

1 つ以上の分散マップ状態を含むワークフロー。

子ワークフロー実行

分散マップ状態の反復。子ワークフローの実行には独自の実行履歴があり、親ワークフローの実行履歴とは別のものです。

マップ実行

分散モードで Map 状態を実行すると、Step Functions はマップ実行リソースを作成します。マップ実行とは、分散マップ状態によって開始する一連の子ワークフロー実行、およびこれらの実行をコントロールするランタイム設定を指します。Step Functions は、マップの実行に Amazon リソースネーム (ARN) を割り当てます。マップ実行は、Step Functions コンソールで確認できます。`DescribeMapRun` API アクションを呼び出すこともできます。Map Run CloudWatch はメトリクスをにも送信します。

詳細については、「マップ実行の確認」を参照してください。

分散マップ状態の定義の例

次の条件を満たす大規模な並列ワークロードをオーケストレーションする必要がある場合は、並列モードモードの Map 状態を使用します。

- データセットのサイズが 256 KB を超えています。

- ワークフローの実行イベント履歴が 25,000 エントリを超えています。
- 40 回以上の反復処理の同時実行が必要です。

次の分散マップ状態の定義例では、データセットを Amazon S3 バケットに格納された CSV ファイルとして指定しています。また、CSV ファイルの各行のデータを処理する Lambda 関数も指定しています。この例では CSV ファイルを使用しているため、CSV 列ヘッダーの場所も指定しています。この例のステートマシン定義の詳細については、「分散マップを使用した大規模 CSV データのコピー」のチュートリアルを参照してください。

```
{
  "Map": {
    "Type": "Map",
    "ItemReader": {
      "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "FIRST_ROW"
      },
      "Resource": "arn:aws:states:::s3:getObject",
      "Parameters": {
        "Bucket": "Database",
        "Key": "csv-dataset/ratings.csv"
      }
    },
    "ItemProcessor": {
      "ProcessorConfig": {
        "Mode": "DISTRIBUTED",
        "ExecutionType": "EXPRESS"
      },
      "StartAt": "LambdaTask",
      "States": {
        "LambdaTask": {
          "Type": "Task",
          "Resource": "arn:aws:states:::lambda:invoke",
          "OutputPath": "$.Payload",
          "Parameters": {
            "Payload.$": "$",
            "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:processCSVData"
          },
          "End": true
        }
      }
```

```
    },
    "Label": "Map",
    "End": true,
    "ResultWriter": {
      "Resource": "arn:aws:states:::s3:putObject",
      "Parameters": {
        "Bucket": "myOutputBucket",
        "Prefix": "csvProcessJobs"
      }
    }
  }
}
```

分散マップを実行するアクセス許可

ワークフローに分散マップ状態を含める場合、Step Functions には、ステートマシンのロールで分散マップ状態の StartExecution API アクションを呼び出すための適切な許可が必要です。

次の IAM ポリシー例では、ステートマシンのロールで分散マップ状態を実行するために必要な最小特権を付与しています。

> **Note**
>
> 必ず *stateMachineName* を、分散マップ状態を使用しているステートマシン名に置き換えてください。例えば arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine* です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": "arn:aws:states:region:accountID:execution:stateMachineName:*"
    }
  ]
}
```

さらに、Amazon S3 バケットなど、Distributed Map AWS 状態で使用されるリソースにアクセスするのに必要な最小限の権限を持っていることを確認する必要があります。詳細については、分散マップ状態を使用するための IAM ポリシー を参照してください。

分散マップ状態のフィールド

ワークフローで分散マップ状態を使用するには、これらのフィールドを 1 つ以上指定します。これらのフィールドは、共通状態フィールドに加えて指定します。

`Type` (必須)

状態のタイプ (`Map` など) を設定します。

`ItemProcessor` (必須)

`Map` 状態処理モードと定義を指定する次の JSON オブジェクトが含まれます。

- `ProcessorConfig` - `Map` 状態の設定を指定する JSON オブジェクト。このオブジェクトには、以下のサブフィールドが含まれています。
  - `Mode` - `Map` 状態を分散モードで使用するには **`DISTRIBUTED`** に設定します。

    > **Note**
    >
    > 現在、Express ワークフロー内で `Map` 状態を使用する場合、`Mode` を `DISTRIBUTED` に設定することはできません。ただし、Standard ワークフロー内で `Map` 状態を使用する場合、`Mode` を `DISTRIBUTED` に設定できます。

  - `ExecutionType` - マップワークフローの実行タイプを [STANDARD] または [EXPRESS] のいずれかに指定します。`DISTRIBUTED` を `Mode`サブフィールドに指定した場合は、このフィールドを指定する必要があります。ワークフロータイプの詳細については、標準ワークフロー対 Express ワークフロー を参照してください。

- `StartAt` - ワークフローの最初の状態を示す文字列を指定します。この文字列は、大文字と小文字が区別され、いずれかの状態オブジェクトの名前と完全に一致する必要があります。この状態は、データセット内の各アイテムで最初に実行されます。Map 状態に提供した実行入力は、まず `StartAt` 状態に渡されます。
- `States` – カンマで区切られた一連の状態を含む JSON オブジェクト。このオブジェクトでは、Map workflowを定義します。

`ItemReader`

データセットとその場所を指定します。Map 状態は指定されたデータセットから入力データを受け取ります。

分散モードでは、以前の状態から渡された JSON ペイロードまたは大規模な Amazon S3 データソースのいずれかをデータセットとして使用できます。詳細については、「ItemReader」を参照してください。

`ItemsPath` (オプション)

ステート入力内の項目の配列を含む JSON JsonPathノードを選択する構文を使用して参照パスを指定します。

分散モードでは、前のステップの JSON 配列を状態入力として使用する場合にのみ、このフィールドを指定します。詳細については、「ItemsPath」を参照してください。

`ItemSelector` (オプション)

各 Map 状態反復に渡される前に、個々のデータセットアイテムの値をオーバーライドします。

このフィールドでは、キーと値のペアのコレクションを含む有効な JSON 入力を指定します。これらのペアは、ステートマシン定義で定義する静的な値でも、パスを使用して状態入力から選択された値でも、コンテキストオブジェクトからアクセスされる値でもかまいません。詳細については、「ItemSelector」を参照してください。

`ItemBatcher` (オプション)

データセットアイテムの一括処理を指定します。子ワークフローを実行するたびに、入力としてこれらのバッチを受け取ります。詳細については、「ItemBatcher」を参照してください。

`MaxConcurrency` (オプション)

並行して実行できる子ワークフローの実行数を指定します。インタープリタは、指定された回数までの子ワークフローの並行実行のみを許可します。同時実行値を指定しなかったり 0 に設定したりしない場合、Step Functions は同時実行を制限せず、子ワークフローを 10,000 回並行実行します。

> **Note**
>
> 子ワークフローのparallel 実行にはより高い同時実行制限を指定できますが、 AWS などのダウンストリームサービスの容量を超えないようにすることをお勧めします。 AWS Lambda

`MaxConcurrencyPath` (オプション)

リファレンスパスを使用して状態の入力からタイムアウト値を動的に最大同時実行値を指定する場合は、`MaxConcurrencyPath` を使用してください。解決されると、リファレンスパスは、値が負でない整数のフィールドを選択する必要があります。

> **Note**
>
> Map 状態には `MaxConcurrency` と `MaxConcurrencyPath` の両方を含めることはできません。

`ToleratedFailurePercentage` (オプション)

マップ実行で許容できる失敗項目の割合を定義します。この割合を超えると、マップ実行は自動的に失敗します。Step Functions は、失敗またはタイムアウトしたアイテムの総数をアイテムの総数で割った結果として、失敗したアイテムの割合を計算します。0 から 100 までの値を指定する必要があります。詳細については、「分散マップ状態の許容される失敗しきい値」を参照してください。

`ToleratedFailurePercentagePath` (オプション)

リファレンスパスを使用して状態の入力から許容される失敗の割合の値を動的に指定したい場合は、`ToleratedFailurePercentagePath` を使用してください。解決されると、リファレンスパスは、値が 0 ~ 100の フィールドを選択する必要があります。

`ToleratedFailureCount` (オプション)

マップ実行で許容される障害アイテムの数を定義します。この数を超えると、マップ実行は自動的に失敗します。詳細については、「分散マップ状態の許容される失敗しきい値」を参照してください。

`ToleratedFailureCountPath` (オプション)

リファレンスパスを使用して状態の入力から許容される失敗数の値を動的に指定したい場合は、`ToleratedFailureCountPath` を使用してください。解決されると、リファレンスパスは、値が負でない整数のフィールドを選択する必要があります。

`Label` (オプション)

`Map` 状態を一意に識別する文字列。Step Functions は、マップ実行ごとにマップ実行 ARN にラベルを追加します。以下は、`demoLabel` という名前のカスタムラベルを持つマップ実行 ARN の例です。

```
arn:aws:states:us-east-1:123456789012:mapRun:demoWorkflow/
demoLabel:3c39a231-69bb-3d89-8607-9e124eddbb0b
```

ラベルを指定しない場合、Step Functions では一意のラベルが自動的に生成されます。

> **Note**
>
> ラベルは 40 文字を超える文字を含めることができず、1 台のステートマシン定義内で一意である必要があり、次の文字を含めることはできません。
>
> - ホワイトスペース文字
> - ワイルドカード文字 (`?` `*`)
> - 角かっこ (`<` `>` `{` `}` `[` `]`)
> - 特殊文字 (`:` `;` `,` `\` `|` `^` `~` `$` `#` `%` `&` `` ` `` `"`)
> - 制御文字 (`\\u0000` - `\\u001f` または `\\u007f` - `\\u009f`)
>
> Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

`ResultWriter` (オプション)

Step Functions がすべての子ワークフローの実行結果を書き込む Amazon S3 内の場所を指定します。

Step Functions は、実行の入出力、ARN、実行状態など、すべての子ワークフローの実行データを統合します。次に、指定した Amazon S3 の場所のそれぞれのファイルに、同じステータスの実行をエクスポートします。詳細については、「ResultWriter」を参照してください。

Map 状態の結果をエクスポートしない場合、すべての子ワークフロー実行結果の配列が返されます。例:

```
[1, 2, 3, 4, 5]
```

`ResultPath` (オプション)

反復の出力を配置する入力内の場所を指定します。その後、入力は `OutputPath` フィールド (ある場合) に従ってフィルタリングされてから状態の出力として渡されます。詳細については、入力および出力処理を参照してください。

`ResultSelector` (オプション)

値が静的であるか、結果から選択されたキーバリューのペアの集合を渡します。詳細については、「ResultSelector」を参照してください。

> **Tip**
>
> ステートマシンで使用している並列またはマップステートが配列の配列を返す場合は、ResultSelector フィールドを使用して配列をフラットな配列に変換できます。詳細については、「配列の配列の平坦化」を参照してください。

`Retry` (オプション)

Retrier と呼ばれるオブジェクトの配列。再試行ポリシーを定義します。状態でランタイムエラーが発生した場合、実行では再試行ポリシーが使用されます。詳細については、「Retry と Catch を使用するステートマシンの例」を参照してください。

> **Note**
>
> 分散マップ状態に Retrier を定義すると、Map ステートが開始したすべての子ワークフロー実行に再試行ポリシーが適用されます。例えば、Map 状態で 3 つの子ワークフロー実行が開始され、そのうちの 1 つが失敗したとします。障害が発生すると、実行では Map 状態の `Retry` フィールド (定義されている場合) が使用されます。再試行ポリシーは、失敗した実行だけでなく、すべての子ワークフロー実行に適用されます。1 つ以上の子ワークフローの実行が失敗すると、マップ実行は失敗します。

Map 状態を再試行すると、新しいマップ実行が作成されます。

`Catch` (オプション)

Catcher と呼ばれるオブジェクトの配列で、フォールバック状態を定義します。Step Functions は、状態で実行時エラーが発生した場合に、`Catch` で定義されている Catcher を使用します。エラーが発生すると、実行は最初に `Retry` で定義されている Retrier を使用します。再試行ポリシーが定義されていないか使い果たされた場合、実行ではその Catcher (定義されている場合) を使用します。詳細については、「フォールバック状態」を参照してください。

次のステップ

分散マップ状態の学習を続けるには、次のリソースを参照してください。

- 入力および出力処理

  分散マップ状態が受け取る入力と生成される出力を設定するために、Step Functions には以下のフィールドが用意されています。

  - ItemReader
  - ItemsPath
  - ItemSelector
  - ItemBatcher
  - ResultWriter
  - 入力 CSV ファイルの解析

  これらのフィールドに加えて、Step Functions では分散マップの許容障害しきい値を定義することもできます。この値により、失敗したアイテムの最大数または割合をマップ実行の失敗しきい値として指定できます。許容障害数のしきい値の設定の詳細については、「分散マップ状態の許容される失敗しきい値」を参照してください。

- 分散マップ状態の使用

  分散マップ状態の使用を開始するには、以下のチュートリアルとサンプルプロジェクトを参照してください。

  - 分散マップ状態の使用開始
  - Lambda 関数でデータバッチ全体を処理する

- Lambda 関数で個々のデータ項目を処理する
- サンプルプロジェクト: 分散マップで CSV ファイルを処理する
- サンプルプロジェクト: Amazon S3 バケットのデータを分散マップで処理する

- 分散マップ状態の実行を調べる

  Step Functions コンソールには、分散マップ状態実行に関連するすべての情報を表示する [マップ実行の詳細] ページがあります。このページに表示される情報を調べる方法については、「マップ実行の確認」を参照してください。

分散マップ状態の許容される失敗しきい値

大規模な並行ワークロードをオーケストレーションする場合、許容される失敗しきい値を定義することもできます。この値により、失敗したアイテムの最大数または割合をマップ実行の失敗しきい値として指定できます。指定した値によっては、しきい値を超えるとマップ実行が自動的に失敗します。両方の値を指定した場合、いずれかの値を超えるとワークフローは失敗します。

しきい値を指定すると、マップ実行全体が失敗する前に、特定の数の項目を失敗させることができます。Step Functions は、指定されたしきい値を超えたためにマップ実行が失敗すると `States.ExceedToleratedFailureThreshold` エラーを返します。

> **Note**
>
> Step Functions は、許容される失敗しきい値を超えた後でも、マップ実行が失敗する前に、マップ実行で子ワークフローを実行し続ける場合があります。

Workflow Studio でしきい値を指定するには、[ランタイム設定] フィールドの [追加設定]で [許容される失敗しきい値を設定] を選択します。

許容される失敗の割合

許容される失敗項目の割合を定義します。この値を超えるとマップ実行は失敗します。Step Functions は、失敗した項目またはタイムアウトした項目の合計数を項目の合計数で割った結果として、失敗した項目の割合を計算します。0 から 100 までの値を指定する必要があります。割合のデフォルトの値は 0 です。つまり、子ワークフローのいずれかの実行が失敗またはタイムアウトすると、ワークフローは失敗します。割合を 100 に指定すると、子ワークフローの実行がすべて失敗しても、ワークフローは失敗しません。

別の方法として、分散マップ状態の入力での既存のキーと値のペアへの参照パスとして割合を指定することもできます。このパスは、実行時に 0 ~ 100 の間の正の整数に変換されなければなりません。ToleratedFailurePercentagePath サブフィールドに参照パスを指定します。

例として、次の入力があるとします。

```
{
  "percentage": 15
}
```

その入力への参照パスを使用して、次のように割合を指定できます。

```
{
  ...
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailurePercentagePath": "$.percentage"
    ...
  }
}
```

> **Important**
>
> 分散マップ状態の定義では、ToleratedFailurePercentage または ToleratedFailurePercentagePath のどちらかを指定できますが、両方は指定できません。

許容される失敗数

許容される失敗項目の数を定義します。この値を超えるとマップ実行は失敗します。

別の方法として、分散マップ状態の入力での既存のキーと値のペアへの参照パスとして数を指定することもできます。このパスは実行時に正の整数に変換されなければなりません。ToleratedFailureCountPath サブフィールドに参照パスを指定します。

例として、次の入力があるとします。

```
{
```

```
  "count": 10
}
```

その入力への参照パスを使用して、次のように数を指定できます。

```
{
  ...
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailureCountPath": "$.count"
    ...
  }
}
```

> **Important**
>
> 分散マップ状態の定義では、ToleratedFailureCount または ToleratedFailureCountPath のどちらかを指定できますが、両方は指定できません。

Transitions

ステートマシンの新しい実行を開始すると、システムは最上位の StartAt フィールドで参照されている状態で開始されます。このフィールドは文字列として指定され、ワークフロー内の状態の名前と正確に一致する必要があります (大文字と小文字が区別されます)。

状態の実行後、AWS Step Functions は Next フィールドの値を使用して、進むべき次の状態を判断します。

Next フィールドでは、状態名を文字列として指定することもできます。この文字列では大文字と小文字が区別され、ステートマシンの説明で指定されたステートの名前と正確に一致する必要があります。

例えば、次の状態には NextState への移行が含まれています。

```
"SomeState" : {
  ...,
```

```
  "Next" : "NextState"
}
```

ほとんどの状態では、Next フィールドによる 1 つの移行ルールのみ許可されます。ただし、特定のフロー制御ステート (Choice ステートなど) では、それぞれ独自の Next フィールドを使用して複数の移行ルールを指定できます。Amazon ステートメント言語では、移行の指定方法など、指定可能な各状態タイプについて詳しく説明されています。

状態には、他の状態からの受信移行を複数指定できます。

プロセスは、終了状態 ("Type": Succeed、"Type": Fail、または "End": true の状態) のいずれかに到達するまで繰り返されます。そうでない場合は、ランタイムエラーが発生します。

実行を redrive 実行すると状態遷移とみなされます。さらに、redrive で再実行されるすべての状態も状態遷移とみなされます。

ステートマシン内の状態には次のルールが適用されます。

- 状態は、囲みブロック内ではどのような順序で発生してもかまいません。ただし、リストに記載されている順序は、実行順序には影響しません。この順序は状態の内容によって決まります。
- ステートマシン内では、start 状態として指定できるのは 1 つの状態のみです。start 状態は最上位構造内の StartAt フィールドの値によって定義されます。
- ステートマシンのロジックによっては (ステートマシンに複数のロジックブランチがあるかどうかなど)、複数の状態が存在する可能性があります。
- ステートマシンが 1 つの状態のみで構成される場合、開始状態と終了状態の両方になることがあります。

分散マップの状態の遷移

Map 状態を分散モードで使用すると、分散マップ状態が開始された子ワークフロー実行ごとに 1 つの状態遷移に対して課金されます。Map 状態をインラインモードで使用すると、状態遷移は、インラインマップ状態が繰り返されるごとには課金されません。

Map 状態を分散モードで使用することによりコストを最適化し、ネストされたワークフローを Map 状態の定義に含めることができます。また、分散マップ状態は、Express タイプの子ワークフロー実行を開始するときに、より高い価値を発揮します。Step Functions は Express の子ワークフロー実行のレスポンスとステータスを保存します。そのため、実行データを CloudWatch Logs に保存す

る必要がなくなります。エラーしきい値の定義や項目グループのバッチ処理など、分散マップの状態で使用できるフロー制御にアクセスすることもできます。Step Functions の料金情報については、「AWS Step Functions の料金」を参照してください。

ステートマシンデータ

ステートマシンのデータは、次の形式です。

- ステートマシンへの最初の入力
- 状態間で渡されるデータ
- ステートマシンからの出力

このセクションでは、ステートマシンのデータを AWS Step Functions でフォーマットして使用する方法について説明します。

トピック

- データ形式
- ステートマシンの入出力
- 状態の入出力

データ形式

ステートマシンデータは JSON テキストで表されます。JSON によってサポートされる任意のデータ型を使用して、ステートマシンに値を指定できます。

> **Note**
>
> - JSON テキスト形式の数値は、JavaScript セマンティクスに準拠します。これらの数値は通常、倍精度 IEEE-854 値に対応しています。
> - 有効な JSON テキストを次に示します。
>   - 引用符で区切られたスタンドアロンの文字列
>   - オブジェクト
>   - 配列
>   - 数字

- ブール値
- null

- 状態の出力は、次の状態の入力になります。ただし、入力および出力処理を使用することにより、状態が入力データのサブセットでのみ機能するよう制限できます。

ステートマシンの入出力

初期入力データを AWS Step Functions ステートマシンに渡すには、2 つある方法のどちらかを使用します。実行を開始すると、データを StartExecution アクションに渡すことができます。Step Functions コンソールから、データをステートマシンに渡すこともできます。初期データは、ステートマシンの StartAt 状態に渡されます。入力が提供されない場合、デフォルトは空のオブジェクト ({}) です。

実行の出力は、最後の状態 (terminal) により返されます。この出力は、実行の結果に JSON テキストとして表示されます。

標準ワークフローの場合、DescribeExecution アクションなど、外部呼び出し元を使用して実行履歴から実行結果を取得できます。実行結果は、Step Functions コンソールで確認できます。

Express ワークフローについては、ログ記録を有効にしている場合は、CloudWatch Logs から結果を取得したり Step Functions コンソールで実行を表示およびデバッグしたりすることができます。詳細については、「CloudWatch Logs を使用したログ記録」および「Step Functions コンソールでの実行の表示とデバッグ」を参照してください。

ステートマシンに関連するクォータも考慮する必要があります。詳細については、「クォータ」を参照してください。

状態の入出力

各状態の入力は、前の状態の JSON テキストで構成されます。または、StartAt 状態の場合、実行への入力で構成されます。特定のフロー制御状態は、その出力への入力を エコー します。

次の例では、ステートマシンが 2 つの数値を同時に追加します。

1. AWS Lambda 関数を定義します。

```
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 });
}
```

2. ステートマシンを定義します。

```
{
  "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
        }
    }
}
```

3. 次の JSON テキストで実行を開始します。

```
{ "numbers": [3, 4] }
```

Add 状態が JSON テキストを受け取り、Lambda 関数に渡します。

Lambda 関数は、計算の結果をその状態に返します。

状態は、その出力で次の値を返します。

```
{ "result": 7 }
```

Add はステートマシンの最終状態でもあるため、この値はステートマシンの出力として返されます。

最終状態が出力を返さない場合、ステートマシンは空のオブジェクト ({}) を返します。

詳細については「Step Functions の入出力処理」を参照してください。

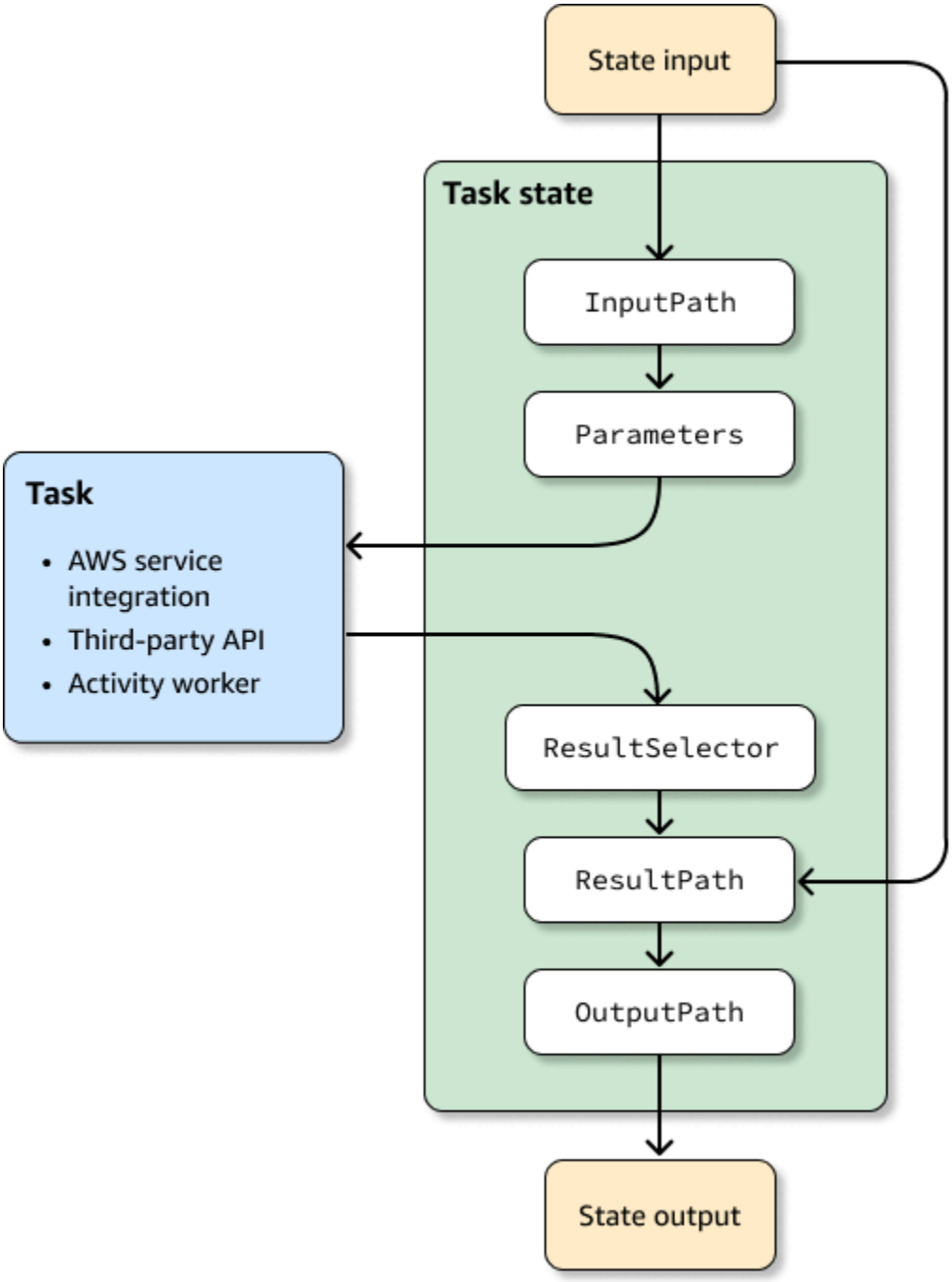
Step Functions の入出力処理

Step Functions を実行すると、JSON テキストを入力として受け取り、その入力をワークフローの最初の状態に渡します。それぞれの状態で JSON を入力として受け取ります。また、通常 JSON を出力として次の状態に渡します。 AWS Step Functions のワークフローを効率的に設計し、実装するには、状態間のこの情報の流れを理解して、このデータのフィルタリングや操作の方法を学ぶことが重要です。

Amazon States Language では、これらのフィールドを使用して、状態によって異なる JSON のフローをフィルタおよび制御します。

- `InputPath`
- `Parameters`
- `ResultSelector`
- `ResultPath`
- `OutputPath`

次の図は、JSON 情報がタスク状態をどのように通過するかを示しています。 は`Task`、状態のタスクに渡す JSON 入力の部分 (AWS Lambda 関数など) `InputPath`を選択します。`ResultPath`次に、 は、出力に渡す状態入力とタスク結果の組み合わせを選択します。 は、JSON 出力をフィルタリングして、出力に渡される情報をさらに制限`OutputPath`できます。



InputPath、Parameters、ResultSelector、ResultPath、および OutputPath ではそれぞれ JSON を使用します。ワークフローの各状態に遷移しているからです。

それぞれがパスを使用して、入力または結果から JSON の部分を選択することができます。パスは $ で始まる文字列です。これを使用して、JSON テキスト内でノードを識別します。Step Functions パスは JsonPath構文を使用します。

> **Tip**
>
> Step Functions コンソールのデータフローシミュレーターを使用して、JSON パス構文をテストし、ステート内でのデータの操作方法に関する理解を深め、ステート間でデータを渡す方法を確認します。

> **Tip**
>
> 入出力処理を含むワークフローの例を にデプロイするには AWS アカウント、「 AWS Step Functions ワークショップ」の「モジュール 6 - 入出力処理」を参照してください。

トピック

- パス
- InputPath、パラメータ、 ResultSelector
- ResultPath
- OutputPath
- InputPath、ResultPath、OutputPath の例
- マップステートの入力および出力フィールド
- コンテキストオブジェクト

パス

Amazon States Language では、パスは、JSON テキスト内でコンポーネントを識別するために使用できる $ で始まる文字列です。パスはJsonPath構文に従います。`InputPath`、`ResultPath`、および `OutputPath` に値を指定するときに、入力のサブセットにアクセスするパスを指定できます。詳細については、Step Functions の入出力処理 を参照してください。

> **Note**
>
> また、状態の定義の `Parameters` フィールド内のパスを使用して、入力あるいはコンテキストオブジェクトの JSON ノードを指定することもできます。サービス API にパラメータを渡す を参照してください。

フィールド名に JsonPath ABNF ルールmember-name-shorthandの定義に含まれていない文字が含まれている場合は、角括弧表記を使用する必要があります。そのため、句読点などの特殊文字 (_ を除外) をエンコードするには、角かっこ表記を使用する必要があります。例えば `$.abc.['def ghi']` です。

リファレンスパス

リファレンスパスは、JSON 構造内の単一ノードのみを識別できるように構文が制限されたパスです。

- オブジェクトのフィールドにアクセスするには、ドット (.) と角括弧 ([ ]) のみを使用して表記します。
- `length()` などの関数はサポートされていません。
- `subsetof` などのシンボリックではないレキシカル演算子はサポートされていません。
- 正規表現によるフィルタリングや、JSON 構造内の別の値の参照によるフィルタリングはサポートされていません。
- フィルター処理される現在のノードを照合する @ 演算子は、スカラー値は照合しません。オブジェクトのみを照合します。

例えば、if 状態入力データには次の値が含まれます。

```
{
  "foo": 123,
  "bar": ["a", "b", "c"],
  "car": {
      "cdr": true
  },
  "jar": [{"a": 1}, {"a": 5}, {"a": 2}, {"a": 7}, {"a": 3}]
}
```

次のリファレンスパスは以下を返します。

```
$.foo => 123
$.bar => ["a", "b", "c"]
$.car.cdr => true
$.jar[?(@.a >= 5)] => [{"a": 5}, {"a": 7}]
```

特定の状態はパスおよびリファレンスパスを使用して、ステートマシンのフローの制御、または状態の設定やオプションを構成します。詳細については、「データフローシミュレータを使用したワー

クフローの入力パスと出力パスの処理のモデリング」および「 での JSONPath を効果的に使用する AWS Step Functions」を参照してください。

配列の配列の平坦化

ステートマシンの 並行 または マッピング 状態により配列の配列が返される場合は、ResultSelector フィールドを使用してフラットな配列に変換できます。このフィールドを並列ステートまたはマップステートの定義に含めて、これらのステートの結果を操作できます。

配列をフラット化するには、次の例のように ResultSelector フィールドで JMESPath 構文 [*] を使用します。

```
"ResultSelector": {
    "flattenArray.$": "$[*][*]"
  }
```

配列をフラット化する方法を示す例については、以下のチュートリアルのステップ 3 を参照してください。

- Lambda 関数でデータバッチ全体を処理する
- Lambda 関数で個々のデータ項目を処理する

InputPath、パラメータ、 ResultSelector

JSON はワークフローを通して動くため、InputPath、Parameters、および ResultSelector フィールドから JSON の操作方法を提供します。InputPath は、パスを使用して JSON 表記をフィルタリングして渡される入力を制限できます (パス を参照)。Parameters フィールドによって、ステートマシンで定義した静的値あるいはパスを使用した入力から選択された静的値のいずれかの値となるキー値ペアのコレクションを渡すことができるようになります。ResultSelector フィールドは、ResultPath が適用される前の状態を操作する方法を提供します。

AWS Step Functions は最初に InputPathフィールドを適用し、次に Parametersフィールドを適用します。最初に InputPath を使用して raw 入力をフィルタリングしてから、Parameters を適用して、入力をさらに操作するか、新しい値を追加します。その後、ResultSelector フィールドを使って、ResultPath を適用する前に、状態の出力を操作できます。

> **Tip**
>
> Step Functions コンソールのデータフローシミュレーターを使用して、JSON パス構文をテストし、ステート内でのデータの操作方法に関する理解を深め、ステート間でデータを渡す方法を確認します。

InputPath

状態入力の一部を選択するには、InputPath を使用します。

例えば、状態への入力に以下が含まれているとします。

```
{
  "comment": "Example for InputPath.",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

InputPath を適用できます。

```
"InputPath": "$.dataset2",
```

上記の InputPath を使用すると、入力として渡される JSON は次のようになります。

```
{
  "val1": "a",
  "val2": "b",
  "val3": "c"
}
```

> **Note**
>
> パスは値の選択になる場合があります。次の例を考えます。
>
> ```
> { "a": [1, 2, 3, 4] }
> ```
>
> パス `$.a[0:2]` を適用すると、結果は次のとおりです。
>
> ```
> [ 1, 2 ]
> ```

パラメータ

このセクションでは、[Parameters] (パラメータ) フィールドを使用するさまざまな方法を示します。

キーバリューペア

入力として渡されるキーバリューのペアのコレクションを作成するには、`Parameters` フィールドを使用します。それぞれの値は、ステートマシンの定義に含める静的な値、または入力からパスまたはコンテキストオブジェクトで選択した静的な値を使用できます。パスを使用して値を選択するキーと値のペアの場合、キーの名前は `.$` で終わる必要があります。

例えば、以下の入力を使用するとします。

```
{
  "comment": "Example for Parameters.",
  "product": {
    "details": {
       "color": "blue",
       "size": "small",
       "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

いくつかの情報を選択するには、これらのパラメータをステートマシン定義で指定できます。

```
"Parameters": {
```

```
        "comment": "Selecting what I care about.",
        "MyDetails": {
          "size.$": "$.product.details.size",
          "exists.$": "$.product.availability",
          "StaticValue": "foo"
        }
      },
```

前の入力と Parameters フィールドを指定する場合、これが渡される JSON となります。

```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
      "size": "small",
      "exists": "in stock",
      "StaticValue": "foo"
  }
},
```

入力に加えて、コンテキストオブジェクトとして知られる特別な JSON にアクセスできます。コンテキストオブジェクトには、ステートマシン実行に関する情報が含まれています。コンテキストオブジェクト を参照してください。

コネクテッドリソース

Parameters フィールドを使用して、接続されたリソースに情報を渡すこともできます。例えば、タスクの状態が AWS Batch ジョブをオーケストレーションしている場合、関連する API パラメータをそのサービスの API アクションに直接渡すことができます。詳細については、以下を参照してください。

- サービス API にパラメータを渡す
- 他の サービスでの使用

Amazon S3

状態間で渡す Lambda 関数のデータが 262,144 バイトを超える場合、Amazon S3 を使用してデータを保存し、次のいずれかの方法を実装することを推奨いたします。

- ワークフローで 分散マップ状態を使用して、Map ステートが Amazon S3 データソースからの入力を直接読み取れるようにします。詳細については、「分散マップ状態の使用」を参照してください。

- Payload パラメータ内のバケットの Amazon リソースネーム (ARN) を解析して、バケット名とキー値を取得します。詳細については、「ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する」を参照してください。

または、実行時に小さいペイロードを渡すように実装を調整します。

ResultSelector

ResultPath を適用する前に、ResultSelector フィールドを利用して、状態の結果を操作します。ResultSelector フィールドでは、キーバリューペアのコレクションを作成できます。ここでは、値は静的であるか、状態の結果から選択されます。ResultSelector フィールドを使用すると、状態の結果のどの部分を ResultPath フィールドに渡すかを選択できます。

> **Note**
>
> ResultPath フィールドを使用すると、ResultSelector フィールドの出力を元の入力に追加できます。

ResultSelector は、次の状態のオプションのフィールドです。

- マッピング
- タスク
- 並行

例えば、Step Functions サービス統合は、結果のペイロードに加えてメタデータを返します。ResultSelector は結果の一部を選択し、ResultPath を含む状態の入力とマージできます。この例では、resourceType と ClusterId だけを選択し、Amazon EMR CreateCluster.sync からの状態入力とマージします。次のものがあるとします。

```
{
  "resourceType": "elasticmapreduce",
  "resource": "createCluster.sync",
  "output": {
    "SdkHttpMetadata": {
      "HttpHeaders": {
        "Content-Length": "1112",
        "Content-Type": "application/x-amz-JSON-1.1",
        "Date": "Mon, 25 Nov 2019 19:41:29 GMT",
```

```
        "x-amzn-RequestId": "1234-5678-9012"
      },
      "HttpStatusCode": 200
    },
    "SdkResponseMetadata": {
      "RequestId": "1234-5678-9012"
    },
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

その後、ResultSelector を使って resourceType と ClusterId を選択できます。

```
"Create Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::elasticmapreduce:createCluster.sync",
  "Parameters": {
    <some parameters>
  },
  "ResultSelector": {
    "ClusterId.$": "$.output.ClusterId",
    "ResourceType.$": "$.resourceType"
  },
  "ResultPath": "$.EMROutput",
  "Next": "Next Step"
}
```

与えられた入力で、ResultSelector を使用して以下を生成します。

```
{
  "OtherDataFromInput": {},
  "EMROutput": {
    "ResourceType": "elasticmapreduce",
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

配列の配列の平坦化

ステートマシンの 並行 または マッピング 状態により配列の配列が返される場合は、ResultSelector フィールドを使用してフラットな配列に変換できます。このフィールドを並列ステートまたはマップステートの定義に含めて、これらのステートの結果を操作できます。

配列をフラット化するには、次の例のように ResultSelector フィールドで JMESPath 構文 [*] を使用します。

```
"ResultSelector": {
    "flattenArray.$": "$[*][*]"
  }
```

配列をフラット化する方法を示す例については、以下のチュートリアルのステップ 3 を参照してください。

- Lambda 関数でデータバッチ全体を処理する
- Lambda 関数で個々のデータ項目を処理する

ResultPath

状態の出力は、入力のコピー、生成される結果 (例: Task 状態の Lambda 関数からの出力)、またはその入力と結果の組み合わせです。ResultPath を使用して、上記のうち、状態出力に渡す組み合わせを制御します。

次の状態タイプでは、結果を生成できます。また、ResultPath: を含むことができます。

- パス
- タスク
- 並行
- マッピング

ResultPath を使用して、タスク結果とタスク入力を組み合わせるか、またはこれらのいずれかを選択します。ResultPath に指定したパスで、出力に渡す情報を制御します。

> **Note**
>
> ResultPath では、リファレンスパスの使用を制限します。これにより、JSON の単一ノードのみ識別できるように範囲を制限できます。Amazon ステートメント言語の リファレンスパス を参照してください。

これらの例は、Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルに記述されているステートマシンおよび Lambda 関数に基づきます。上記のチュートリアルを実行し、出力をテストするには、`ResultPath` フィールドにさまざまなパスを入力します。

ResultPath を使用して:

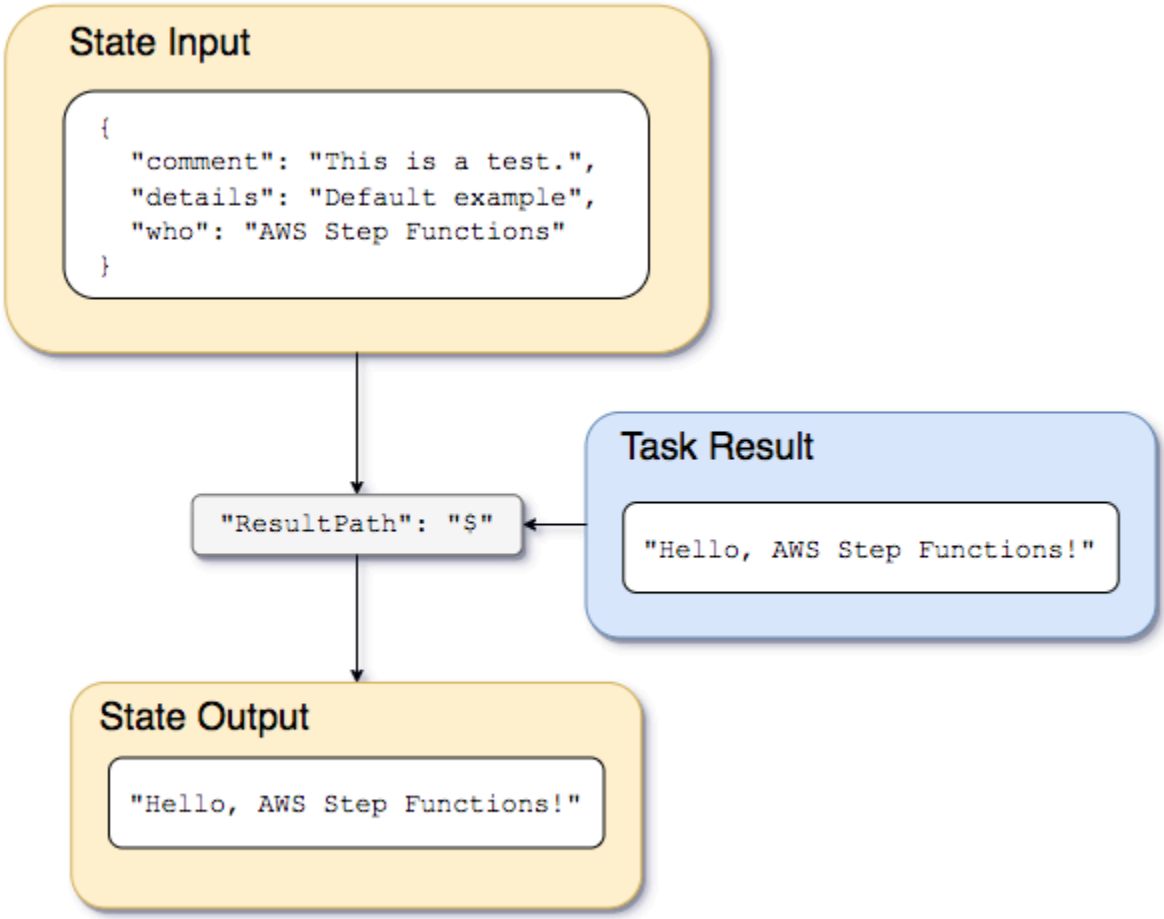
- ResultPath を使用して入力を結果に置き換える
- 結果を破棄し、元の入力を保持する
- ResultPath を使用して、結果を入力に含めます。
- ResultPath を使用して、結果を含む入力のノードを更新する
- ResultPath を使用して、エラーと入力の両方を に含める Catch

> **Tip**
>
> Step Functions コンソールのデータフローシミュレーターを使用して、JSON パス構文をテストし、ステート内でのデータの操作方法に関する理解を深め、ステート間でデータを渡す方法を確認します。

ResultPath を使用して入力を結果に置き換える

`ResultPath` を指定しない場合、デフォルトの動作は、`"ResultPath": "$"` を指定した場合と同様になります。上記の場合、入力全体を結果に置き換えるように状態に指示されるため、状態入力は、タスク結果によって生じる結果によって完全に置き換えられます。

次の図は、`ResultPath` によって、入力がタスクの結果に完全に置き換えられる様子を表します。



「Lambda を使用する Step Functions ステートマシン状態の作成」で説明されているステートマシンと Lambda 関数を使用し、Lambda 関数のサービス統合タイプを AWS SDK 統合に変更します。そのためには、次の例に示すように Task 状態の Resource フィールドで Lambda 関数の Amazon リソースネーム (ARN) を指定します。 AWS SDK 統合を使用すると、Task状態結果にはメタデータなしで Lambda 関数出力のみが含まれます。

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
         "Resource":"arn:aws:lambda:us-east-2:123456789012:function:HelloFunction",
         "End": true
      }
   }
```

```
}
```

次に、次の入力を渡します。

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 関数では、以下のような結果が生成されます。

```
"Hello, AWS Step Functions!"
```

> **Tip**
>
> この結果は、Step Functions コンソールで確認できます。そのためには、コンソールの実行詳細ページで、[グラフ表示]内の Lambda 関数を選択します。次に、ステップの詳細 ペインの [出力] タブを選択してこの結果を確認します。

`ResultPath` が状態で指定されていない場合、または `"ResultPath": "$"` が設定されている場合、状態の入力は、Lambda 関数の結果に置き換えられます。また、状態の出力は次のとおりです。

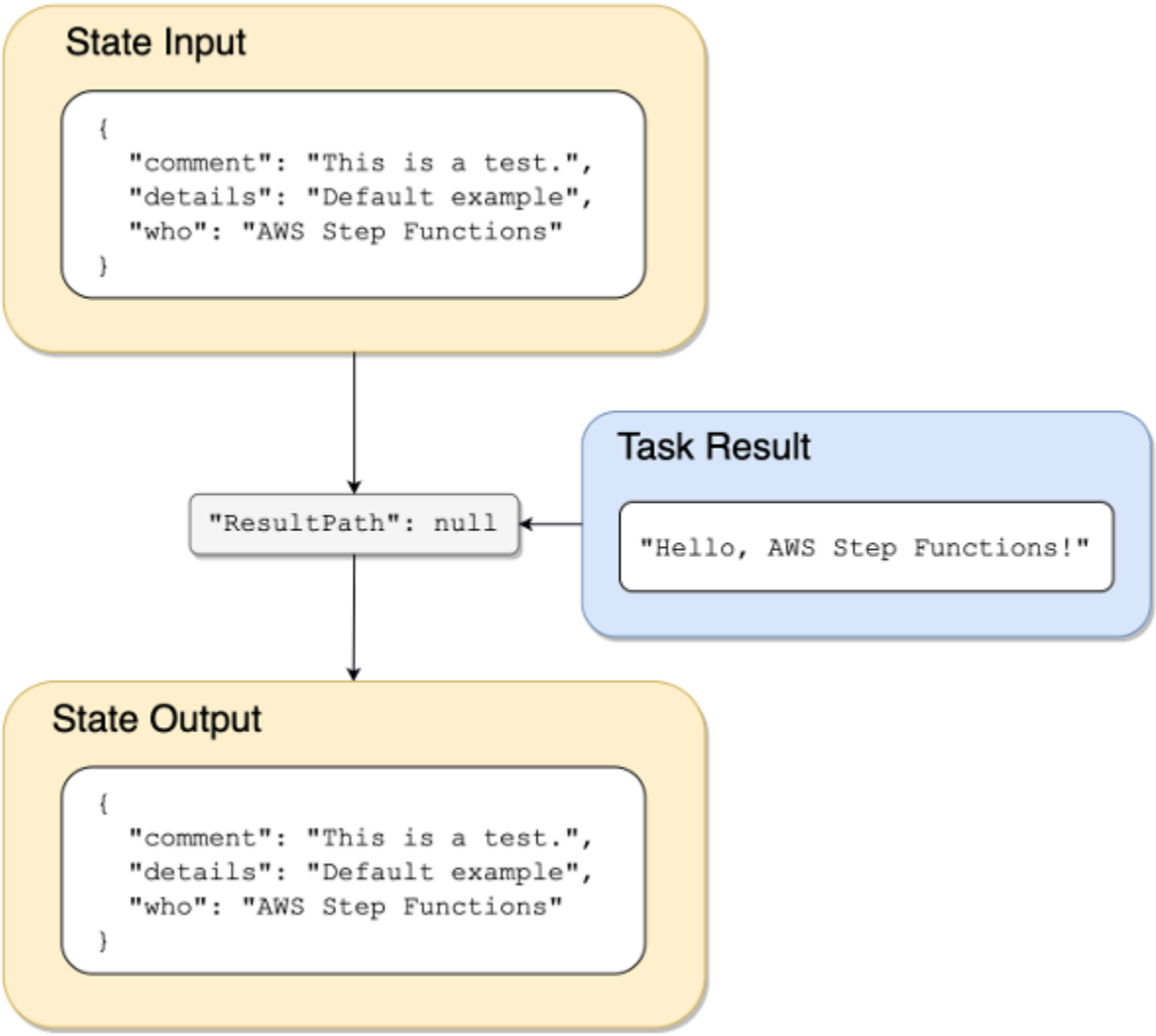
```
"Hello, AWS Step Functions!"
```

> **Note**
>
> `ResultPath` は、結果の内容を入力に含むために、出力に渡す前に使用します。ただし、`ResultPath` が指定されていない場合、デフォルトで入力全体が置き換えられます。

結果を破棄し、元の入力を保持する

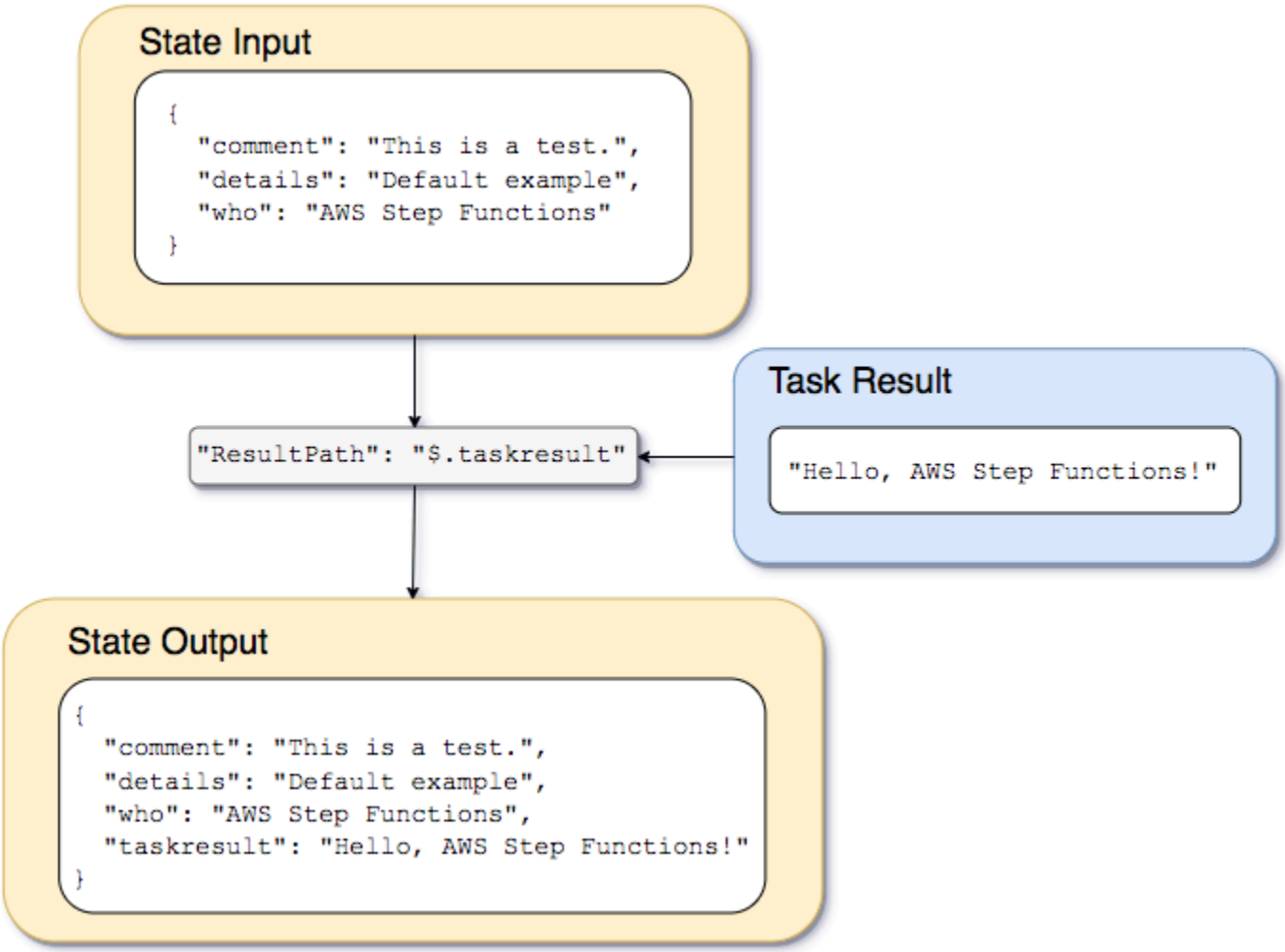
`ResultPath` を `null` に設定すると、元の入力が出力に渡されます。`"ResultPath": null` を使用すると、状態の入力ペイロードは、結果に関係なく、出力に直接コピーされます。

次の図は、null `ResultPath` が入力を直接出力にコピーする方法を示しています。



ResultPath を使用して、結果を入力に含めます。

次の図は、ResultPath を使用して結果を入力に含める方法を示します。



Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルで述べられているステートマシンおよび Lambda 関数を使用して、以下の入力を渡すことができます。

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 関数の結果は次のとおりです。

```
"Hello, AWS Step Functions!"
```

入力を保持するには、Lambda 関数の結果を挿入し、結合された JSON を次の状態に渡すために、次のように ResultPath を設定できます。

```
"ResultPath": "$.taskresult"
```

こうすると、元の入力がある Lambda 関数の結果が含められます。

```
{
  "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!"
}
```

Lambda 関数の出力は、taskresult の値として元の入力に付加されます。新しく挿入された値を含む入力は、次の状態に渡されます。

また、その結果を入力の子ノードに挿入することもできます。ResultPath を以下のように設定します。

```
"ResultPath": "$.strings.lambdaresult"
```

次の入力を使用して実行を開始します。

```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
    "string2": "bar",
    "string3": "baz"
  },
  "who": "AWS Step Functions"
}
```

Lambda 関数の結果は、入力の strings ノードの子として挿入されます。

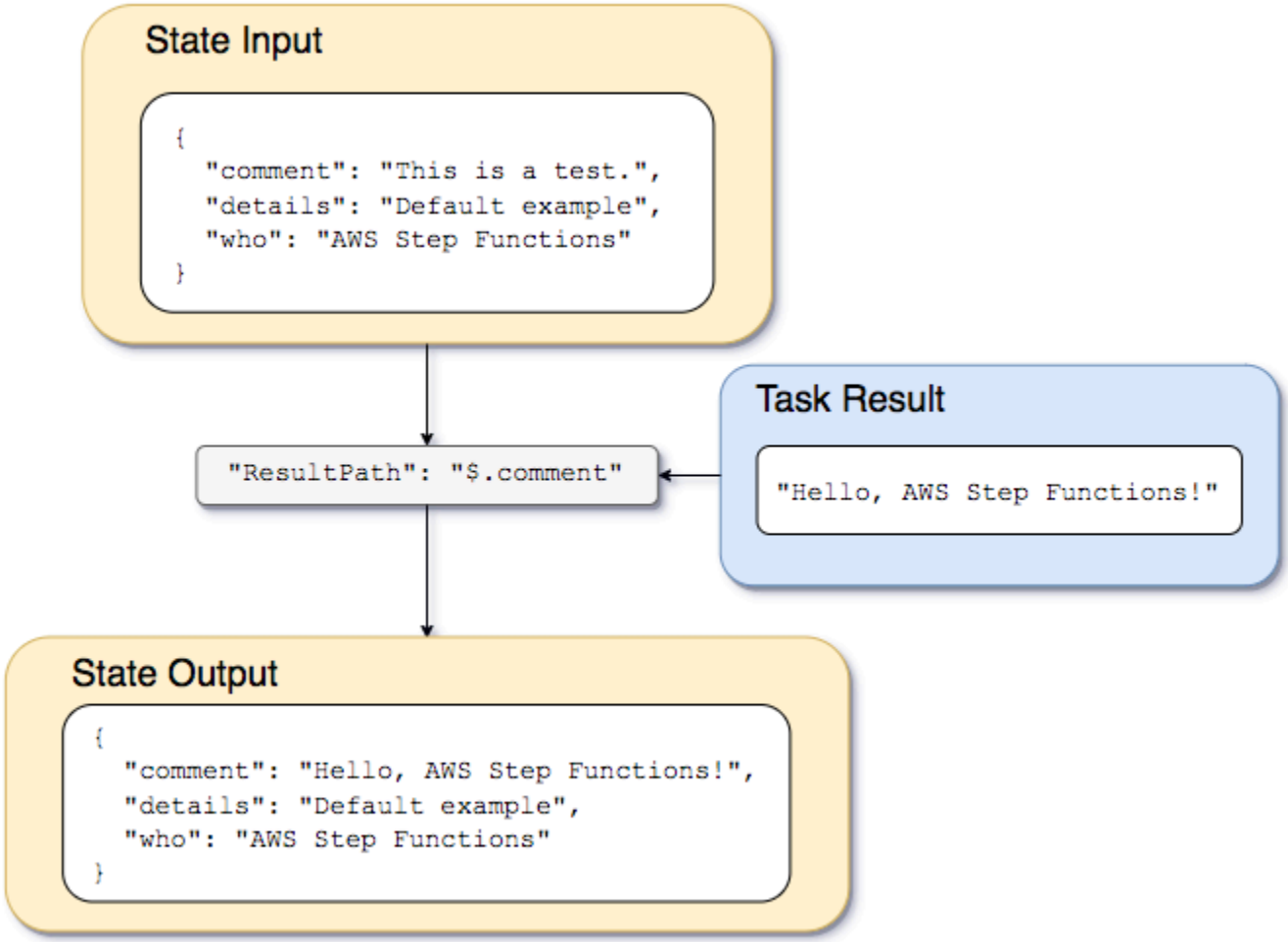
```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
```

```
    "string2": "bar",
    "string3": "baz",
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "who": "AWS Step Functions"
}
```

状態出力によって、子ノードとしての結果に元の入力 JSON が含まれるようになりました。

ResultPath を使用して、結果を含む入力のノードを更新する

次の図は、ResultPath を使用して、入力の既存 JSON ノードの値をタスク結果の値に更新する方法を示します。



Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルのステートマシンおよび Lambda 関数の例を使用して、 以下の入力を渡すことができます。

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 関数の結果は次のとおりです。

```
Hello, AWS Step Functions!
```

入力を保存し、その結果を JSON の新しいノードとして挿入せずに、既存のノードを上書きすることができます。

例えば、"ResultPath": "$" を省略または設定するとノード全体が上書きされるため、個々のノードを指定して結果を上書きすることができます。

```
"ResultPath": "$.comment"
```

comment ノードは状態入力に存在しているため、ResultPath を "$.comment" に設定すると、入力のノードは、Lambda 関数の結果に置き換えられます。OutputPath でフィルタリングしない場合、以下が出力に渡されます。

```
{
  "comment": "Hello, AWS Step Functions!",
  "details": "Default behavior example",
  "who": "AWS Step Functions",
}
```

comment ノードの値である "This is a test of the input and output of a Task state." は、Lambda 関数の結果 (状態出力の "Hello, AWS Step Functions!") によって置き換えられます。

ResultPath を使用して、エラーと入力の両方を に含める **Catch**

Step Functions ステートマシンを使用してエラー条件を処理する チュートリアルでは、ステートマシンを使用してエラーを検出する方法を示します。場合によっては、元の入力をエラーとともに保持する場合があります。Catch の ResultPath を使用して、エラーを置き換えずに元の入力に含めます。

```
"Catch": [{
```

```
  "ErrorEquals": ["States.ALL"],
  "Next": "NextTask",
  "ResultPath": "$.error"
}]
```

前の Catch ステートメントでエラーが検出された場合は、状態入力の error ノードに結果が含まれます。例えば、次の入力を示します。

```
{"foo": "bar"}
```

エラー検出時の状態出力は次のようになります。

```
{
  "foo": "bar",
  "error": {
    "Error": "Error here"
  }
}
```

エラー処理の詳細については、以下を参照してください。

- Step Functions のエラー処理
- Step Functions ステートマシンを使用してエラー条件を処理する

OutputPath

OutputPath を使用すると、状態の出力の一部を選択して次の状態に渡すことができるようになります。これにより、不要な情報をフィルタして、必要な一部の JSON のみを渡すことができるようになります。

OutputPath を指定しない場合、デフォルトの値は $ です。これによって、JSON ノード全体 (状態の入力、タスクの結果、および ResultPath によって決定) が次の状態に渡されます。

> **Tip**
>
> Step Functions コンソールのデータフローシミュレーターを使用して、JSON パス構文をテストし、ステート内でのデータの操作方法に関する理解を深め、ステート間でデータを渡す方法を確認します。

詳細については、次を参照してください。

- Amazon ステートメント言語のパス
- InputPath、ResultPath、OutputPath の例
- 静的 JSON をパラメータとして渡す
- Step Functions の入出力処理

InputPath、ResultPath、OutputPath の例

失敗 状態または 成功 状態以外のすべての状態には、InputPath、ResultPath、OutputPathなどの入力および出力処理フィールドを含めることができます。また、待機 および 選択 状態は ResultPath フィールドをサポートしていません。これらのフィールドを使用すると、JSJsonPath を使用してワークフロー内を移動する JSON データをフィルタリングできます。

Parameters フィールドを使用して、JSON データがワークフロー内を移動する際にそのデータを操作することもできます。Parameters の使用の詳細については、「InputPath、パラメータ、ResultSelector」を参照してください。

例えば、AWS Lambda チュートリアルに示す Lambda を使用する Step Functions ステートマシン状態の作成 関数とステートマシンから開始します。次の InputPath、ResultPath、OutputPath が含まれるように、ステートマシンを変更します。

```
{
  "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
    }
  }
}
```

次の入力を使用して実行を開始します。

```
{
  "comment": "An input comment.",
  "data": {
    "val1": 23,
    "val2": 17
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
  }
}
```

comment と extra ノードは破棄できますが、Lambda 関数の出力を含め、data ノードに情報を保存したいと思っています。

更新されたステートマシンで、Task 状態は、タスクへの入力を処理するように変更されます。

```
"InputPath": "$.lambda",
```

ステートマシン定義のこの行によって、タスク入力は、状態入力から lambda ノードのみに制限されます。Lambda 関数は、入力として JSON オブジェクト ({"who": "AWS Step Functions"}) のみを受信します。

```
"ResultPath": "$.data.lambdaresult",
```

この ResultPath は、元のステートマシン入力の data ノードの子として、Lambda 関数の結果を lambdaresult という名前のノードに挿入するよう指示します。元の入力および OutputPath を使った結果に対してこれ以上処理を実行しないため、状態の出力には元の入力を使った Lambda 関数の結果が含まれます。

```
{
  "comment": "An input comment.",
  "data": {
    "val1": 23,
    "val2": 17,
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
```

```
  }
}
```

しかし、目的は data ノードのみを保持し、Lambda 関数の結果を含めることです。OutputPath を使用して、状態出力に渡す前に、結合されたこの JSON をフィルタリングします。

```
"OutputPath": "$.data",
```

これにより、出力に渡される、元の入力の data ノード (例: lambdaresult によって挿入される ResultPath の子) のみ選択されます。状態出力は、次のようにフィルタリングされます。

```
{
  "val1": 23,
  "val2": 17,
  "lambdaresult": "Hello, AWS Step Functions!"
}
```

この Task 状態では次のようになります。

1. InputPath では、Lambda 関数への入力から lambda ノードのみ送信します。
2. ResultPath では、元の入力の data ノードの子として結果を挿入します。
3. OutputPath では、data ノードのみが状態出力に渡されるように、状態入力 (現在は Lambda 関数の結果が含まれる) がフィルタリングされます。

Example JsonPath を使用して、元のステートマシンの入力、結果、最終出力を操作する

保険申請者の身元と住所を検証する以下のステートマシンを考えてみましょう。

> **Note**
>
> 完全な例を表示するには、「Step Functions での JSON パスの使用方法」を参照してください。

```
{
  "Comment": "Sample state machine to verify an applicant's ID and address",
  "StartAt": "Verify info",
  "States": {
    "Verify info": {
```

```
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Verify identity",
          "States": {
            "Verify identity": {
              "Type": "Task",
              "Resource": "arn:aws:states:::lambda:invoke",
              "Parameters": {
                "Payload.$": "$",
                "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:check-
identity:$LATEST"
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Verify address",
          "States": {
            "Verify address": {
              "Type": "Task",
              "Resource": "arn:aws:states:::lambda:invoke",
              "Parameters": {
                "Payload.$": "$",
                "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:check-
address:$LATEST"
              },
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

次の入力を使用してこのステートマシンを実行すると、検証を実行する Lambda 関数は検証が必要なデータのみを入力として想定するため、実行は失敗します。そのため、適切な JsonPath を使用して検証する情報を含むノードを指定する必要があります。

```
{
```

```
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    },
    "interests": [
      {
        "category": "home",
        "type": "own",
        "yearBuilt": 2004
      },
      {
        "category": "boat",
        "type": "snowmobile",
        "yearBuilt": 2020
      },
      {
        "category": "auto",
        "type": "RV",
        "yearBuilt": 2015
      },
    ]
  }
}
```

check-identity Lambda 関数が使用する必要があるノードを指定するには、以下のように InputPath フィールドを使用します。

```
"InputPath": "$.data.identity"
```

加えて、*check-address* Lambda 関数が使用する必要があるノードを指定するには、以下のように InputPath フィールドを使用します。

```
"InputPath": "$.data.address"
```

ここで、検証結果を元のステートマシン入力内に保存する場合は、以下のように ResultPath フィールドを使用します。

```
"ResultPath": "$.results"
```

ただし、ID と検証の結果のみが必要で、元の入力を破棄する場合は、以下のように OutputPath フィールドを使用します。

```
"OutputPath": "$.results"
```

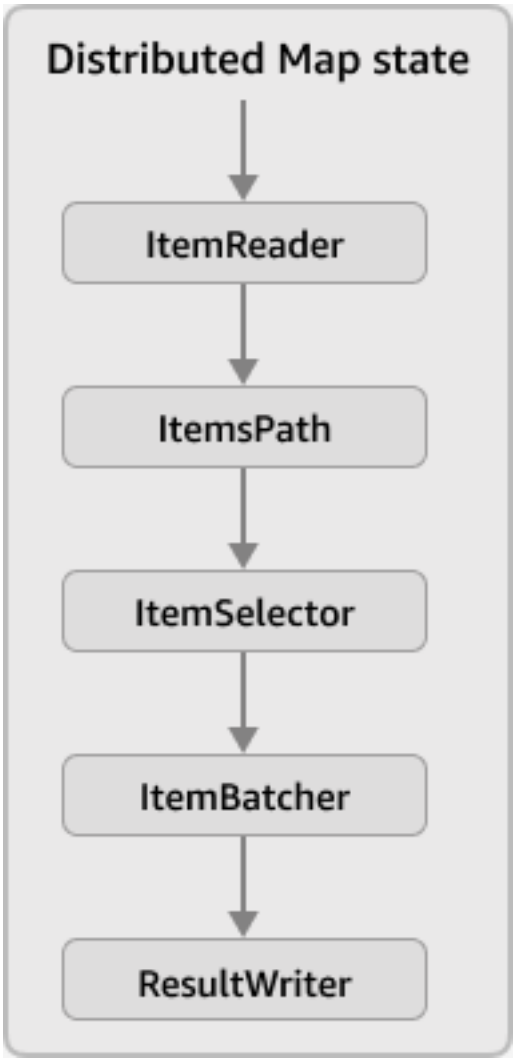
詳細については「Step Functions の入出力処理」を参照してください。

マップステートの入力および出力フィールド

Map 状態は、JSON 配列、Amazon S3 オブジェクトのリスト、Amazon S3 バケット内の CSV ファイルの行など、データセット内の項目のコレクションを同時に反復処理します。コレクション内の項目ごとに一連のステップを反復します。これらのフィールドを使用して、Map 状態が受け取る入力と生成される出力を設定できます。Step Functions は、以下のリストと図に示されている順序で、分散マップ状態の各フィールドを適用します。

> **Note**
>
> ユースケースによっては、これらのフィールドをすべて適用する必要がない場合があります。

1. ItemReader
2. ItemsPath
3. ItemSelector
4. ItemBatcher
5. ResultWriter



> **Note**
>
> これらの Map 状態入力フィールドと出力フィールドは、現在 Step Functions コンソールのデータフローシミュレーターでは使用できません。

ItemReader

`ItemReader` フィールドは JSON オブジェクトで、データセットとその場所を指定します。分散マップ状態はこのデータセットを入力として使用します。次の例は、データセットが Amazon S3 バケットに保存されている CSV ファイルである場合の `ItemReader` フィールドの構文を示しています。

```
"ItemReader": {
```

```
    "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "FIRST_ROW"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
        "Bucket": "myBucket",
        "Key": "csvDataset/ratings.csv"
    }
}
```

> **Tip**
>
> Workflow Studio では、データセットとその場所を [アイテムソース] フィールドで指定します。

目次

- ItemReader フィールドの内容
- データセットの例
- データセットの IAM ポリシー

ItemReader フィールドの内容

データセットによって、`ItemReader` フィールドの内容は異なります。例えば、データセットがワークフローの前のステップから渡された JSON 配列である場合、`ItemReader` フィールドは省略されます。データセットが Amazon S3 データソースである場合、このフィールドには次のサブフィールドが含まれます。

`ReaderConfig`

以下の詳細を指定する JSON オブジェクト。

- `InputType`

  CSV ファイル、オブジェクト、JSON ファイル、Amazon S3 インベントリリストなど、Amazon S3 データソースのタイプを指定します。Workflow Studio では、[項目ソース] フィールドの下にある [Amazon S3 項目ソース] ドロップダウンリストから入力タイプを選択できます。

- `CSVHeaderLocation`

> **Note**
>
> このフィールドは、CSV ファイルをデータセットとして使用する場合にのみ指定する必要があります。

以下の値のいずれかを受け入れ、列ヘッダーの位置を指定します。

> **Important**
>
> 現在、Step Functions は最大 10 KB の CSV ヘッダーをサポートしています。

- `FIRST_ROW` - ファイルの最初の行がヘッダーの場合は、このオプションを使用します。
- `GIVEN` - このオプションを使用して、ステートマシン定義内のヘッダーを指定します。例えば、CSV ファイルに次のデータが含まれている場合を考えてみましょう。

```
1,307,3.5,1256677221
1,481,3.5,1256677456
1,1091,1.5,1256677471
...
```

次の JSON 配列を CSV ヘッダーとして指定します。

```
"ItemReader": {
    "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "GIVEN",
        "CSVHeaders": [
            "userId",
            "movieId",
            "rating",
            "timestamp"
        ]
    }
}
```

> **Tip**
>
> Workflow Studio では、このオプションは [項目ソース] フィールドの [追加設定] にあります。

- `MaxItems`

  `Map` 状態に渡されるデータ項目の数を制限します。例えば、1000 行の CSV ファイルを用意し、100 という制限を指定したとします。そうすると、インタープリタは 100 行のみを `Map` 状態に渡します。`Map` 状態は項目をヘッダー行の後から順番に処理します。

  デフォルトでは、`Map` 状態は指定されたデータセット内のすべての項目を反復処理します。

  > **Note**
  >
  > 現在、100,000,000 件の制限まで指定できます。分散マップ状態は、この制限を超える項目の読み取りを停止します。

  > **Tip**
  >
  > Workflow Studio では、このオプションは [項目ソース] フィールドの [追加設定] にあります。

  または、分散マップ状態の入力に既存のキーと値のペアへの参照パスを指定することもできます。このパスは正の整数に変換する必要があります。`MaxItemsPath` サブフィールドで参照パスを指定します。

  > **Important**
  >
  > `MaxItems` または `MaxItemsPath` サブフィールドのいずれかを指定できますが、両方を指定することはできません。

`Resource`

指定されたデータセットに応じて、Step Functions が呼び出す必要がある Amazon S3 API アクション。

`Parameters`

データセットが保存されている Amazon S3 バケットの名前とオブジェクトキーを指定する JSON オブジェクト。

> **Important**
>
> Amazon S3 バケットがステートマシンと同じ AWS アカウント と AWS リージョン にあることを確認してください。

データセットの例

データセットとして以下のいずれかのオプションを指定できます。

- 前のステップの JSON 配列
- Amazon S3 オブジェクトのリスト
- Amazon S3 バケット内の JSON ファイル
- Amazon S3 バケット内の CSV ファイル
- Amazon S3 インベントリリスト

> **Important**
>
> Step Functions には、使用する Amazon S3 データセットにアクセスするための適切な許可が必要です。データセットの IAM ポリシーの詳細については、「データセットの IAM ポリシー」を参照してください。

前のステップの JSON 配列

分散マップ状態は、ワークフローの前のステップから渡された JSON 入力を受け入れることができます。この入力は配列であるか、特定のノード内に配列を含んでいる必要があります。配列を含むノードを選択するには、`ItemsPath` フィールドを使用できます。

配列内の個々の項目を処理するために、分散マップ状態は配列項目ごとに子ワークフロー実行を開始します。以下のタブには、`Map` 状態に渡される入力と、それに対応する子ワークフロー実行への入力の例が表示されます。

> **Note**
>
> Step Functions は、データセットが前のステップの JSON 配列の場合、`ItemReader` フィールドを省略します。

Input passed to the Map state

次の 3 つの項目からなる JSON 配列を考えてみましょう。

```
"facts": [
    {
        "verdict": "true",
        "statement_date": "6/11/2008",
        "statement_source": "speech"
    },
    {
        "verdict": "false",
        "statement_date": "6/7/2022",
        "statement_source": "television"
    },
    {
        "verdict": "mostly-true",
        "statement_date": "5/18/2016",
        "statement_source": "news"
    }
]
```

Input passed to a child workflow execution

分散マップ状態は、3 つの子ワークフローの実行を開始します。実行するたびに、配列項目を入力として受け取ります。次の例は、子ワークフロー実行が受け取る入力を示しています。

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 オブジェクトの例

分散マップ状態では、Amazon S3 バケットに格納されているオブジェクトを反復できます。ワークフローの実行が `Map` 状態に達すると、Step Functions は ListObjectsV2 API アクションを呼び出し、Amazon S3 オブジェクトメタデータの配列を返します。この配列の各項目には、バケットに保存されているデータの [ETag] や [Key] などのデータが含まれます。

配列内の個々の項目を処理するために、分散マップ状態は子ワークフロー実行を開始します。例えば、Amazon S3 バケットに 100 個のイメージが含まれているとします。次に、`ListObjectsV2` API アクションを呼び出した後に返される配列には 100 個の項目が含まれます。分散マップ状態は 100 個の子ワークフロー実行を開始して各配列項目を処理します。

> **Note**
>
> - 現在、Step Functions には、Amazon S3 コンソールを使用して特定の Amazon S3 バケットに作成する各フォルダの項目も含まれています。その結果、分散マップ状態によって開始される、余分の子ワークフロー実行が生じます。フォルダに余分な子ワークフロー実行が作成されないようにするため、AWS CLI を使用してフォルダを作成することをお勧めします。詳細については、「AWS Command Line Interface ユーザーガイド」の「高レベルな S3 コマンド」を参照してください。
> - Step Functions には、使用する Amazon S3 データセットにアクセスするための適切な許可が必要です。データセットの IAM ポリシーの詳細については、「データセットの IAM ポリシー」を参照してください。

以下のタブは、このデータセットの子ワークフロー実行に渡される `ItemReader` フィールド構文と入力の例を示しています。

ItemReader syntax

この例では、イメージ、JSON ファイル、オブジェクトを含むデータを、`myBucket` という名前の Amazon S3 バケットの `processData` というプレフィックス内に整理しました。

```
"ItemReader": {
    "Resource": "arn:aws:states:::s3:listObjectsV2",
    "Parameters": {
        "Bucket": "myBucket",
        "Prefix": "processData"
    }
```

```
}
```

Input passed to a child workflow execution

分散マップ状態は、Amazon S3 バケットに存在する項目数と同じ数の子ワークフローの実行を開始します。次の例は、子ワークフロー実行が受け取る入力を示しています。

```
{
  "Etag": "\"05704fbdccb224cb01c59005bebbad28\"",
  "Key": "processData/images/n02085620_1073.jpg",
  "LastModified": 1668699881,
  "Size": 34910,
  "StorageClass": "STANDARD"
}
```

Amazon S3 バケット内の JSON ファイル

分散マップ状態は、Amazon S3 バケットにデータセットとして保存されている JSON ファイルを受け入れることができます。JSON ファイルには配列が含まれている必要があります。

ワークフローの実行が Map 状態に達すると、Step Functions は GetObject API アクションを呼び出して、指定された JSON ファイルを取得します。その後、Map 状態は配列内の各項目を反復処理し、項目ごとに子ワークフローの実行を開始します。例えば、JSON ファイルに 1000 個の配列項目が含まれている場合、Map 状態は 1000 個の子ワークフロー実行を開始します。

> **Note**
>
> - 子ワークフロー実行を開始するために使用される実行入力は、256 KB を超えることができません。ただし、Step Functions では、オプションの `ItemSelector` フィールドを適用して項目のサイズを小さくすると、CSV または JSON ファイルから最大 8 MB の項目を読み取ることができます。
> - 現在、Step Functions は Amazon S3 インベントリレポートの個々のファイルに対して 10 GB の最大サイズをサポートしています。ただし、Step Functions は、個々のファイルが 10 GB 未満の場合、10 GB 以上を処理できます。
> - Step Functions には、使用する Amazon S3 データセットにアクセスするための適切な許可が必要です。データセットの IAM ポリシーの詳細については、「データセットの IAM ポリシー」を参照してください。

以下のタブは、このデータセットの子ワークフロー実行に渡される ItemReader フィールド構文と入力の例を示しています。

この例では、*factcheck.json* という JSON ファイルがあるとします。このファイルを Amazon S3 バケットに *jsonDataset* という名前のプレフィックスで保存しました。以下は、JSON データセットの例です。

```
[
  {
    "verdict": "true",
    "statement_date": "6/11/2008",
    "statement_source": "speech"
  },
  {
    "verdict": "false",
    "statement_date": "6/7/2022",
    "statement_source": "television"
  },
  {
    "verdict": "mostly-true",
    "statement_date": "5/18/2016",
    "statement_source": "news"
  },
  ...
]
```

ItemReader syntax

```
"ItemReader": {
    "Resource": "arn:aws:states:::s3:getObject",
    "ReaderConfig": {
        "InputType": "JSON"
    },
    "Parameters": {
        "Bucket": "myBucket",
        "Key": "jsonDataset/factcheck.json"
    }
}
```

Input to a child workflow execution

分散マップ状態は、Json ファイルに存在する配列項目数と同じ数の子ワークフローの実行を開始します。次の例は、子ワークフロー実行が受け取る入力を示しています。

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 バケット内の CSV ファイル

分散マップ状態は、Amazon S3 バケットにデータセットとして保存されている CSV ファイルを受け入れることができます。CSV ファイルをデータセットとして使用する場合は、CSV 列ヘッダーを指定する必要があります。CSV ヘッダーを指定する方法については、「ItemReader フィールドの内容」を参照してください。

CSV ファイルのデータを作成および管理するための標準化された形式がないため、Step Functions は次のルールに基づいて CSV ファイルを解析します。

- カンマ (,) は個々のフィールドを区切る区切り文字です。
- 改行は個々のレコードを分割する区切り文字です。
- フィールドは文字列として扱われます。データ型変換には、ItemSelector の `States.StringToJson` 組み込み関数を使用します。
- 文字列を二重引用符 (" ") で囲む必要はありません。ただし、二重引用符で囲まれた文字列には、区切り文字として機能しないカンマや改行を含めることができます。
- 二重引用符は、繰り返しにより工スケープできます。
- 行のフィールド数がヘッダーのフィールド数より少ない場合、Step Functions は欠落している値に空の文字列を提供します。
- 行のフィールド数がヘッダーのフィールド数よりも多い場合、Step Functions は追加のフィールドをスキップします。

Step Functions が CSV ファイルを解析する方法の詳細については、「Example of parsing an input CSV file」を参照してください。

ワークフローの実行が Map 状態に達すると、Step Functions は GetObject API アクションを呼び出して、指定された CSV ファイルを取得します。その後、Map 状態は CSV 内の各行を反復処理し、行ごとに子ワークフローの実行を開始します。例えば、100 行を含む CSV ファイルを入力として用意したとします。そうすると、インタープリタは各行を Map 状態に渡します。Map 状態は項目をヘッダー行の後から順次処理します。

> **Note**
>
> - 子ワークフロー実行を開始するために使用される実行入力は、256 KB を超えることができません。ただし、Step Functions では、オプションの ItemSelector フィールドを適用して項目のサイズを小さくすると、CSV または JSON ファイルから最大 8 MB の項目を読み取ることができます。
> - 現在、Step Functions は Amazon S3 インベントリレポートの個々のファイルに対して 10 GB の最大サイズをサポートしています。ただし、Step Functions は、個々のファイルが 10 GB 未満の場合、10 GB 以上を処理できます。
> - Step Functions には、使用する Amazon S3 データセットにアクセスするための適切な許可が必要です。データセットの IAM ポリシーの詳細については、「データセットの IAM ポリシー」を参照してください。

以下のタブは、このデータセットの子ワークフロー実行に渡される ItemReader フィールド構文と入力の例を示しています。

ItemReader syntax

例えば、*ratings.csv* という名前の CSV ファイルがあるとします。次に、このファイルを Amazon S3 バケットに *csvDataset* という名前のプレフィックスで保存しました。

```
{
  "ItemReader": {
    "ReaderConfig": {
      "InputType": "CSV",
      "CSVHeaderLocation": "FIRST_ROW"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
      "Bucket": "myBucket",
      "Key": "csvDataset/ratings.csv"
    }
```

```
  }
}
```

Input to a child workflow execution

分散マップ状態は、ヘッダー行を除いて (ファイル内にある場合)、CSV ファイルに存在する行数と同じ数の子ワークフローの実行を開始します。次の例は、子ワークフロー実行が受け取る入力を示しています。

```
{
  "rating": "3.5",
  "movieId": "307",
  "userId": "1",
  "timestamp": "1256677221"
}
```

S3 インベントリの例

分散マップ状態は、Amazon S3 バケットにデータセットとして保存されている Amazon S3 インベントリマニフェストファイルを受け入れることができます。

ワークフローの実行が `Map` 状態に達すると、Step Functions は GetObject API アクションを呼び出して、指定された Amazon S3 インベントリマニフェストファイルを取得します。その後、`Map` 状態はインベントリ内のオブジェクトを反復して、Amazon S3 インベントリオブジェクトメタデータの配列を返します。

> **Note**
>
> - 現在、Step Functions は Amazon S3 インベントリレポートの個々のファイルに対して 10 GB の最大サイズをサポートしています。ただし、Step Functions は、個々のファイルが 10 GB 未満の場合、10 GB 以上を処理できます。
> - Step Functions には、使用する Amazon S3 データセットにアクセスするための適切な許可が必要です。データセットの IAM ポリシーの詳細については、「データセットの IAM ポリシー」を参照してください。

CSV 形式のインベントリファイルの例を次に示します。このファイルには `imageDataset` および `csvDataset` という名前のオブジェクトが含まれます。これらのオブジェクトは、`sourceBucket` という名前の Amazon S3 バケットに保存されます。

```
"sourceBucket","csvDataset/","0","2022-11-16T00:27:19.000Z"
"sourceBucket","csvDataset/titles.csv","3399671","2022-11-16T00:29:32.000Z"
"sourceBucket","imageDataset/","0","2022-11-15T20:00:44.000Z"
"sourceBucket","imageDataset/n02085620_10074.jpg","27034","2022-11-15T20:02:16.000Z"
...
```

Important

現在、Step Functions はユーザー定義の Amazon S3 インベントリレポートをデータセットとしてサポートしていません。また、Amazon S3 インベントリレポートの出力形式が CSV であることを確認する必要があります。Amazon S3 インベントリとその設定方法の詳細については、「Amazon S3 ユーザーガイド」の「Amazon S3 インベントリ」を参照してください。

以下のインベントリマニフェストファイルの例は、インベントリオブジェクトメタデータの CSV ヘッダーを示しています。

```
{
  "sourceBucket" : "sourceBucket",
  "destinationBucket" : "arn:aws:s3:::inventory",
  "version" : "2016-11-30",
  "creationTimestamp" : "1668560400000",
  "fileFormat" : "CSV",
  "fileSchema" : "Bucket, Key, Size, LastModifiedDate",
  "files" : [ {
    "key" : "source-bucket/destination-prefix/
data/20e55de8-9c21-45d4-99b9-46c732000228.csv.gz",
    "size" : 7300,
    "MD5checksum" : "a7ff4a1d4164c3cd55851055ec8f6b20"
  } ]
}
```

以下のタブは、このデータセットの子ワークフロー実行に渡される ItemReader フィールド構文と入力の例を示しています。

ItemReader syntax

```
{
  "ItemReader": {
```

```
    "ReaderConfig": {
      "InputType": "MANIFEST"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
      "Bucket": "destinationBucket",
      "Key": "destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/
manifest.json"
    }
  }
}
```

Input to a child workflow execution

```
{
  "LastModifiedDate": "2022-11-16T00:29:32.000Z",
  "Bucket": "sourceBucket",
  "Size": "3399671",
  "Key": "csvDataset/titles.csv"
}
```

Amazon S3 インベントリレポートの設定時に選択したフィールドによっては、示されている例と `manifest.json` ファイルの内容が異なる場合があります。

データセットの IAM ポリシー

Step Functions コンソールでワークフローを作成すると、Step Functions はワークフロー定義内のリソースに基づいて IAM ポリシーを自動的に生成できます。これらのポリシーには、ステートマシンロールが分散マップ状態の `StartExecution` API アクションを呼び出すために必要な最小特権が含まれています。これらのポリシーには、Amazon S3 バケットやオブジェクト、Lambda 関数などの AWS リソースへのアクセスに必要な最小特権を持つ Step Functions も含まれています。IAM ポリシーには必要なアクセス許可のみを含めることを強くお勧めします。例えばワークフローに分散モードの `Map` 状態が含まれている場合は、ポリシーの範囲をデータセットを含む特定の Amazon S3 バケットとフォルダに限定します。

> **Important**
>
> 分散マップ状態の入力で、既存のキーと値のペアへの参照パスとともに、Amazon S3 バケットやオブジェクト、またはプレフィックスを指定する場合は、ワークフローの IAM ポリシー

を必ず更新してください。ポリシーの範囲は、ランタイムでパスから解釈されるバケット名とオブジェクト名に限定します。

次の IAM ポリシーの例では、ListObjectSv2 および GetObject API アクションを使用して Amazon S3 データセットにアクセスするのに必要な最小特権を付与しています。

Example データセットとしての Amazon S3 オブジェクトの IAM ポリシー

以下は、Amazon S3 バケットの *processImages* に配置した *myBucket* という名前のオブジェクトにアクセスするための最小特権を付与する IAM ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example データセットとしての CSV ファイルの IAM ポリシー

以下は、*ratings.csv* という名前の CSV ファイルにアクセスするための最小特権を付与する IAM ポリシーの例です。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example データセットとしての Amazon S3 インベントリの IAM ポリシー

以下は、Amazon S3 インベントリレポートにアクセスするための最小特権を付与する IAM ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-
MMZ/manifest.json",
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/data/*"
            ]
        }
    ]
}
```

ItemsPath

ItemsPath フィールドを使用して、Map 状態に提供される JSON 入力内の配列を選択します。Map 状態は配列内の各項目に対して一連のステップを反復します。デフォルトでは、Map 状態は ItemsPath を $ に設定し、これにより入力全体が選択されます。Map 状態への入力が JSON 配列の場合は、配列内の各項目に対して反復処理を実行し、その項目を入力として反復に渡します。

> **Note**
>
> 分散マップ状態で ItemsPath を使用できるのは、ワークフローの前のステートから渡された JSON 入力を使用する場合のみです。

ItemsPath フィールドを使用すると、反復処理に使用する JSON 配列を指す、入力内の位置を指定できます。ItemsPath の値は参照パスで、そのパスは JSON 配列を指している必要があります。例えば、次の例のように、Map 状態への入力に 2 つの配列が含まれている場合について考えてみます。

```
{
  "ThingsPiratesSay": [
    {
      "say": "Avast!"
    },
    {
      "say": "Yar!"
    },
    {
      "say": "Walk the Plank!"
    }
  ],
  "ThingsGiantsSay": [
    {
      "say": "Fee!"
    },
    {
      "say": "Fi!"
    },
    {
      "say": "Fo!"
    },
    {
      "say": "Fum!"
    }
  ]
}
```

この場合、ItemsPath で配列を選択して、Map 状態の反復処理に使用する配列を指定できます。次のステートマシン定義では ItemsPath を使用して入力内の ThingsPiratesSay 配列を指定し、ThingsPiratesSay 配列内の各項目に対して SayWord パス状態の反復処理を実行します。

```
{
  "StartAt": "PiratesSay",
  "States": {
    "PiratesSay": {
      "Type": "Map",
      "ItemsPath": "$.ThingsPiratesSay",
      "ItemProcessor": {
         "StartAt": "SayWord",
         "States": {
           "SayWord": {
             "Type": "Pass",
             "End": true
           }
         }
      },
      "End": true
    }
  }
}
```

入力を処理する場合、Map 状態は InputPath の後に ItemsPath を適用します。InputPath によって入力がフィルタリングされた後、状態への有効な入力に対して処理されます。

Map 状態の詳細については、以下を参照してください。

- マップステート
- マップステート処理モード
- インラインマップステートを使用したアクションの反復
- インライン Map 状態の入出力処理

ItemSelector

デフォルトでは、Map 状態の有効な入力は、未加工の状態入力に含まれる個々のデータ項目のセットです。ItemSelector フィールドでは、データ項目の値を Map 状態に渡す前にオーバーライドできます。値をオーバーライドするには、キーと値のペアのコレクションを含む有効な JSON 入力を

指定します。これらのペアは、ステートマシン定義で提供される静的な値でも、パスを使用して状態入力から選択された値でも、コンテキストオブジェクトからアクセスされる値でもかまいません。

パスまたはコンテキストオブジェクトを使用してキーと値のペアを指定する場合、キー名は `.$` で終わる必要があります。

> **Note**
>
> Map 状態内の Parameters フィールドは ItemSelector フィールドに置き換わります。Map 状態定義内の Parameters フィールドを使用してカスタム入力を作成する場合は、ItemSelector に置き換えることを強くお勧めします。

ItemSelector フィールドは、インラインマップ状態と分散マップ状態の両方で指定できます。

例えば、imageData ノード内の 3 つの項目の配列を含む次の JSON 入力があるとします。*Map* 状態反復ごとに、配列項目が入力として反復に渡されます。

```
[
  {
    "resize": "true",
    "format": "jpg"
  },
  {
    "resize": "false",
    "format": "png"
  },
  {
    "resize": "true",
    "format": "jpg"
  }
]
```

次の例に示すように、ItemSelector フィールドを使用して、元の入力をオーバーライドするカスタム JSON 入力を定義できます。次に、Step Functions はこのカスタム入力を各*Map* 状態反復に渡します。カスタム入力には、Map 状態のための size の静的値とコンテキストオブジェクトデータの値が含まれます。$$.Map.Item.Value コンテキストオブジェクトには、個々のデータ項目の値が含まれます。

```
{
```

```
  "ItemSelector": {
    "size": 10,
    "value.$": "$$.Map.Item.Value"
  }
}
```

次の例は、インラインマップステートの反復ごとに受け取る入力を示しています。

```
{
  "size": 10,
  "value": {
    "resize": "true",
    "format": "jpg"
  }
}
```

> **Tip**
>
> ItemSelector フィールドを使用する分散マップ状態の完全な例については、「分散マップ状態の使用開始」を参照してください。

ItemBatcher

ItemBatcher フィールドは、1 つの子ワークフロー実行内の項目グループを処理するよう指定する JSON オブジェクトです。大きな CSV ファイルや JSON 配列、または大量の Amazon S3 オブジェクトのセットを処理する場合は、バッチ処理を使用してください。

次の例は、ItemBatcher フィールドの構文を示しています。次の構文では、子ワークフローを実行するたびに処理すべき項目の最大数が 100 に設定されています。

```
{
  "ItemBatcher": {
    "MaxItemsPerBatch": 100
  }
}
```

デフォルトでは、データセット内の各項目が個々の子ワークフロー実行への入力として渡されます。例えば、次の配列を含む JSON ファイルを入力として指定するとします。

```
[
  {
    "verdict": "true",
    "statement_date": "6/11/2008",
    "statement_source": "speech"
  },
  {
    "verdict": "false",
    "statement_date": "6/7/2022",
    "statement_source": "television"
  },
  {
    "verdict": "true",
    "statement_date": "5/18/2016",
    "statement_source": "news"
  },
  ...
]
```

所定の入力があると、子ワークフロー実行はそれぞれ配列項目を入力として受け取ります。次の例は、子ワークフロー実行の入力を示しています。

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

処理ジョブのパフォーマンスとコストを最適化するには、アイテムの数とアイテムの処理時間のバランスを取るバッチサイズを選択してください。バッチ処理を使用する場合、Step Functions は [項目] 配列に項目を追加します。次に、その配列を各子ワークフロー実行に入力として渡します。次の例は、子ワークフロー実行の入力として渡された 2 つの項目のバッチを示しています。

```
{
  "Items": [
    {
      "verdict": "true",
      "statement_date": "6/11/2008",
      "statement_source": "speech"
    },
    {
```

```
      "verdict": "false",
      "statement_date": "6/7/2022",
      "statement_source": "television"
    }
  ]
}
```

> **Tip**
>
> ワークフローでの ItemBatcher フィールドの使用方法の詳細については、以下のチュートリアルとワークショップを試してみてください。
>
> - Lambda 関数内でデータのバッチ全体を処理する
> - 子ワークフロー実行内でバッチの項目を反復処する
> - AWS Step Functions ワークショップのモジュール 14 - データ処理の分散マップによる大規模並列化

目次

- 項目のバッチ処理を指定するフィールド

項目のバッチ処理を指定するフィールド

項目をバッチ処理するには、バッチ処理する項目の最大数、最大バッチサイズ、またはその両方を指定します。バッチ項目には、次のいずれかの値を指定する必要があります。

バッチあたりの最大項目数

子ワークフロー実行ごとに処理する項目の最大数を指定します。インタープリタは、Items 配列内でバッチ処理される項目数をこの値に制限します。バッチ番号とサイズの両方を指定すると、インタープリタは指定されたバッチサイズ制限を超えないようにバッチ内の項目数を減らします。

この値を指定せずに最大バッチサイズの値を指定した場合、Step Functions は、バイト単位の最大バッチサイズを超えずに、子ワークフロー実行ごとにできるだけ多くの項目を処理します。

例えば、1130 個のノードを含む入力 JSON ファイルを持つ実行ファイルを実行するとします。各バッチの最大項目値を 100 に指定すると、Step Functions は 12 のバッチを作成します。この

うち、11 個のバッチにはそれぞれ 100 個のアイテムが含まれ、12 番目のバッチには残りの 30 個のアイテムが含まれます。

または、分散マップ状態入力の既存のキーと値のペアへの参照パスとして、各バッチの最大項目数を指定することもできます。このパスは正の整数に変換する必要があります。

例として、次の入力があるとします。

```
{
  "maxBatchItems": 500
}
```

バッチ処理する項目の最大数は、次のように参照パスを使用して指定できます。

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxItemsPerBatchPath": "$.maxBatchItems"
    }
    ...
    ...
  }
}
```

> **Important**
>
> MaxItemsPerBatch または MaxItemsPerBatchPath サブフィールドのいずれかを指定できますが、両方を指定することはできません。

バッチあたりの最大 KB

バッチの最大値を 256 KB までのバイト値で指定します。最大バッチ番号とサイズの両方を指定すると、Step Functions は指定されたバッチサイズ制限を超えないようにバッチ内の項目数を減らします。

または、分散マップ状態入力の既存のキーと値のペアへの参照パスとして、最大バッチサイズを指定することもできます。このパスは正の整数に変換する必要があります。

> **Note**
>
> バッチ処理を使用し、最大バッチサイズを指定しない場合、インタープリタは子ワークフローを実行するたびに最大 256 KB まで処理できる項目数を処理します。

例として、次の入力があるとします。

```
{
  "batchSize": 131072
}
```

次のように、参照パスを使用して最大バッチサイズを指定できます。

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxInputBytesPerBatchPath": "$.batchSize"
    }
    ...
    ...
  }
}
```

> **Important**
>
> MaxInputBytesPerBatch または MaxInputBytesPerBatchPath サブフィールドのいずれかを指定できますが、両方を指定することはできません。

バッチ入力

オプションで、各子ワークフロー実行に渡される各バッチに含める固定 JSON 入力を指定することもできます。Step Functions は、この入力を個々の子ワークフロー実行の入力とマージします。例えば、項目の配列のファクトチェック日を次の固定入力にするとします。

```
"ItemBatcher": {
```

```
    "BatchInput": {
        "factCheck": "December 2022"
    }
}
```

子ワークフローの実行ごとに、入力として以下のものを受け取ります。

```
{
  "BatchInput": {
    "factCheck": "December 2022"
  },
  "Items": [
    {
      "verdict": "true",
      "statement_date": "6/11/2008",
      "statement_source": "speech"
    },
    {
      "verdict": "false",
      "statement_date": "6/7/2022",
      "statement_source": "television"
    },
    ...
  ]
}
```

ResultWriter

`ResultWriter` フィールドは、Step Functions が分散マップ状態により開始される子ワークフローの実行結果が書き込まれる Amazon S3 の場所を指定する、JSON オブジェクトです。デフォルトでは、Step Functions はこれらの結果をエクスポートしません。

> **Important**
>
> マップの実行結果をエクスポートするために使用する Amazon S3 バケットが、ステートマシンと同じ AWS アカウント と AWS リージョン にあることを確認してください。それ以外の場合は、`States.ResultWriterFailed` エラーが発生してステートマシンの実行が失敗します。

出力ペイロードサイズが 256 KB を超える場合は、結果を Amazon S3 バケットにエクスポートすると便利です。Step Functions は、実行の入出力、ARN、実行状態など、すべての子ワークフローの実行データを統合します。次に、指定した Amazon S3 の場所のそれぞれのファイルに、同じステータスの実行をエクスポートします。次の例は、子ワークフローの実行結果をエクスポートする場合の `ResultWriter` フィールドの構文を示しています。この例では、`csvProcessJobs` というプレフィックス内にある、`myOutputBucket` という名前のバケットに結果を保存します。

```
{
  "ResultWriter": {
    "Resource": "arn:aws:states:::s3:putObject",
    "Parameters": {
      "Bucket": "myOutputBucket",
      "Prefix": "csvProcessJobs"
    }
  }
}
```

> **Tip**
>
> Workflow Studio では、[マップステートの結果を Amazon S3 にエクスポート] を選択すると、子ワークフローの実行結果をエクスポートできます。次に、結果をエクスポートする Amazon S3 バケットの名前とプレフィックスを入力します。

Step Functions には、結果をエクスポートするバケットとフォルダにアクセスするための適切な権限が必要です。必要な IAM ポリシーの詳細については、「ResultWriter の IAM ポリシー」を参照してください。

子ワークフローの実行結果をエクスポートすると、分散マップ状態実行により、マップ実行 ARN と、Amazon S3 のエクスポート場所に関するデータが次の形式で返されます。

```
{
  "MapRunArn": "arn:aws:states:us-
east-2:123456789012:mapRun:csvProcess/Map:ad9b5f27-090b-3ac6-9beb-243cd77144a7",
  "ResultWriterDetails": {
    "Bucket": "myOutputBucket",
    "Key": "csvProcessJobs/ad9b5f27-090b-3ac6-9beb-243cd77144a7/manifest.json"
  }
}
```

Step Functions は、同じステータスの実行をそれぞれのファイルにエクスポートします。例えば、子ワークフローの実行で 500 件の成功、200 件の失敗の結果が得られた場合、Step Functions は指定された Amazon S3 の場所に成功と失敗の結果用の 2 つのファイルを作成します。この例では、成功結果ファイルには 500 件の成功結果が含まれ、失敗結果ファイルには 200 件の失敗結果が含まれています。

実行を試みると、Step Functions は実行出力に応じて、指定された Amazon S3 の場所に次のファイルを作成します。

- `manifest.json` - エクスポート場所、マップ実行 ARN、結果ファイルに関する情報などのマップ実行メタデータが含まれます。

  マップ実行を redriven した場合、`manifest.json` ファイルには、マップ実行のすべての試行で正常に実行されたすべての子ワークフローへの参照が含まれます。ただし、このファイルには、特定の redrive の実行に失敗した実行と保留中の実行への参照が含まれています。
- `SUCCEEDED_n.json` - 正常に実行されたすべての子ワークフローの統合データが含まれます。n はファイルのインデックス番号を表します。インデックス番号は 0 から始まります。例えば、`SUCCEEDED_1.json` です。
- `FAILED_n.json` - 失敗した、タイムアウトした、中止された子ワークフローのすべての実行に関する統合データが含まれます。このファイルを使用して、失敗した実行から回復します。n はファイルのインデックスを表します。インデックス番号は 0 から始まります。例えば、`FAILED_1.json` です。
- `PENDING_n.json` - マップ実行が失敗または中止されたために開始されなかったすべての子ワークフロー実行の統合データが含まれます。n はファイルのインデックスを表します。インデックス番号は 0 から始まります。例えば、`PENDING_1.json` です。

Step Functions は、最大 5 GB の個別の結果ファイルをサポートします。ファイルサイズが 5 GB を超える場合、Step Functions は残りの実行結果を書き込む別のファイルを作成し、ファイル名にインデックス番号を追加します。例えば、`Succeeded_0.json` ファイルのサイズが 5 GB を超える場合、Step Functions は残りの結果を記録する `Succeeded_1.json` ファイルを作成します。

子ワークフロー実行結果をエクスポートするように指定しなかった場合、ステートマシン実行は次の例のように子ワークフロー実行結果の配列を返します。

> **Note**
>
> 返される出力サイズが 256 KB を超えると、ステートマシンの実行は失敗し、States.DataLimitExceeded エラーを返します。

```
[
  {
    "statusCode": 200,
    "inputReceived": {
      "show_id": "s1",
      "release_year": "2020",
      "rating": "PG-13",
      "type": "Movie"
    }
  },
  {
    "statusCode": 200,
    "inputReceived": {
      "show_id": "s2",
      "release_year": "2021",
      "rating": "TV-MA",
      "type": "TV Show"
    }
  },
  ...
]
```

ResultWriter の IAM ポリシー

Step Functions コンソールでワークフローを作成すると、Step Functions はワークフロー定義内のリソースに基づいて IAM ポリシーを自動的に生成できます。これらのポリシーには、ステートマシンロールが分散マップ状態の StartExecution API アクションを呼び出すために必要な最小特権が含まれています。これらのポリシーには、Amazon S3 バケットやオブジェクト、Lambda 関数などの AWS リソースへのアクセスに必要な最小特権を持つ Step Functions も含まれています。IAM ポリシーには必要なアクセス許可のみを含めることを強くお勧めします。例えばワークフローに分散モードの Map 状態が含まれている場合は、ポリシーの範囲をデータセットを含む特定の Amazon S3 バケットとフォルダに限定します。

> **Important**
>
> 分散マップ状態の入力で、既存のキーと値のペアへの参照パスとともに、Amazon S3 バケットやオブジェクト、またはプレフィックスを指定する場合は、ワークフローの IAM ポリシーを必ず更新してください。ポリシーの範囲は、ランタイムでパスから解釈されるバケット名とオブジェクト名に限定します。

次の IAM ポリシーの例では `PutObject` API アクションを使用して、Amazon S3 バケット内にある *`csvJobs`* という名前のフォルダに、子ワークフローの実行結果を書き込むために必要な最小特権を付与しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::resultBucket/csvJobs/*"
            ]
        }
    ]
}
```

子ワークフローの実行結果を書き込む Amazon S3 バケットが AWS Key Management Service (AWS KMS) キーを使用して暗号化されている場合は、IAM ポリシーに必要な AWS KMS アクセス許可を含める必要があります。詳細については「AWS KMS key 暗号化された Amazon S3 バケットの IAM アクセス許可」を参照してください。

入力 CSV ファイルの解析

CSV ファイルのデータを作成および管理するための標準化された形式がないため、Step Functions は次のルールに基づいて CSV ファイルを解析します。

- カンマ (,) は個々のフィールドを区切る区切り文字です。

- 改行は個々のレコードを区切る区切り文字です。
- フィールドは文字列として扱われます。データ型の変換には、ItemSelector の States.StringToJson 組み込み関数を使用してください。
- 文字列を二重引用符 (" ") で囲む必要はありません。ただし、二重引用符で囲まれた文字列には、区切り文字として機能しないカンマや改行を含めることができます。
- 二重引用符は、繰り返しにより工スケープできます。
- 行のフィールド数がヘッダーのフィールド数より少ない場合、Step Functions は、欠落している値に空の文字列を提供します。
- 行のフィールド数がヘッダーのフィールド数よりも多い場合、Step Functions は追加のフィールドをスキップします。

Example 入力 CSV ファイルの解析での

例えば、1 行を入力として含む *myCSVInput.csv* という CSV ファイルを指定したとします。このファイルが、*my-bucket* という Amazon S3 バケットに保存されました。CSV ファイルは次のとおりです。

```
abc,123,"This string contains commas, a double quotation marks (""), and a newline (
)",{""MyKey"":""MyValue""},"[1,2,3]"
```

次のステートマシンはこの CSV ファイルを読み取り、ItemSelector を使用して一部のフィールドのデータ型を変換します。

```
{
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "Pass",
        "States": {
          "Pass": {
            "Type": "Pass",
            "End": true
          }
```

```
        }
      },
      "End": true,
      "Label": "Map",
      "MaxConcurrency": 1000,
      "ItemReader": {
        "Resource": "arn:aws:states:::s3:getObject",
        "ReaderConfig": {
          "InputType": "CSV",
          "CSVHeaderLocation": "GIVEN",
          "CSVHeaders": [
            "MyLetters",
            "MyNumbers",
            "MyString",
            "MyObject",
            "MyArray"
          ]
        },
        "Parameters": {
          "Bucket": "my-bucket",
          "Key": "myCSVInput.csv"
        }
      },
      "ItemSelector": {
        "MyLetters.$": "$$.Map.Item.Value.MyLetters",
        "MyNumbers.$": "States.StringToJson($$.Map.Item.Value.MyNumbers)",
        "MyString.$": "$$.Map.Item.Value.MyString",
        "MyObject.$": "States.StringToJson($$.Map.Item.Value.MyObject)",
        "MyArray.$": "States.StringToJson($$.Map.Item.Value.MyArray)"
      }
    }
  }
}
```

このステートマシンを実行すると、以下のような出力が生成されます。

```
[
  {
    "MyNumbers": 123,
    "MyObject": {
      "MyKey": "MyValue"
    },
    "MyString": "This string contains commas, a double quote (\"), and a newline (\n)",
```

```
    "MyLetters": "abc",
    "MyArray": [
      1,
      2,
      3
    ]
  }
]
```

コンテキストオブジェクト

コンテキストオブジェクトは、実行中に使用できる内部の JSON 構造であり、ステートマシンおよび実行に関する情報が含まれています。これにより、ワークフローが特定の実行に関する情報にアクセスできるようになります。コンテキストオブジェクトには、次のフィールドからアクセスできます。

- `InputPath`
- `OutputPath`
- `ItemsPath` (マップ状態)
- `Variable` (選択状態)
- `ResultSelector`
- `Parameters`
- 変数と変数の比較演算子

コンテキストオブジェクトの形式

コンテキストオブジェクトには、ステートマシン、筐体、実行、およびタスクに関する情報が含まれています。この JSON オブジェクトにはデータ型ごとのノードが含まれ、次の形式になります。

```
{
    "Execution": {
        "Id": "String",
        "Input": {},
        "Name": "String",
        "RoleArn": "String",
        "StartTime": "Format: ISO 8601",
        "RedriveCount": Number,
        "RedriveTime": "Format: ISO 8601"
    },
```

```
    "State": {
        "EnteredTime": "Format: ISO 8601",
        "Name": "String",
        "RetryCount": Number
    },
    "StateMachine": {
        "Id": "String",
        "Name": "String"
    },
    "Task": {
        "Token": "String"
    }
}
```

実行中、コンテキストオブジェクトの Parameters フィールドがアクセスされた場所から関連するデータで入力されます。Parameters フィールドがタスクの状態外にある場合、Task フィールドの値は null になります。

実行をまだ redriven していない場合、RedriveCount コンテキストオブジェクトの値は 0 です。さらに、RedriveTime コンテキストオブジェクトは、実行を redriven した場合にのみ使用できます。redriven a Map Run 済みの場合、RedriveTime コンテキストオブジェクトは Standard タイプの子ワークフローでのみ使用できます。Express タイプの子ワークフローを含む redriven マップ実行では、RedriveTime は使用できません。

稼働中の実行からのコンテンツには、次の形式による仕様が含まれています。

```
{
    "Execution": {
        "Id": "arn:aws:states:us-
east-1:123456789012:execution:stateMachineName:executionName",
        "Input": {
           "key": "value"
        },
        "Name": "executionName",
        "RoleArn": "arn:aws:iam::123456789012:role...",
        "StartTime": "2019-03-26T20:14:13.192Z"
    },
    "State": {
        "EnteredTime": "2019-03-26T20:14:13.192Z",
        "Name": "Test",
        "RetryCount": 3
    },
```

```
    "StateMachine": {
        "Id": "arn:aws:states:us-east-1:123456789012:stateMachine:stateMachineName",
        "Name": "stateMachineName"
    },
    "Task": {
        "Token": "h7XRiCdLtd/83p1E0dMccoxlzFhglsdkzpK9mBVKZsp7d9yrT1W"
    }
}
```

> **Note**
>
> Map 状態に関連するコンテキストオブジェクトデータについては、マップ状態のコンテキストオブジェクトデータ を参照してください。

コンテキストオブジェクトへのアクセス

コンテキストオブジェクトにアクセスするには、パスを使用して状態の入力を選択したときと同様に、.$ を末尾に追加したパラメータ名をまず指定します。次に、入力の代わりにコンテキストオブジェクトデータにアクセスするには、$$. をパスの先頭に追加します。これは、 パスを使用してコンテキストオブジェクト内のノードを選択する AWS Step Functions ように に指示します。

次の例は、実行 ID、名前、開始時間、redrive 回数などのコンテキストオブジェクトにアクセスする方法を示しています。

- 実行 ARN を取得してダウンストリームサービスに渡す
- パス状態で実行開始時間と名前にアクセス
- 実行の redrive 回数にアクセス

実行 ARN を取得してダウンストリームサービスに渡す

このタスク状態の例は、パスを使用して実行 Amazon リソースネーム (ARN) を取得して Amazon Simple Queue Service (Amazon SQS) メッセージに渡します。

```
{
  "Order Flight Ticket Queue": {
    "Type": "Task",
    "Resource": "arn:aws:states:::sqs:sendMessage",
    "Parameters": {
```

```
      "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/flight-purchase",
      "MessageBody": {
        "From": "YVR",
        "To": "SEA",
        "Execution.$": "$$.Execution.Id"
      }
    },
    "Next": "NEXT_STATE"
  }
}
```

統合されたサービスの呼び出しにタスクトークンを使用する方法の詳細については、タスクトークンのコールバックまで待機する を参照してください。

パス状態で実行開始時間と名前にアクセス

```
{
  "Comment": "Accessing context object in a state machine",
  "StartAt": "Get execution context data",
  "States": {
    "Get execution context data": {
      "Type": "Pass",
      "Parameters": {
        "startTime.$": "$$.Execution.StartTime",
        "execName.$": "$$.Execution.Name"
      },
      "ResultPath": "$.executionContext",
      "End": true
    }
  }
}
```

実行の redrive 回数にアクセス

以下のタスク状態定義の例は、実行の redrive 回数にアクセスする方法を示しています。

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "OutputPath": "$.Payload",
  "Parameters": {
    "Payload": {
      "Number.$": "$$.Execution.RedriveCount"
```

```
    },
    "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:functionName"
  },
  "End": true
}
```

マップ状態のコンテキストオブジェクトデータ

Map 状態を処理する際にコンテキストオブジェクトの 2 つの項目 Value と Index を追加で使用できます。各 Map 状態の反復において、Index には現在処理中の配列項目のインデックス番号が含まれますが、Value には処理中の配列項目が含まれます。Map 状態内では、コンテキストオブジェクトに以下のデータが含まれます。

```
"Map": {
   "Item": {
      "Index": Number,
      "Value": "String"
   }
}
```

これらは Map 状態でのみ使用でき、ItemSelector フィールドで指定することができます。

> **Note**
>
> コンテキストオブジェクトのパラメータは、ItemSelector セクションに含まれている状態内ではなく、メインの Map 状態の ItemProcessor ブロック内に定義する必要があります。

シンプルな Map 状態のステートマシンでは、コンテキストオブジェクトの情報を次のように挿入できます。

```
{
  "StartAt": "ExampleMapState",
  "States": {
    "ExampleMapState": {
      "Type": "Map",
      "ItemSelector": {
        "ContextIndex.$": "$$.Map.Item.Index",
        "ContextValue.$": "$$.Map.Item.Value"
```

```
      },
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "INLINE"
        },
        "StartAt": "TestPass",
        "States": {
          "TestPass": {
            "Type": "Pass",
            "End": true
          }
        }
      },
      "End": true
    }
  }
}
```

次の入力を使用して前のステートマシンを実行すると、 Index と Value が出力に挿入されます。

```
[
  {
    "who": "bob"
  },
  {
    "who": "meg"
  },
  {
    "who": "joe"
  }
]
```

実行の出力では、以下のように、3 回の各反復に対して Index および Value 項目の値が返されます。

```
[
  {
    "ContextIndex": 0,
    "ContextValue": {
      "who": "bob"
    }
  },
  {
```

```
    "ContextIndex": 1,
    "ContextValue": {
      "who": "meg"
    }
  },
  {

    "ContextIndex": 2,
    "ContextValue": {
      "who": "joe"
    }
  }
]
```

データフローシミュレータ

Step Functions コンソールでワークフローを設計、実装、およびデバッグできます。 JsonPath の入出力データ処理を使用して、ワークフロー内のデータの流れを制御することもできます。データフローシミュレーターを使用すると、ワークフロー内の タスク ステートがランタイムにデータを処理する順序をシミュレートできます。シミュレーターを使用すると、あるステートから別のステートに流れるデータがどのようにフィルタリングされて操作されるかを理解できます。Step Functions が JSON データのフローを処理および制御するために使用する以下の各フィールドをシミュレートします。

`InputPath`

全体の入力ペイロードのうち、タスクの入力としてどの部分を使用するかを選択します。このフィールドを指定すると、Step Functions は、最初にこのフィールドを適用します。

`#####`

タスクを起動する前に、入力をどのように表示するかを指定します。`Parameters` フィールドを使用すると、AWS Lambda 関数などのAWS のサービス の統合の入力として渡されるキーと値のペアのコレクションを作成できます。これらの値は、静的にすることも、ステート入力またはワークフローコンテキストオブジェクトから動的に選択することもできます。

`ResultSelector`

タスクの出力から何を選択するかを決定します。`ResultSelector` フィールドを使用すると、ステートの結果を置き換えるキーと値のペアのコレクションを作成し、そのコレクションを `ResultPath` に渡すことができます。

ResultPath

タスクの出力をどこに置くかを決めます。ResultPath を使用して、ステートの出力が、入力のコピー、生成された結果、またはその両方の組み合わせなのかを判断します。

OutputPath

次のステートに何を送るかを決めます。OutputPath を使用すると、不要な情報をフィルタリングして、JSON データのうちの必要な部分のみを渡すことができます。

このトピックの内容

- データフローシミュレータの使用
- データフローシミュレーターの使用に関する考慮事項

データフローシミュレータの使用

シミュレーターでは、入出力データ処理フィールドを適用する前と適用した後に、データをリアルタイムで並べて比較できます。シミュレーターを使用するには、JSON 入力を指定します。次にそれを、入力処理フィールドと出力処理フィールドのそれぞれで評価します。シミュレーターは JSON 入力を自動的に検証し、構文エラーがあれば強調表示します。

データフローシミュレーターを使用するには

次のステップでは、JSON 入力を指定して、InputPath フィールドと ##### フィールドを適用します。使用可能な他のフィールドを適用して、その出力を表示することもできます。

1. Step Functions コンソールを開きます。
2. ナビゲーションペインで、[データフローシミュレーター] を選択します。
3. [状態の入力]エリアで、事前に入力されたサンプル JSON データを次の JSON データに置き換えます。次に、[次へ] を選択します。

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
```

```
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    }
  }
}
```

4. [InputPath] には、入力 JSON データのアドレスノードを選択するために **$.data.address** を入力します。

   [InputPath の後の状態入力] ボックスには、次の結果が表示されます。

```
{
  "street": "123 Main St",
  "city": "Columbus",
  "state": "OH",
  "zip": "43219"
}
```

5. [次へ] をクリックします。
6. Parameters フィールドを適用して、結果の JSON データを文字列に変換します。データを変換するには、次のようにします。

   - [パラメータ] ボックスに、次のコードを入力して、addressString という文字列を作成します。

```
{
    "addressString.$": "States.Format('{}. {}, {} - {}', $.street, $.city,
 $.state, $.zip)"
  }
```

7. Parameters フィールドアプリケーションの結果が [パラメータの後にフィルタリングされた入力] ボックスに表示されます。

データフローシミュレーターの使用に関する考慮事項

データフローシミュレータを使用する前に、以下を含む制限事項を考慮してください。ただし、制限事項はこれらに限定されません。

- サポートされていないフィルター式

  フィルター式の動作は、シミュレーターと Step Functions サービスでは異なります。これには、次の構文を使用する式が含まれます。`[?(expression)]`。以下は、サポートされていない式の一覧です。これらの式を使用すると、評価後に期待どおりの結果が得られない場合があります。

  - `$..book[?(@.isInStock==true)]`
  - `$..book[?(@.price > 10.0)].title`
- 単一項目配列の、正しくない JsonPath 評価

  単一の配列項目を返す JsonPath 式でデータをフィルタリングすると、シミュレーターは配列のない項目を返します。例えば、`items` というデータ配列を考えてみましょう。

```
{
  "items": [
    {
      "name": "shoe",
      "color": "blue",
      "comment": "nice shoe"
    },
    {
      "name": "hat",
      "color": "red"
    },
    {
      "name": "shirt",
      "color": "yellow"
    }
  ]
}
```

  この `items` 配列が指定されている場合に InputPath フィールドに **`$..comment`** と入力すると、出力は次のようになります。

```
[
  "nice shoe"
]
```

  ただし、データフローシミュレーターは代わりに次の出力を返します。

```
"nice shoe"
```

複数の項目を含む配列の JsonPath 評価では、シミュレーターは期待された出力を返します。

バージョニングとエイリアスによる継続的デプロイの管理

Step Functions を使用して、ステートマシンのバージョニングとエイリアスを通じてワークフローの継続的なデプロイを管理できます。バージョンとは、実行可能なステートマシンの番号が付けられた変更不可能なスナップショットです。エイリアスとはステートマシンの最大 2 つのバージョンを示すポインタです。

ステートマシンの複数のバージョンを管理し、プロダクションワークフローへのデプロイを管理できます。エイリアスを使用すると、異なるワークフローバージョン間でトラフィックをルーティングし、それらのワークフローを実稼働環境に段階的にデプロイできます。

さらに、バージョンまたはエイリアスを使用してステートマシンの実行を開始できます。ステートマシンの実行を開始するときにバージョンまたはエイリアスを使用しない場合、Step Functions はステートマシン定義の最新リビジョンを使用します。

> **ステートマシンのリビジョン**
>
> ステートマシンは 1 つまたは複数のリビジョンを持つことができます。UpdateStateMachine API アクションを使用してステートマシンを更新すると、新しいステートマシンリビジョンが作成されます。リビジョンは、ステートマシンの定義と設定の不変で読み取り専用のスナップショットです。ステートマシンの実行をリビジョンから開始することはできず、リビジョンには ARN がありません。リビジョンには、共通の一意の識別子 (UUID) である、`revisionId` があります。

目次

- ステートマシンバージョン
- ステートマシンエイリアス
- バージョニングとエイリアスの認証
- バージョンまたはエイリアスへのステートマシン実行の関連付け
- エイリアスとバージョンのデプロイ例

- ステートマシンバージョンの段階的なデプロイの実行

ステートマシンバージョン

バージョンとは、ステートマシンの番号が付けられた変更不可能なスナップショットです。そのステートマシンに作成された最新リビジョンのバージョンを公開します。各バージョンには、固有の Amazon リソースネーム (ARN) があります。この ARN は、ステートマシン ARN とコロン (:) で区切られたバージョン番号の組み合わせです。次の例は、ステートマシンのバージョン ARN の形式を示しています。

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:1
```

ステートマシンのバージョンの使用を開始するには、最初のバージョンを公開する必要があります。バージョンを公開したら、バージョン ARN を使用して StartExecution API アクションを呼び出すことができます。バージョンを編集することはできませんが、ステートマシンを更新して新しいバージョンを公開することができます。ステートマシンの複数のバージョンを公開することもできます。

ステートマシンの新しいバージョンを公開すると、Step Functions によってバージョン番号が割り当てられます。バージョン番号は 1 から始まり、新しいバージョンごとに単純に増えていきます。バージョン番号は特定のステートマシンでは再利用されません。ステートマシンのバージョン 10 を削除して新しいバージョンを公開する場合、Step Functions はそれをバージョン 11 として公開します。

以下のプロパティはステートマシンのすべてのバージョンで同じになります。

- ステートマシンのすべてのバージョンは同じタイプ (標準または Express) を共有します。
- ステートマシンの名前または作成日をバージョン間で変更することはできません。
- タグはステートマシンにグローバルに適用されます。ステートマシンのタグは、 TagResourceおよび UntagResource API アクションを使用して管理できます。

ステートマシンには各バージョンと revision に含まれるプロパティも含まれますが、これらのプロパティは 2 つのバージョンまたはリビジョンによって異なる場合があります。これらのプロパティには、ステートマシン定義、IAM ロール、トレーシング設定、ログ記録設定が含まれます。

コンテンツ

- ステートマシンのバージョンの公開 (コンソール)
- Step Functions API 操作によるバージョンの管理
- コンソールからのステートマシンバージョンの実行

ステートマシンのバージョンの公開 (コンソール)

ステートマシンのバージョンは最大 1000 個公開できます。このソフトリミットの引き上げをリクエストするには、AWS Management Console の [サポートセンター] ページを使用してください。コンソールから、または DeleteStateMachineVersion API アクションを呼び出して、未使用のバージョンを手動で削除できます。

ステートマシンのバージョンを公開するには

1. Step Functions コンソールを開き、既存のステートマシンを選択します。
2. [ステートマシンの詳細] ページで、[編集] を選択します。
3. 必要に応じてステートマシン定義を編集し、[保存] を選択します。
4. [Publish version] を選択します。
5. (オプション) 表示されたダイアログボックスの[説明] フィールドに、ステートマシンのバージョンに関する簡単な説明を入力します。
6. [発行] を選択します。

> **Note**
>
> ステートマシンの新しいバージョンを公開すると、Step Functions によってバージョン番号が割り当てられます。バージョン番号は 1 から始まり、新しいバージョンごとに単純に増えていきます。バージョン番号は特定のステートマシンでは再利用されません。ステートマシンのバージョン 10 を削除して新しいバージョンを公開する場合、Step Functions はそれをバージョン 11 として公開します。

Step Functions API 操作によるバージョンの管理

Step Functions には、ステートマシンのバージョンを公開および管理するための次の API オペレーションが用意されています。

- PublishStateMachineVersion - ステートマシンrevisionの現在の からバージョンを発行します。
- UpdateStateMachine – ステートマシンを更新し、同じリクエストtrueで publishパラメータをに設定すると、新しいステートマシンバージョンを発行します。
- CreateStateMachine – publishパラメータを に設定した場合、ステートマシンの最初のリビジョンを発行しますtrue。
- ListStateMachineVersions – 指定されたステートマシン ARN のバージョンを一覧表示します。
- DescribeStateMachine – で指定されたバージョン ARN のステートマシンバージョンの詳細を返しますstateMachineArn。
- DeleteStateMachineVersion – ステートマシンのバージョンを削除します。

myStateMachine を使用して というステートマシンの現在のリビジョンから新しいバージョンを公開するには AWS Command Line Interface、 publish-state-machine-version コマンドを使用します。

```
aws stepfunctions publish-state-machine-version --state-machine-arn arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
```

レスポンスは stateMachineVersionArn を返します。例えば、前のコマンドは arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:1 のレスポンスを返します。

> **Note**
>
> ステートマシンの新しいバージョンを公開すると、Step Functions によってバージョン番号が割り当てられます。バージョン番号は 1 から始まり、新しいバージョンごとに単純に増えていきます。バージョン番号は特定のステートマシンでは再利用されません。ステートマシンのバージョン 10 を削除して新しいバージョンを公開する場合、Step Functions はそれをバージョン 11 として公開します。

コンソールからのステートマシンバージョンの実行

ステートマシンバージョンの使用を開始するには、まず現在のステートマシン revision からバージョンを公開する必要があります。バージョンを公開するには、Step Functions コンソールを使用するか、 PublishStateMachineVersion API アクションを呼び出します。という名前のオプションパラメータを使用して UpdateStateMachineAlias API アクションを呼び出しpublishて、ステートマシンを更新し、そのバージョンを公開することもできます。

コンソールを使用するか、 StartExecution API アクションを呼び出してバージョン ARN を指定することで、バージョンの実行を開始できます。エイリアスを使用してバージョンの実行を開始することもできます。エイリアスはルーティング設定に基づいて、トラフィックを特定のバージョンにルーティングします。

バージョンを使用せずにステートマシンの実行を開始すると、Step Functions はステートマシンの最新リビジョンを使用して実行します。Step Functions が実行をバージョンに関連付ける方法については、「バージョンまたはエイリアスへの実行の関連付け」を参照してください。

ステートマシンバージョンを使用して実行を開始するには

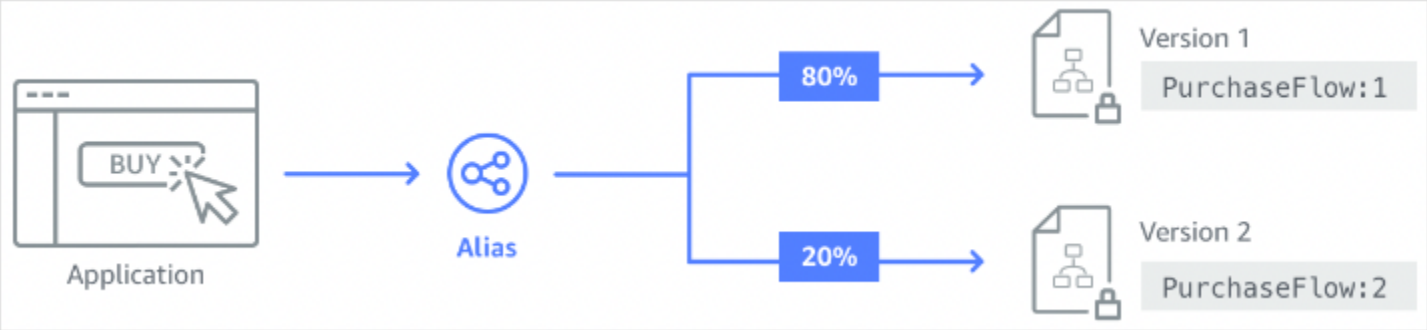
1. Step Functions コンソールを開き、1 つ以上のバージョンを公開している既存のステートマシンを選択します。バージョンを公開する方法については、「ステートマシンのバージョンの公開 (コンソール)」を参照してください。
2. [ステートマシンの詳細]」ページで [バージョン] タブを選択します。
3. [バージョン] セクションで、次の操作を行います。
   a. 実行の開始に使用するバージョンを選択します。
   b. [実行のスタート] を選択します。
4. (オプション) [実行を開始] ダイアログボックスで、実行の名前を入力します。
5. (オプション) 必要に応じて実行の入力を指定し、[実行を開始] を選択します。

ステートマシンエイリアス

エイリアスは、同じステートマシンの最大 2 つのバージョンを示すポインタです。ステートマシンには複数のエイリアスを作成できます。各エイリアスには固有の Amazon リソースネーム (ARN) があります。エイリアス ARN は、ステートマシンの ARN とエイリアス名をコロン (:) で区切ったものです。次の例は、ステートマシンのエイリアス ARN の形式を示しています。

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:aliasName
```

エイリアスを使用すると、2 つのステートマシンバージョンのいずれかの間でトラフィックをルーティングできます。1 つのバージョンを指すエイリアスを作成することもできます。エイリアスはステートマシンのバージョンのみを指すことができます。エイリアスを使って別のエイリアスを指すことはできません。ステートマシンの別のバージョンを指すよう、エイリアスを更新することもできます。



コンテンツ

- ステートマシンエイリアスの作成 (コンソール)
- Step Functions API 操作によるエイリアスの管理
- エイリアスのルーティング設定
- エイリアスを使用したステートマシンの実行 (コンソール)

ステートマシンエイリアスの作成 (コンソール)

Step Functions コンソールを使用するか、 CreateStateMachineAlias API アクションを呼び出して、ステートマシンごとに最大 100 個のエイリアスを作成できます。このソフトリミットの引き上げをリクエストするには、AWS Management Console の [サポートセンター] ページを使用してください。コンソールから、または DeleteStateMachineAlias API アクションを呼び出して、未使用のエイリアスを削除します。

ステートマシンエイリアスを作成するには

1. Step Functions コンソールを開き、既存のステートマシンを選択します。
2. [ステートマシンの詳細]」ページで [エイリアス] タブを選択します。
3. [新しいエイリアスの作成] を選択します。
4. [Create alias (エイリアスの作成)] ページで、以下の操作を行います。

a. [エイリアス名] を入力します。

b. (オプション) [Description (説明)] で、エイリアスの説明を入力します。

5. エイリアスのルーティングを設定するには、「エイリアスのルーティング設定」を参照してください。

6. [エイリアスの作成] を選択します。

Step Functions API 操作によるエイリアスの管理

Step Functions では、ステートマシンエイリアスの作成と管理やエイリアスに関する情報の取得に使用できる、以下の API 操作を使用できます。

- CreateStateMachineAlias – ステートマシンのエイリアスを作成します。
- DescribeStateMachineAlias – ステートマシンエイリアスに関する詳細を返します。
- ListStateMachineAliases – 指定されたステートマシン ARN のエイリアスを一覧表示します。
- UpdateStateMachineAlias - 既存のステートマシンエイリアスの設定を更新するには、その `description`または を変更します`routingConfiguration`。
- DeleteStateMachineAlias – ステートマシンのバージョンを削除します。

myStateMachine を使用して という名前のステートマシンのバージョン 1 *PROD*を指す という名前のエイリアスを作成するには AWS Command Line Interface、 `create-state-machine-alias` コマンドを使用します。

```
aws stepfunctions create-state-machine-alias --name PROD  --routing-
configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
```

エイリアスのルーティング設定

エイリアスを使用すると、ステートマシンの 2 つのバージョンの間で実行トラフィックをルーティングできます。例えば、ステートマシンの新しいバージョンを起動するとします。エイリアスにルーティングを設定すると、新しいバージョンのデプロイに伴うリスクを軽減できます。ルーティングを設定すると、ほとんどのトラフィックを以前のテスト済みバージョンのステートマシンに送信できます。これで、新しいバージョンが安全にロールフォワードされることが確認できるまで、新しいバージョンが受け取るトラフィックの割合が少なくなります。

ルーティング設定を定義するには、エイリアスが指すステートマシンの両方のバージョン必ず公開してください。エイリアスから実行を開始すると、Step Functions は、ルーティング設定で指定されたバージョンから実行するステートマシンのバージョンをランダムに選択します。この選択は、エイリアスルーティング設定の各バージョンに割り当てられたトラフィックの割合に基づいて行われます。

エイリアスのルーティング設定を構成するには

- [エイリアスの作成] ページの [ルーティングの設定]」で、次の操作を行います。

  a. [バージョン] では、エイリアスが指す最初のステートマシンのバージョンを選択します。

  b. [2 つのバージョン間でトラフィックを分割します] チェックボックスを選択します。

  > **Tip**
  >
  > 1 つのバージョンを指定するには、[2 つのバージョン間でトラフィックを分割します] チェックボックスをオフにします。

  c. [バージョン] では、エイリアスが指す必要のある 2 番目のバージョンを選択します。

  d. [トラフィックの割合] フィールドに、各バージョンにルーティングするトラフィックの割合を指定します。例えば、実行トラフィックの 60 パーセントを最初のバージョンにルーティングして、40 パーセントのトラフィックを 2 番目のバージョンにルーティングするには、**`60`** と **`40`** を入力します。

  トラフィックの割合の合計は 100% になる必要があります。

エイリアスを使用したステートマシンの実行 (コンソール)

ステートマシンの実行は、コンソールから、またはエイリアスの ARN を使用して StartExecution API アクションを呼び出すことによって、エイリアスを使用して開始できます。次に、Step Functions はエイリアスで指定されたバージョンを実行します。ステートマシンの実行を開始するときにバージョンまたはエイリアスを指定していない場合、Step Functions はデフォルトで最新のリビジョンを使用します。

エイリアスを使用してステートマシンの実行を開始するには

1. Step Functions コンソールを開き、エイリアスを作成した既存のステートマシンを選択します。エイリアスの作成の詳細については、「ステートマシンエイリアスの作成 (コンソール)」を参照してください。

2. [ステートマシンの詳細]」ページで [エイリアス] タブを選択します。
3. [エイリアス] セクションで、次の操作を行います。
   a. 実行の開始に使用するエイリアスを選択します。
   b. [実行のスタート] を選択します。
4. (オプション) [実行を開始] ダイアログボックスで、実行の名前を入力します。
5. 必要に応じて実行の入力を指定し、[実行を開始] を選択します。

バージョニングとエイリアスの認証

バージョンまたはエイリアスを使用して Step Functions API アクションを呼び出すには、適切なアクセス許可が必要です。バージョンまたはエイリアスに API アクションを呼び出す権限を与えるために、Step Functions はバージョン ARN またはエイリアス ARN の代わりにステートマシンの ARN を使用します。特定のバージョンまたはエイリアスの権限の範囲を制限することもできます。詳細については、「アクセス許可を制限する」を参照してください。

次の *myStateMachine* という名前の IAM ポリシー例を使用し CreateStateMachineAlias API アクションを呼び出して、ステートマシンエイリアスを作成できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "states:CreateStateMachineAlias",
      "Resource": "arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine"
    }
  ]
}
```

ステートマシンのバージョンまたはエイリアスを使用して、API アクションへのアクセスを許可または拒否するアクセス許可を設定する場合は、次の点を考慮してください。

- CreateStateMachine API アクションおよび UpdateStateMachine API アクションの `publish` パラメータを使用して新しいステートマシンのバージョンを発行する場合は、PublishStateMachineVersion API アクションの `ALLOW` のアクセス許可も必要です。
- DeleteStateMachine API アクションは、ステートマシンに関連付けられているすべてのバージョニングとエイリアスを削除します。

このトピックの内容

- バージョンまたはエイリアスのアクセス許可の範囲を制限する

バージョンまたはエイリアスのアクセス許可の範囲を制限する

修飾子を使用して、バージョンまたはエイリアスに必要なアクセス許可の権限をさらに制限できます。修飾子は、バージョン番号またはエイリアス名を指します。修飾子を使用して、ステートマシンを修飾します。次の例は、修飾子として PROD という名前のエイリアスを使用するステートマシン ARN です。

```
arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
```

修飾 ARN と非修飾 ARN の詳細については、「バージョンまたはエイリアスへの実行の関連付け」を参照してください。

IAM ポリシーの Condition ステートメントで、states:StateMachineQualifier という名前のオプションのコンテキストキーを使用して、アクセス許可の範囲を制限します。例えば、myStateMachine という名前のステートマシンの次の IAM ポリシーは、PROD またはバージョン 1 という名前のエイリアスを使って、DescribeStateMachine API アクションを呼び出すアクセスを拒否します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "states:DescribeStateMachine",
      "Resource": "arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "states:StateMachineQualifier": [
            "PROD",
            "1"
          ]
        }
      }
    }
  ]
}
```

次のリストでは、StateMachineQualifier コンテキストキーを使用して、アクセス許可の範囲を制限できる API アクションを指定しています。

- CreateStateMachineAlias
- DeleteStateMachineAlias
- DeleteStateMachineVersion
- DescribeStateMachine
- DescribeStateMachineAlias
- ListExecutions
- ListStateMachineAliases
- StartExecution
- StartSyncExecution
- UpdateStateMachineAlias

バージョンまたはエイリアスへのステートマシン実行の関連付け

Step Functions は、StartExecutionAPI アクションの呼び出しに使用する Amazon リソースネーム (ARN) に基づいて、実行をバージョンまたはエイリアスに関連付けます。Step Functions は、実行開始時にこのアクションを実行します。

修飾 ARN または非修飾 ARN を使用して、ステートマシンの実行を開始できます。

- 修飾 ARN - バージョン番号またはエイリアス名が末尾に付加されたステートマシン ARN を指します。

  修飾 ARN の以下の例は、myStateMachine というステートマシンのバージョン 3 を表しています。

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:3
  ```

  修飾 ARN の以下の例は、myStateMachine というステートマシンの PROD というエイリアスを表しています。

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
  ```

- 非修飾 ARN - バージョン番号やエイリアス名が末尾に付加されていないステートマシン ARN を指します。

```
arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
```

例えば、修飾 ARN がバージョン 3 を参照している場合、Step Functions は実行をこのバージョンに関連付けます。実行は、バージョン 3 を指すエイリアスには関連付けられません。

修飾 ARN がエイリアスを参照している場合、Step Functions は実行をそのエイリアスとエイリアスが指すバージョンに関連付けます。1 つの実行は、1 つのエイリアスにのみ関連付けることができます。

> **Note**
>
> 修飾 ARN を使用して実行を開始した場合、そのバージョンが同じステートマシン revision を使用していても、Step Functions はその実行をバージョンに関連付けません。例えば、バージョン 3 が最新リビジョンを使用しているが非修飾 ARN で実行を開始した場合、Step Functions はその実行をバージョン 3 に関連付けません。

このトピックの内容

- 特定のバージョンまたはエイリアスで開始された実行の表示

特定のバージョンまたはエイリアスで開始された実行の表示

Step Functions では、バージョンまたはエイリアスで開始された実行を以下の方法で表示できます。

API アクションの使用

DescribeExecution および ListExecutions API アクションを呼び出すことで、バージョンまたはエイリアスに関連付けられているすべての実行を表示できます。これらの API アクションは、実行を開始するために使用されたバージョンまたはエイリアスの ARN を返します。これらのアクションは、実行のステータスや ARN など、その他の詳細も返します。

ステートマシンのエイリアス ARN またはバージョン ARN を指定して、特定のエイリアスまたはバージョンに関連する実行を一覧表示することもできます。

次の ListExecutions API アクションのレスポンス例は、 という名前のステートマシンの実行を開始するために使用されるエイリアスの ARN を示しています*myFirstExecution*。

次のコードスニペットの*#########*テキストは、リソース固有の情報を表しています。

```
{
    "executions": [
        {
            "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:myStateMachine:myFirstExecution",
            "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine",
            "stateMachineAliasArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:PROD",
            "name": "myFirstExecution",
            "status": "SUCCEEDED",
            "startDate": "2023-04-20T23:07:09.477000+00:00",
            "stopDate": "2023-04-20T23:07:09.732000+00:00"
        }
    ]
}
```

Step Functions コンソールの使用

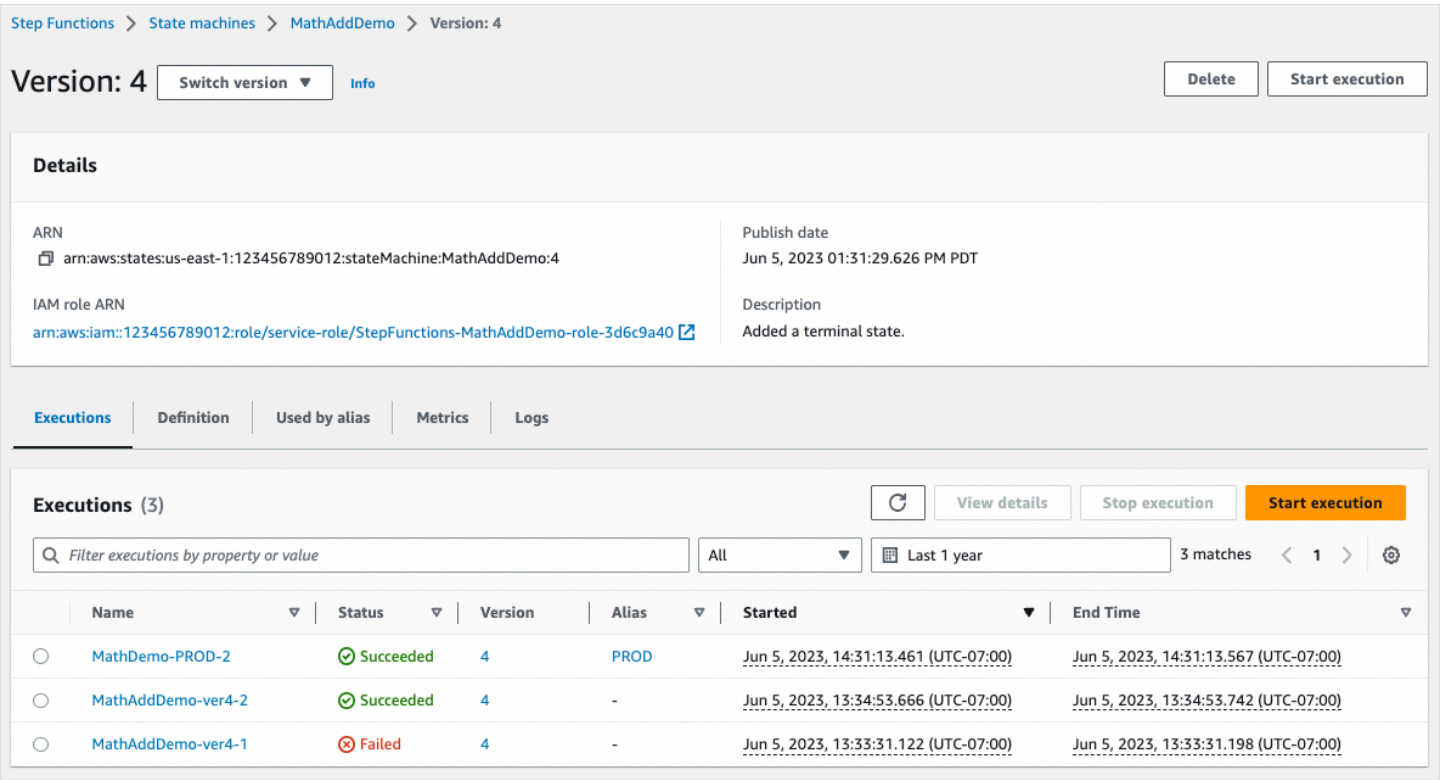
Step Functions コンソールから、バージョンまたはエイリアスによって開始された実行を確認することもできます。以下の手順は、特定のバージョンで開始された実行を表示する方法を示しています。

1. Step Functions コンソールを開き、バージョンを公開したかエイリアスを作成した既存のステートマシンを選択します。この例は、特定のステートマシンバージョンで開始された実行を表示する方法を示しています。
2. [バージョン] タブを選択し、[バージョン] リストからバージョンを選択します。

   > **Tip**
   >
   > プロパティまたは値ボックスでフィルタリングして、特定のバージョンを検索します。

3. [バージョンの詳細] ページには、選択したバージョンで開始された進行中および過去のステートマシン実行のリストが表示されます。

以下のイメージは、[バージョンの詳細] コンソールページを示しています。このページには、*MathAddDemo* というステートマシンのバージョン 4 によって開始された実行が一覧表示されます。このリストには、*PROD* というエイリアスによって開始された実行も表示されます。このエイリアスは実行トラフィックをバージョン 4 にルーティングしました。



CloudWatch メトリクスの使用

Qualified ARN で開始するステートマシンの実行ごとに、Step Functions は、現在出力されているメトリクスと同じ名前と値を持つ追加のメトリクスを出力します。これらの追加メトリクスには、実行を開始するバージョン識別子とエイリアス名のそれぞれのディメンションが含まれます。これらのメトリクスを使用すると、ステートマシンの実行をバージョンレベルでモニタリングし、ロールバックシナリオがいつ必要になるかを判断できます。これらのメトリクスに基づいて Amazon CloudWatch アラームを作成することもできます。

Step Functions は、エイリアスまたはバージョンで開始した実行について、以下のメトリクスを出力します。

- `ExecutionTime`
- `ExecutionsAborted`
- `ExecutionsFailed`
- `ExecutionsStarted`
- `ExecutionsSucceeded`
- `ExecutionsTimedOut`

バージョン ARN で実行を開始した場合、Step Functions は、StateMachineArn を含むメトリクスと、Version ディメンションを含む 2 つ目のメトリクスを公開します。

エイリアス ARN を使用して実行を開始した場合、Step Functions は次のメトリクスを出力します。

- 非修飾 ARN とバージョンの 2 つのメトリクス。
- StateMachineArn と Alias のディメンションを含むメトリクス。

エイリアスとバージョンのデプロイ例

次の Canary デプロイ手法の例は、AWS Command Line Interface を使用して新しいステートマシンバージョンをデプロイする方法を示しています。この例では、作成したエイリアスは実行トラフィックの 20 パーセントを新しいバージョンにルーティングします。次に、残りの 80% を以前のバージョンにルーティングします。新しいステートマシンバージョンをデプロイし、エイリアスを使用して実行トラフィックをシフトするには、次の手順を実行します。

1. 現在のステートマシンリビジョンからバージョンを公開します。

   AWS CLI で publish-state-machine-version コマンドを使用して、*myStateMachine*: というステートマシンの現在のリビジョンからバージョンを公開します。

   ```
   aws stepfunctions publish-state-machine-version --state-machine-arn
    arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
   ```

   レスポンスでは、公開したバージョンの stateMachineVersionArn が返されます。例えば、arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:1 です。

2. ステートマシンバージョンを指すエイリアスを作成します。

   create-state-machine-alias コマンドを使用して、*myStateMachine* のバージョン 1 を指す *PROD* というエイリアスを作成します。

   ```
   aws stepfunctions create-state-machine-alias --name PROD --routing-
   configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
   east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
   ```

3. エイリアスによって開始された実行で、正しい公開バージョンが使用されていることを確認します。

start-execution コマンドにエイリアス **PROD** の ARN を指定して、*myStateMachine* の新しい実行を開始します。

```
aws stepfunctions start-execution
  --state-machine-arn arn:aws:states:us-
east-1:123456789012:stateMachineAlias:myStateMachine:PROD
  --input "{}"
```

StartExecution リクエストでステートマシン ARN を指定すると、実行を開始するためにエイリアスで指定されたバージョンではなく、ステートマシンの最新の revision が使用されます。

4. ステートマシン定義を更新し、新しいバージョンを公開します。

   myStateMachine を更新して新しいバージョンを公開します。これを行うには、update-state-machine コマンドのオプションの publish パラメータを使用します。

```
aws stepfunctions update-state-machine
    --state-machine-arn arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine
    --definition $UPDATED_STATE_MACHINE_DEFINITION
    --publish
```

   レスポンスでは、新しいバージョンの stateMachineVersionArn が返されます。例えば、arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:2 です。

5. 両方のバージョンを指すようエイリアスを更新し、エイリアスのルーティング設定を指定します。

   update-state-machine-alias コマンドを使用して、エイリアス PROD のルーティング設定を更新します。実行トラフィックの 80% がバージョン 1 に送られ、残りの 20% がバージョン 2 に送られるようエイリアスを設定します。

```
aws stepfunctions update-state-machine-alias --state-machine-alias-arn
 arn:aws:states:us-east-1:123456789012:stateMachineAlias:myStateMachine:PROD
 --routing-configuration "[{\"stateMachineVersionArn\":
\"arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:1\",
\"weight\":80}, {\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:2\",\"weight\":20}]"
```

6. バージョン 1 をバージョン 2 に置き換えます。

新しいステートマシンバージョンが正しく動作することを確認したら、新しいステートマシンバージョンをデプロイできます。そのためには、エイリアスを再度更新して、実行トラフィックの 100% を新しいバージョンに割り当てます。

update-state-machine-alias コマンドを使用して、バージョン 2 のエイリアス PROD のルーティング設定を 100% に設定します。

```
aws stepfunctions update-state-machine-alias --state-machine-alias-arn
 arn:aws:states:us-east-1:123456789012:stateMachineAlias:myStateMachine:PROD
 --routing-configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:2\",\"weight\":100}]"
```

Tip

バージョン 2 のデプロイをロールバックするには、エイリアスのルーティング設定を編集して、新しくデプロイされたバージョンにトラフィックの 100% を移行します。

```
aws stepfunctions update-state-machine-alias
  --state-machine-alias-arn arn:aws:states:us-
east-1:123456789012:stateMachineAlias:myStateMachine:PROD
  --routing-configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
```

バージョンとエイリアスを使用すると、他のタイプのデプロイを実行できます。例えば、ステートマシンの新しいバージョンのローリングデプロイを実行できます。そのためには、新しいバージョンを指すエイリアスのルーティング設定の加重割合を徐々に増やします。

バージョンとエイリアスを使ってブルー/グリーンデプロイを行うこともできます。そのためには、ステートマシンの現在のバージョン 1 を実行する green というエイリアスを作成します。次に、新しいバージョン(例: *2*)を実行する blue という別のエイリアスを作成します。新しいバージョンをテストするには、blue エイリアスに実行トラフィックを送信します。新しいバージョンが正しく動作することが確認できたら、新しいバージョンを指すよう green エイリアスを更新します。

ステートマシンバージョンの段階的なデプロイの実行

ローリングデプロイとは、アプリケーションの以前のバージョンを新しいバージョンのアプリケーションに徐々に置き換えるデプロイ戦略です。ステートマシンバージョンのローリングデプロイを行

うには、徐々に増加する実行トラフィックを新しいバージョンに送信します。トラフィックの量と増加率は、設定するパラメータです。

次のいずれかのオプションを使用して、バージョンのローリングデプロイを実行できます。

- Step Functions コンソール - 同じステートマシンの 2 つのバージョンを指すエイリアスを作成します。このエイリアスでは、2 つのバージョン間でトラフィックをシフトするようルーティングを設定します。コンソールを使用したバージョンのロールアウトに関する詳細については、「バージョン」および「エイリアス」を参照してください。
- AWS CLI と SDK のスクリプト - AWS CLI または AWS SDK を使用してシェルスクリプトを作成します。詳細については、AWS CLI および AWS SDK の使用に関する以下のセクションを参照してください。
- AWS CloudFormation テンプレート - `AWS::StepFunctions::StateMachineVersion` と `AWS::StepFunctions::StateMachineAlias` のリソースを使用して複数のステートマシンバージョンを公開し、そのうちの 1 つまたは 2 つのバージョンを指すエイリアスを作成します。

AWS CLI を使用してステートマシンの新しいバージョンをデプロイします。

このセクションのスクリプト例は、AWS CLI を使用してトラフィックを以前のステートマシンバージョンから新しいステートマシンバージョンに徐々に移行する方法を示しています。このサンプルスクリプトを使用することも、必要に応じて更新することもできます。

このスクリプトは、エイリアスを使用して新しいステートマシンバージョンをデプロイするための Canary デプロイを示しています。以下の手順は、このスクリプトが実行するタスクの概要を示しています。

1. `publish_revision` パラメータが true に設定されている場合は、最新の revision を、ステートマシンの次のバージョンとして公開します。デプロイが成功すると、このバージョンが新しいライブバージョンになります。

   `publish_revision` パラメータを false に設定すると、スクリプトは、最後に公開されたバージョンのステートマシンをデプロイします。
2. エイリアスがまだ存在しない場合は作成します。エイリアスが存在しない場合は、このエイリアスへのトラフィックの 100% を新しいバージョンにルーティングして、スクリプトを終了します。
3. エイリアスのルーティング設定を更新して、トラフィックのごく一部を以前のバージョンから新しいバージョンにシフトします。canary のこの割合は `canary_percentage` パラメータで設定します。

4. デフォルトでは、設定可能な CloudWatch アラームを 60 秒ごとにモニタリングします。これらのアラームのいずれかが発せられたら、トラフィックの 100% を以前のバージョンにルーティングし、すぐにデプロイをロールバックしてください。

   alarm_polling_interval で定義した時間間隔 (秒単位) が経過したら、引き続きアラームをモニタリングします。canary_interval_seconds で定義した時間間隔が経過するまでモニタリングを続けます。

5. canary_interval_seconds の間にアラームが鳴らなかった場合は、トラフィックの 100% を新しいバージョンにシフトします。

6. 新しいバージョンが正常にデプロイされたら、history_max パラメータで指定された番号より古いバージョンをすべて削除します。

```
#!/bin/bash
#
# AWS StepFunctions example showing how to create a canary deployment with a
# State Machine Alias and versions.
#
# Requirements: AWS CLI installed and credentials configured.
#
# A canary deployment deploys the new version alongside the old version, while
# routing only a small fraction of the overall traffic to the new version to
# see if there are any errors. Only once the new version has cleared a testing
# period will it start receiving 100% of traffic.
#
# For a Blue/Green or All at Once style deployment, you can set the
# canary_percentage to 100. The script will immediately shift 100% of traffic
# to the new version, but keep on monitoring the alarms (if any) during the
# canary_interval_seconds time interval. If any alarms raise during this period,
# the script will automatically rollback to the previous version.
#
# Step Functions allows you to keep a maximum of 1000 versions in version history
# for a state machine. This script has a version history deletion mechanism at
# the end, where it will delete any versions older than the limit specified.
#
# For a fuller example, that also demonstrates linear (or rolling) deployments,
# please see
# https://github.com/aws-samples/aws-stepfunctions-examples/blob/main/gradual-deploy/
sfndeploy.py

set -euo pipefail
```

```
# ********************************************************************************
# you can safely change the variables in this block to your values
state_machine_name="my-state-machine"
alias_name="alias-1"
region="us-east-1"

# array of cloudwatch alarms to poll during the test period.
# to disable alarm checking, set alarm_names=()
alarm_names=("alarm1" "alarm name with a space")

# true to publish the current revision as the next version before deploy.
# false to deploy the latest version from the state machine's version history.
publish_revision=true

# true to force routing configuration update even if the current routing
# for the alias does not have a 100% routing config.
# false will abandon deploy attempt if current routing config not 100% to a
# single version.
# Be careful when you combine this flag with publish_revision - if you just
# rerun the script you might deploy the newly published revision from the
# previous run.
force=false

# percentage of traffic to route to the new version during the test period
canary_percentage=10

# how many seconds the canary deployment lasts before full deploy to 100%
canary_interval_seconds=300

# how often to poll the alarms
alarm_polling_interval=60

# how many versions to keep in history. delete versions prior to this.
# set to 0 to disable old version history deletion.
history_max=0
# ********************************************************************************

########################################
# Update alias routing configuration.
#
# If you don't specify version 2 details, will only create 1 routing entry. In
# this case the routing entry weight must be 100.
#
```

```
# Globals:
#   alias_arn
# Arguments:
#   1. version 1 arn
#   2. version 1 weight
#   3. version 2 arn (optional)
#   4. version 2 weight (optional)
#######################################
function update_routing() {
  if [[ $# -eq 2 ]]; then
    local routing_config="[{\"stateMachineVersionArn\": \"$1\", \"weight\":$2}]"
  elif [[ $# -eq 4 ]]; then
    local routing_config="[{\"stateMachineVersionArn\": \"$1\", \"weight\":$2},
 {\"stateMachineVersionArn\": \"$3\", \"weight\":$4}]"
  else
    echo "You have to call update_routing with either 2 or 4 input arguments." >&2
    exit 1
  fi

  ${aws} update-state-machine-alias --state-machine-alias-arn ${alias_arn} --routing-
configuration "${routing_config}"
}

# ******************************************************************************
# pre-run validation
if [[ (("${#alarm_names[@]}" -gt 0)) ]]; then
  alarm_exists_count=$(aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}"
 --alarm-types "CompositeAlarm" "MetricAlarm" --query "length([MetricAlarms,
 CompositeAlarms][])" --output text)

  if [[ (("${#alarm_names[@]}" -ne "${alarm_exists_count}")) ]]; then
    echo All of the alarms to monitor do not exist in CloudWatch: $(IFS=,; echo
 "${alarm_names[*]}") >&2
    echo Only the following alarm names exist in CloudWatch:
    aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}" --alarm-types
 "CompositeAlarm" "MetricAlarm" --query "join(', ', [MetricAlarms, CompositeAlarms]
[].AlarmName)" --output text
    exit 1
  fi
fi

if [[ (("${history_max}" -gt 0)) && (("${history_max}" -lt 2)) ]]; then
  echo The minimum value for history_max is 2. This is the minimum number of older
 state machine versions to be able to rollback in the future. >&2
```

```
  exit 1
fi
# ****************************************************************************
# main block follows

account_id=$(aws sts get-caller-identity --query Account --output text)

sm_arn="arn:aws:states:${region}:${account_id}:stateMachine:${state_machine_name}"

# the aws command we'll be invoking a lot throughout.
aws="aws stepfunctions"

# promote the latest revision to the next version
if [[ "${publish_revision}" = true ]]; then
  new_version=$(${aws} publish-state-machine-version --state-machine-arn=$sm_arn --
query stateMachineVersionArn --output text)
  echo Published the current revision of state machine as the next version with arn:
 ${new_version}
else
  new_version=$(${aws} list-state-machine-versions --state-machine-arn ${sm_arn} --max-
results 1 --query "stateMachineVersions[0].stateMachineVersionArn" --output text)
  echo "Since publish_revision is false, using the latest version from the state
 machine's version history: ${new_version}"
fi

# find the alias if it exists
alias_arn_expected="${sm_arn}:${alias_name}"
alias_arn=$(${aws} list-state-machine-aliases --state-machine-arn
 ${sm_arn} --query "stateMachineAliases[?stateMachineAliasArn==\`
${alias_arn_expected}\`].stateMachineAliasArn" --output text)

if [[ "${alias_arn_expected}" == "${alias_arn}" ]]; then
  echo Found alias ${alias_arn}

  echo Current routing configuration is:
  ${aws} describe-state-machine-alias --state-machine-alias-arn "${alias_arn}" --query
 routingConfiguration
else
  echo Alias does not exist. Creating alias ${alias_arn_expected} and routing 100%
 traffic to new version ${new_version}

  ${aws} create-state-machine-alias --name "${alias_name}" --routing-configuration
 "[{\"stateMachineVersionArn\": \"${new_version}\", \"weight\":100}]"
```

```
  echo Done!
  exit 0
fi

# find the version to which the alias currently points (the current live version)
old_version=$(${aws} describe-state-machine-alias --state-machine-alias-arn $alias_arn
 --query "routingConfiguration[?weight==\`100\`].stateMachineVersionArn" --output text)

if [[ -z "${old_version}" ]]; then
  if [[ "${force}" = true ]]; then
    echo Force setting is true. Will force update to routing config for alias to point
 100% to new version.
    update_routing "${new_version}" 100

    echo Alias ${alias_arn} now pointing 100% to ${new_version}.
    echo Done!
    exit 0
  else
    echo Alias ${alias_arn} does not have a routing config entry with 100% of the
 traffic. This means there might be a deploy in progress, so not starting another
 deploy at this time. >&2
    exit 1
  fi
fi

if [[ "${old_version}" == "${new_version}" ]]; then
  echo The alias already points to this version. No update necessary.
  exit 0
fi

echo Switching ${canary_percentage}% to new version ${new_version}
(( old_weight = 100 - ${canary_percentage} ))
update_routing "${new_version}" ${canary_percentage} "${old_version}" ${old_weight}

echo New version receiving ${canary_percentage}% of traffic.
echo Old version ${old_version} is still receiving ${old_weight}%.

if [[ ${#alarm_names[@]} -eq 0 ]]; then
  echo No alarm_names set. Skipping cloudwatch monitoring.
  echo Will sleep for ${canary_interval_seconds} seconds before routing 100% to new
 version.
  sleep ${canary_interval_seconds}
  echo Canary period complete. Switching 100% of traffic to new version...
else
```

```
  echo Checking if alarms fire for the next ${canary_interval_seconds} seconds.

  (( total_wait = canary_interval_seconds + $(date +%s) ))

  now=$(date +%s)
  while [[ (($(now} -lt ${total_wait})) ]]; do
    alarm_result=$(aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}"
 --state-value ALARM --alarm-types "CompositeAlarm" "MetricAlarm" --query "join(', ',
 [MetricAlarms, CompositeAlarms][].AlarmName)" --output text)

    if [[ ! -z "${alarm_result}" ]]; then
      echo The following alarms are in ALARM state: ${alarm_result}. Rolling back
 deploy. >&2
      update_routing "${old_version}" 100

      echo Rolled back to ${old_version}
      exit 1
    fi

    echo Monitoring alarms...no alarms have triggered.
    sleep ${alarm_polling_interval}
    now=$(date +%s)
  done

  echo No alarms detected during canary period. Switching 100% of traffic to new
 version...
fi

update_routing "${new_version}" 100

echo Version ${new_version} is now receiving 100% of traffic.

if [[ (("${history_max}" -eq 0 ))]]; then
  echo Version History deletion is disabled. Remember to prune your history, the
 default limit is 1000 versions.
  echo Done!
  exit 0
fi

echo Keep the last ${history_max} versions. Deleting any versions older than that...

# the results are sorted in descending order of the version creation time
```

```
version_history=$(${aws} list-state-machine-versions --state-
machine-arn ${sm_arn} --max-results 1000 --query "join(\`\"\\n\"\`,
 stateMachineVersions[].stateMachineVersionArn)" --output text)

counter=0

while read line; do
  ((counter=${counter} + 1))

  if [[ (( ${counter} -gt ${history_max})) ]]; then
    echo Deleting old version ${line}
    ${aws} delete-state-machine-version --state-machine-version-arn ${line}
  fi
done <<< "${version_history}"

echo Done!
```

AWS SDK を使用してステートマシンの新しいバージョンをデプロイします。

aws-stepfunctions-examples にあるサンプルスクリプトは、AWS SDK for Python を使用して、トラフィックを以前のバージョンから新しいバージョンのステートマシンに徐々に移行する方法を示しています。このサンプルスクリプトを使用することも、必要に応じて更新することもできます。

このスクリプトは、以下のデプロイ戦略を示しています。

- Canary - トラフィックは 2 つの増分で移行されます。

  1 回目の増分では、トラフィックのごく一部、例えば 10% が新しいバージョンにシフトされます。2 回目の増分では、指定した秒単位の時間間隔が経過する前に、残りのトラフィックが新しいバージョンにシフトされます。残りのトラフィックの新しいバージョンへの切り替えは、指定された時間間隔内に CloudWatch アラームがオフになっていない場合にのみ行われます。
- リニアまたはローリング - トラフィックを同じ間隔で、各増分の間隔を同じ秒数空けて新しいバージョンにシフトします。

  例えば、増加率を **600** 秒のうちの `--interval` の **20** として指定すると、このデプロイでは、新しいバージョンがトラフィックの 100% を受信するまで、600 秒ごとに 20% ずつトラフィックが増加します。

  このデプロイでは、いずれかの CloudWatch アラームがオフになると、すぐに新しいバージョンがロールバックされます。

- All at Once または Blue/Green - トラフィックの 100% をただちに新しいバージョンに移行します。このデプロイでは新しいバージョンをモニタリングし、CloudWatch アラームがオフになっている場合は、以前のバージョンに自動的にロールバックします。

AWS CloudFormation を使用して、新しいステートマシンバージョンをデプロイします。

次の CloudFormation テンプレートの例では、*MyStateMachine* というステートマシンの 2 つのバージョンを公開しています。両方のバージョンを指す *PROD* というエイリアスを作成して、バージョン 2 をデプロイします。

この例では、このバージョンがトラフィックの 100% を受信するまで、5 分ごとにトラフィックの 10% がバージョン 2 に移行されます。この例は、CloudWatch アラームを設定する方法も示しています。設定したアラームのいずれかが ALARM 状態になると、デプロイは失敗し、すぐにロールバックされます。

```
MyStateMachine:
  Type: AWS::StepFunctions::StateMachine
  Properties:
    Type: STANDARD
    StateMachineName: MyStateMachine
    RoleArn: arn:aws:iam::123456789012:role/myIamRole
    Definition:
      StartAt: PassState
      States:
        PassState:
          Type: Pass
          Result: Result
          End: true

MyStateMachineVersionA:
  Type: AWS::StepFunctions::StateMachineVersion
  Properties:
    Description: Version 1
    StateMachineArn: !Ref MyStateMachine

MyStateMachineVersionB:
  Type: AWS::StepFunctions::StateMachineVersion
  Properties:
    Description: Version 2
    StateMachineArn: !Ref MyStateMachine

PROD:
```

```
  Type: AWS::StepFunctions::StateMachineAlias
  Properties:
    Name: PROD
    Description: The PROD state machine alias taking production traffic.
    DeploymentPreference:
      StateMachineVersionArn: !Ref MyStateMachineVersionB
      Type: LINEAR
      Percentage: 10
      Interval: 5
      Alarms:
        # A list of alarms that you want to monitor. If any of these alarms trigger,
rollback the deployment immediately by pointing 100 percent of traffic to the previous
version.
        - !Ref CloudWatchAlarm1
        - !Ref CloudWatchAlarm2
```

Step Functions で実行

ステートマシンの実行は、AWS Step Functions ステートマシンが実行されタスクを行うときに発生します。各 Step Functions ステートマシンは、複数の同時実行をすることができます。これは、Step Functions コンソールから開始したり、AWS SDK、 API アクション、または AWS Command Line Interface (AWS CLI) を使用して開始できます。実行は JSON 入力を受け取り JSON 出力を生成します。以下の方法で Step Functions の実行をスタートできます。

- StartExecution API アクションを呼び出します。
- Step Functions コンソール内の新しい実行をスタートします。
- Amazon EventBridge を使用し、イベントに応答して実行を開始します。
- Amazon EventBridge Scheduler を使用して、スケジュールに従い、ステートマシンの実行を開始します。
- Amazon API Gateway を使って、実行をスタートします。
- タスク状態からネストされたワークフロー実行を開始します。

Step Functions を使った作業の別の方法の詳細については、開発オプションを参照してください。

タスク状態からワークフロー実行を開始する

AWS Step Functions は、ステートマシンの Task 状態から直接ワークフロー実行を開始できます。これにより、ワークフローを小さなステートマシンに分割し、これらの他のステートマシンの実行を開始できます。これらの新しいワークフロー実行を開始することで、以下のことが可能になります。

- 高レベルのワークフローを、低レベルのタスク固有のワークフローから分離する。
- 別のステートマシンを複数回呼び出すことによって、要素が繰り返されるのを回避する。
- モジュール式の再利用可能なワークフローのライブラリを作成して、開発を迅速化する。
- 複雑さを軽減し、ステートマシンの編集とトラブルシューティングを容易にする。

Step Functions は、独自の API を統合サービスとして呼び出して、これらのワークフロー実行をスタートできます。Task 状態から StartExecution API アクションを呼び出し、必要なパラメータを渡すだけです。サービス統合パターンのいずれかを使用する Step Functions API を呼び出すことができます。

> **Tip**
>
> ネストされたワークフローの例を AWS アカウント にデプロイするには、「モジュール 13 - ネストされた Express ワークフロー」を参照してください。

ステートマシンの新しい実行を開始するには、次の例のような Task 状態を使用します。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Input":{
         "Comment":"Hello world!"
      },
   },
   "Retry":[
      {
        "ErrorEquals":[
            "StepFunctions.ExecutionLimitExceeded"
        ]
```

```
      }
   ],
   "End":true
}
```

この Task 状態は、HelloWorld ステートマシンの新しい実行を開始し、JSON コメントを入力として渡します。

> **Note**
>
> StartExecution API アクションクォータでは、開始できる実行の数を制限できます。StepFunctions.ExecutionLimitExceeded で Retry を使用して、実行が確実に開始されるようにします。以下を参照してください。
>
> - API アクションのスロットリングに関連するクォータ
> - Step Functions のエラー処理

ワークフロー実行の関連付け

開始されたワークフロー実行とそれを開始した実行を関連付けるには、コンテキストオブジェクトから実行入力に実行 ID を渡します。実行中の Task 状態から、コンテキストオブジェクトの ID にアクセスできます。パラメータ名に .$ を追加し、$$.Execution.Id を使用してコンテキストオブジェクトの ID を参照することで、実行 ID を渡します。

```
"AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
```

実行を開始するときに、AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID という特別なパラメータを使用できます。含まれている場合は、この関連付けは Step Functions コンソールの [Step details] (ステップ詳細) セクションでリンクを提供します。指定すると、ワークフローの実行を、実行の開始から、開始したワークフロー実行まで簡単にトレースできます。前の例を使用して、次のように実行 ID を HelloWorld ステートマシンの開始した実行に関連付けます。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
```

```
      "Input": {
        "Comment": "Hello world!",
        "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
       }
   },
   "End":true
}
```

詳細については、次を参照してください。

- 他の サービスでの使用
- サービス API にパラメータを渡す
- コンテキストオブジェクトへのアクセス
- AWS Step Functions

AWS Step Functions での Amazon EventBridge スケジューラの使用

Amazon EventBridge スケジューラはサーバーレススケジューラで、一元化されたマネージドサービスからタスクを作成、実行、管理できます。EventBridge スケジューラでは、繰り返しのパターンに cron やレート式を使ってスケジュールを作成したり、1回限りの呼び出しを設定したりできます。配信時間枠の柔軟な設定、再試行制限の定義、失敗した API 呼び出しの最大保持時間の設定を行うことができます。

例えば、EventBridge Scheduler を使用すると、セキュリティ関連のイベントが発生したときにスケジュールに従ってステートマシンの実行を開始したり、データ処理ジョブを自動化したりすることができます。

このページでは、EventBridge スケジューラを使用して Step Functions ステートマシンをスケジュールどおりに実行する方法を説明します。

トピック

- 実行ロールを設定する
- 新しいスケジュールを作成する
- 関連リソース

実行ロールを設定する

新しいスケジュールを作成する場合、EventBridge スケジューラにはユーザーに代わってターゲット API オペレーションを呼び出すアクセス許可が必要です。実行ロールを使用して、これらのアクセス許可を EventBridge スケジューラに付与します。スケジュールの実行ロールにアタッチするアクセス許可ポリシーによって、必要なアクセス許可が定義されます。これらのアクセス許可は、EventBridge スケジューラが呼び出すターゲット API によって異なります。

次の手順のように EventBridge スケジューラコンソールを使用してスケジュールを作成すると、EventBridge スケジューラは選択したターゲットに基づき実行ロールを自動的に設定します。EventBridge スケジューラ SDK、AWS CLI、または AWS CloudFormation のいずれかを使用してスケジュールを作成する場合、EventBridge スケジューラがターゲットを呼び出すために必要なアクセス許可を付与する既存の実行ロールが必要です。スケジュールに合わせて実行ロールを手動で設定する方法についての詳細は、「EventBridge スケジューラユーザーガイド」の「実行ロールを設定する」を参照してください。

新しいスケジュールを作成する

コンソールを使用してスケジュールを作成するには

1. Amazon EventBridge スケジューラコンソール (https://console.aws.amazon.com/scheduler/home) を開きます。
2. [スケジュール] ページで、[スケジュールを作成] を選択します。
3. [スケジュールの詳細を指定] ページの [スケジュールの名前と説明] セクションで、次を実行します。

   a. [スケジュール名] で、スケジュールの名前を入力します。例えば、**`MyTestSchedule`** です。

   b. (オプション) [説明] で、スケジュールの説明を入力します。例えば、**`My first schedule`** です。

   c. [スケジュールグループ] で、ドロップダウンリストからスケジュールグループを選択します。グループがない場合は、[デフォルト] を選択します。スケジュールグループを作成するには、[独自のスケジュールを作成] を選択します。

   スケジュールグループを使用して、スケジュールのグループにタグを追加します。

4. • スケジュールオプションを選択します。

頻度	手順	
[1 回限りのスケジュール] 1 回限りのスケジュールは、指定した日時に 1 回だけターゲットを呼び出します。	[日付と時刻] で、次を実行します。 • 有効な日付を YYYY/MM/DD　形式で入力します。 • タイムスタンプを 24 時間 (hh:mm) 形式で入力します。 • [タイムゾーン] で、タイムゾーンを選択します。	

頻度	手順	
[繰り返しのスケジュール] 繰り返しのスケジュールは、cron 式またはレート式を使用して指定したレートでターゲットを呼び出します。	a. [スケジュールの種類] では、次のいずれかを実行します。 • Cron 式を使用してスケジュールを定義するには、[cron ベースのスケジュール] を選択して Cron 式を入力します。 • Rate 式を使用してスケジュールを定義するには、[rate ベースのスケジュール] を選択して Rate 式を入力します。 cron およびレート式の詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「EventBridge スケジューラのスケジュールタイプ」を参照してください。 b. [フレックスタイムウィンドウ] で、[オフ] を選択してオプションをオフにするか、事前定義された時間枠のいずれかを選択します。例えば、[15 分] を選択し、1 時間に 1 回ターゲットを呼び出す繰り返しのスケジュー	

頻度	手順	
	ルを設定した場合、スケジュールは毎時の開始後 15 分以内に実行されます。	

5. (オプション) 前のステップで [定期的なスケジュール] を選択した場合は、[時間枠] セクションで次を実行します。

   a. [タイムゾーン] で、タイムゾーンを選択します。

   b. [開始日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。

   c. [終了日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。

6. [次へ] をクリックします。

7. [ターゲットを選択] ページで、EventBridge スケジューラが呼び出す AWS API オペレーションを選択します。

   a. [AWS Step Functions StartExecution] を選択します。

   b. [StartExecution] セクションで、ステートマシンを選択するか、[新しいステートマシンを作成] を選択します。

      現在、同期 Express ワークフローをスケジュールどおりに実行することはできません。

   c. 実行用の JSON ペイロードを入力します。ステートマシンが JSON ペイロードを必要としない場合でも、次の例に示すように、JSON 形式での入力を引き続き含める必要があります。

```
{
    "Comment": "sampleJSONData"
}
```

8. [次へ] をクリックします。

9. [Settings] (設定) ページで、以下の操作を行います。

   a. スケジュールをオンにするには、[スケジュールの状態] で [スケジュールを有効にする] をオンに切り替えます。

b. スケジュールの再試行ポリシーを設定するには、[再試行ポリシーとデッドレターキュー (DLQ)] で次を実行します。

- [再試行] を切り替えてオンにします。
- [イベントの最大有効期間] で、EventBridge スケジューラが未処理のイベントを保持しなければならない最大の [時間] と [分] を入力します。
- 最大 24 時間です。
- [最大再試行回数] で、ターゲットがエラーを返した場合に EventBridge スケジューラがスケジュールを再試行する最大回数を入力します。

  再試行の最大値は 185 です。

再試行ポリシーを使用すると、スケジュールがそのターゲットの呼び出しに失敗した場合、EventBridge スケジューラはスケジュールを再実行します。設定されている場合は、スケジュールの最大保持時間と再試行を設定する必要があります。

c. EventBridge スケジューラが未配信のイベントを保存する場所を選択します。

[デッドレターキュー (DLQ)] オプション	手順	
保存しない	[None] を選択します。	
スケジュールを作成しようとしている同じ AWS アカウント にイベントを保存する	a. [自分の AWS アカウント の Amazon SQS キューを DLQ として選択] を選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を選択します。	

[デッドレターキュー (DLQ)] オプション	手順
スケジュールを作成しようとしているのとは別の AWS アカウント にイベントを保存する	a. [他の AWS アカウント の Amazon SQS キューを DLQ として指定] を選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を入力します。

d. カスタマーマネージドキーを使用してターゲットの入力を暗号化するには、[暗号化] で [暗号化設定をカスタマイズする (高度)] を選択します。

   このオプションを選択した場合は、既存の KMS キー ARN を入力するか、[AWS KMS key を作成] を選択して AWS KMS コンソールに移動します。EventBridge スケジューラが保管中のデータを暗号化する方法の詳細については、「Amazon EventBridge スケジューラユーザーガイド」の「保管中の暗号化」を参照してください。

e. EventBridge スケジューラに新しい実行ロールを作成させるには、[このスケジュールの新しいロールを作成] を選択します。その後、[ロール名] で名前を入力します。このオプションを選択すると、EventBridge スケジューラは、テンプレート化されたターゲットに必要な許可をロールにアタッチします。

10. [次へ] をクリックします。
11. [スケジュールの確認と作成] ページで、スケジュールの詳細を確認します。各セクションで、そのステップに戻って詳細を編集するには、[編集] を選択します。
12. [スケジュールを作成] を選択します。

    [スケジュール] ページで、新規および既存のスケジュールのリストを表示できます。[ステータス] 列で、新しいスケジュールが [有効] になっていることを確認します。

EventBridge スケジューラがステートマシンを呼び出したことを確認するには、ステートマシンの Amazon CloudWatch Logs を確認してください。

関連リソース

EventBridge スケジューラに関する詳細については、次を参照してください。

- EventBridge スケジューラユーザーガイド
- EventBridge スキーマ API リファレンス
- EventBridge Scheduler Pricing

コンソールでの Standard ワークフローと Express ワークフローの実行

ステートマシンを作成するときは、[Standard] または [Express] のいずれかの [タイプ] を選択します。ステートマシンのデフォルトの [タイプ] は [Standard] です。[タイプ] が [Standard] のステートマシンは Standard ワークフローと呼ばれ、[タイプ] が [Express] のステートマシンは Express ワークフローと呼ばれます。

Standard ワークフローと Express ワークフローの両方で、Amazon ステートメント言語 を使用してステートマシンを定義します。ステートマシンの実行の動作は、選択した [タイプ] によって異なります。

> **Important**
>
> ステートマシンが作成された後は、選択した [タイプ] を変更することはできません。

Standard ワークフローと Express ワークフローの詳細については、「標準ワークフロー対 Express ワークフロー」を参照してください。

Standard ワークフロー実行の履歴は Step Functions に記録されますが、Express ワークフローの実行履歴は Step Functions に記録されません。Express ワークフロー実行の履歴を記録するには、Amazon にログを送信するように設定する必要があります CloudWatch。詳細については、「CloudWatch Logs を使用したログ記録」を参照してください。

Express ワークフローでログ記録を設定すると、Step Functions コンソールでその実行を確認できます。Express ワークフロー実行と Standard ワークフロー実行を表示するコンソール操作は似ていますが、以下の違いと制限があります。

> **Note**
>
> Express ワークフローの実行データは CloudWatch Logs Insights を使用して表示されるため、ログをスキャンすると料金が発生します。デフォルトでは、ロググループには過去 3 時間に完了した実行のみが一覧表示されます。より多くの実行イベントを含むより広い時間範

囲を指定すると、コストが増加します。詳細については、「 CloudWatch 料金表」ページの「ログ」タブにある「提供されるログ」および「」を参照してくださいCloudWatch Logs を使用したログ記録。

コンテンツ

- コンソールエクスペリエンスの違い
- Express ワークフロー実行の表示に関する考慮事項と制限

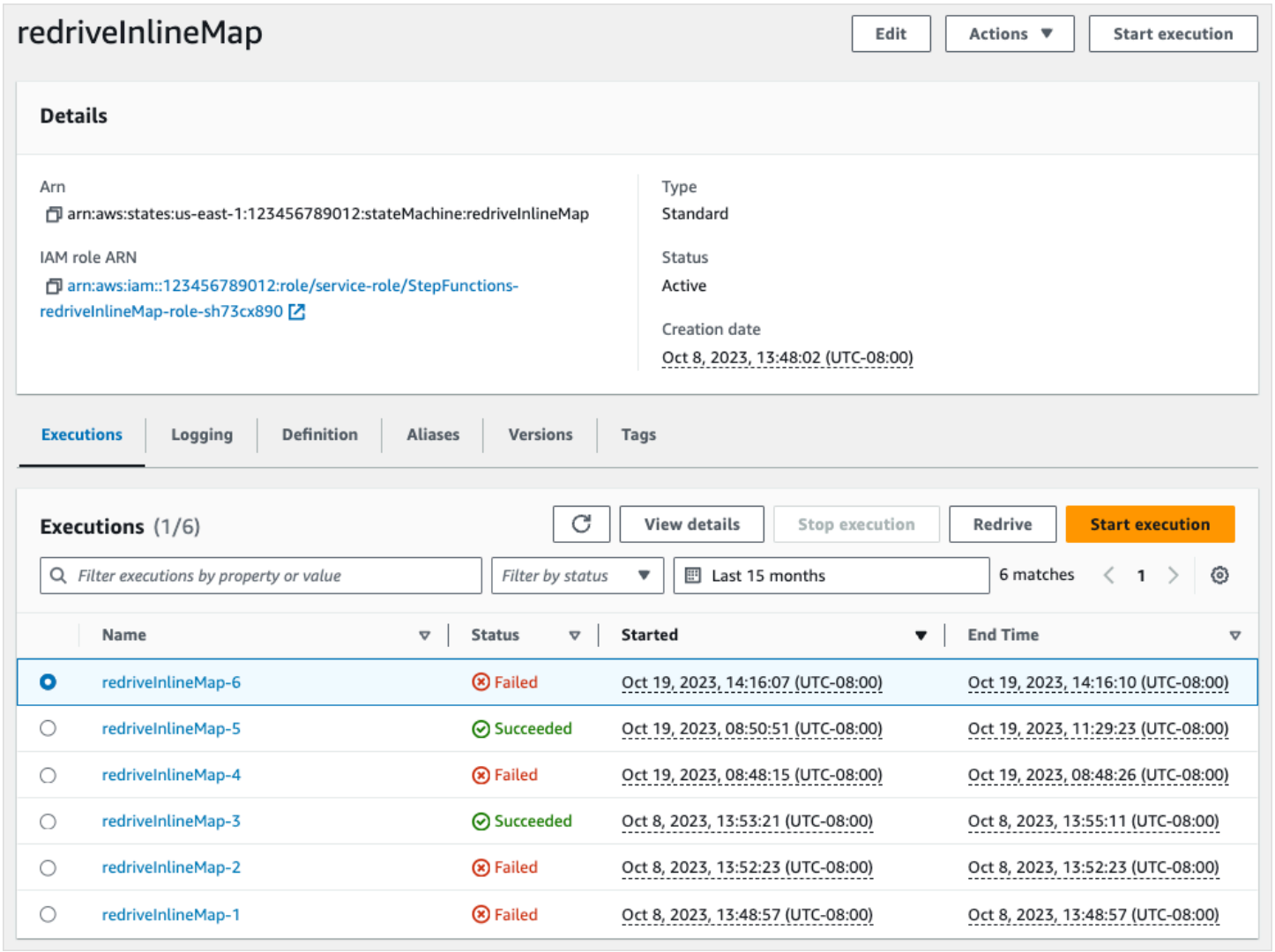
コンソールエクスペリエンスの違い

すべての Standard ワークフローと Express ワークフローでは、ステートマシンやその IAM ロール ARN などの詳細を Step Functions コンソールのステートマシンの詳細ページで確認できます。

ステートマシンの詳細ページの [実行] タブには、ステートマシンの実行履歴のリストも表示されます。[プロパティまたは値で実行をフィルタリングする] ボックスを使用して、選択したステートマシンの特定の実行、バージョン、またはエイリアスを検索します。[すべて] ドロップダウンを使用して、実行履歴をステータスでフィルタリングします。実行履歴を選択し、[詳細を表示] ボタンを選択して [実行の詳細] ページを開くこともできます。

Standard ワークフロー

Standard ワークフローの実行履歴は、過去 90 日間に完了した実行の履歴をいつでも確認できます。



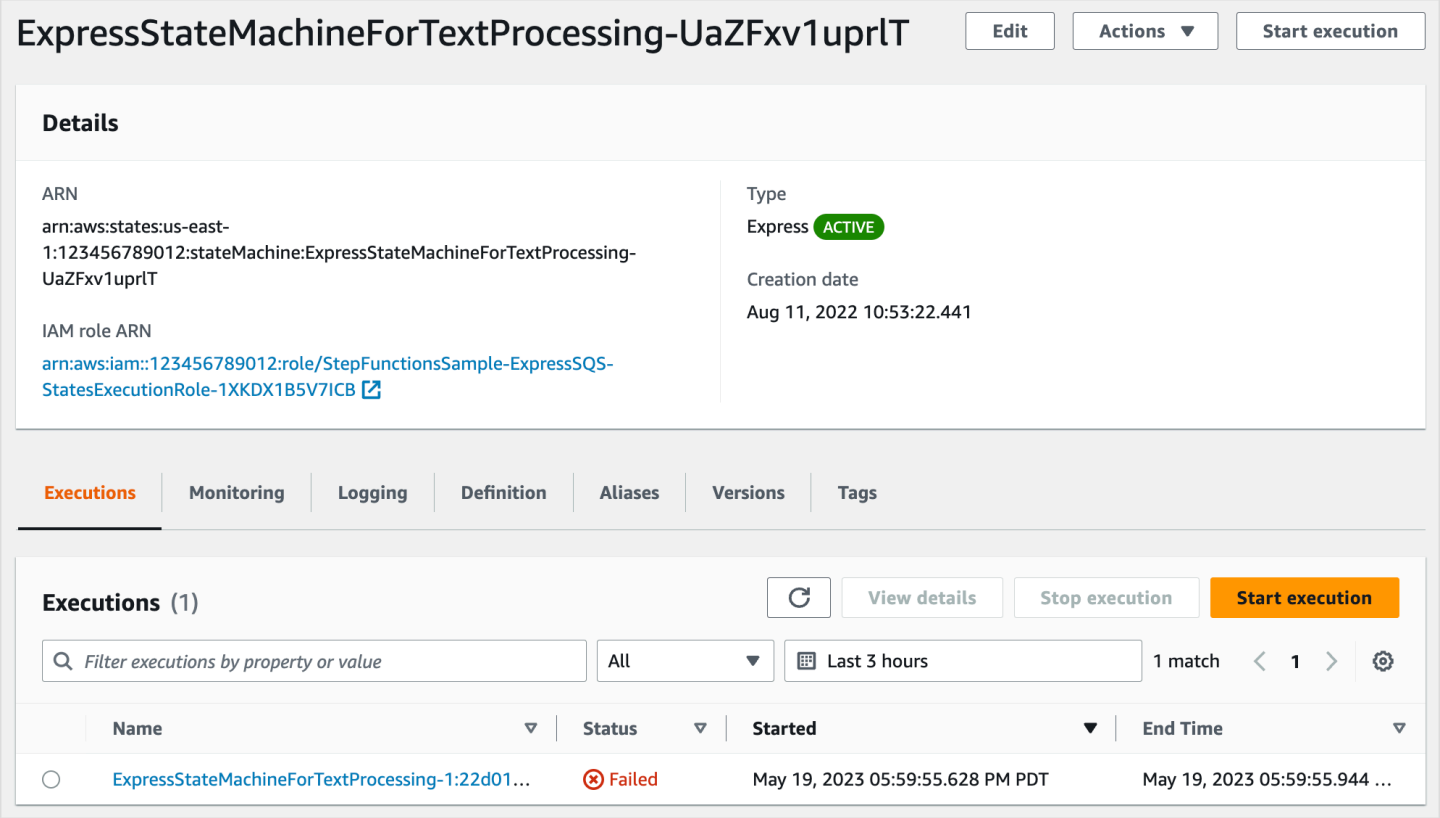
Express ワークフロー

Express ワークフローの実行履歴を表示するには、Step Functions コンソールで Logs CloudWatch ロググループを通じて収集されたログデータを取得します。

Express ワークフローの実行を表示するには、新しいコンソールエクスペリエンスを有効にする必要もあります。これを行うには、[実行] タブのバナー内に表示されている [有効化] ボタンを選択します。このボタンを選択すると、今後は表示されなくなります。

> **Tip**
>
> コンソールエクスペリエンスを有効にするか無効にするかを切り替えるには、[高速実行の履歴の有効化] トグルボタンを使用します。

過去 3 時間以内に完了した実行の履歴はデフォルトで表示されます。この時間範囲は調整することも、カスタム範囲を指定することもできます。より多くの実行イベントを含むより広い時間範囲を指定すると、ログをスキャンするコストが増加します。詳細については、「 料金表」ページの「 ログ」タブにある「提供されるログ」および「」を参照してくださいCloudWatch Logs を使用したログ記録。 CloudWatch



Express ワークフロー実行の表示に関する考慮事項と制限

Step Functions コンソールで Express ワークフローの実行を表示する場合、次の考慮事項と制限事項に留意してください。

- Express ワークフロー実行の詳細の可用性は Amazon CloudWatch Logs によって異なります
- Partial Express ワークフロー実行の詳細は、ログ記録レベルが ERROR または FATAL の場合に表示されます。
- 古い実行のステートマシン定義は、一度更新すると表示できなくなります。

Express ワークフロー実行の詳細の可用性は Amazon CloudWatch Logs によって異なります

> **Note**
>
> Express ワークフローの実行を表示する新しいコンソールエクスペリエンスを有効にしない場合、実行履歴とそれに対応する実行の詳細は Step Functions コンソールでは使用できません。新しいコンソールエクスペリエンスを有効にするには、[実行] タブのバナー内に表示されている [有効化] ボタンを選択します。

Express ワークフローの場合、実行履歴と詳細な実行情報は CloudWatch Logs Insights を通じて収集されます。この情報は、ステートマシンの作成時に指定した CloudWatch Logs ロググループに保持されます。ステートマシンの実行履歴は、Step Functions コンソールの [実行] タブに表示されます。ステートマシンの各実行に関する詳細情報は、選択した実行の [実行の詳細] ページに表示されます。

> **Warning**
>
> Express ワークフローの CloudWatch ログを削除すると、実行タブの下には表示されません。

すべての実行イベントタイプをログに記録するには、デフォルトのログレベル [すべて] を使用することをお勧めします。ステートマシンを編集するときに、必要に応じて既存のステートマシンのログレベルを更新できます。詳細については、CloudWatch Logs を使用したログ記録およびログレベルを参照してください。

Partial Express ワークフロー実行の詳細は、ログ記録レベルが ERROR または FATAL の場合に表示されます。

デフォルトでは、Express ワークフロー実行のログ記録レベルは [すべて] に設定されています。ログレベルを変更しても、完了した実行の実行履歴と実行の詳細には影響しません。ただし、新しい実行ではすべて、更新されたログレベルに基づいてログが出力されます。詳細については、「CloudWatch Logs を使用したログ記録」および「ログレベル」を参照してください。

例えば、ログレベルを [すべて] から [ERROR] または [FATAL] に変更した場合、Step Functions コンソールの [実行] タブには、失敗した実行のみが表示されます。[イベントビュー] タブでは、コンソールには失敗したステートマシンステップのイベント詳細のみが表示されます。

すべての実行イベントタイプをログに記録するには、デフォルトのログレベル [すべて] を使用することをお勧めします。ステートマシンを編集するときに、必要に応じて既存のステートマシンのログレベルを更新できます。

古い実行のステートマシン定義は、一度更新すると表示できなくなります。

Express ワークフローでは、過去の実行のステートマシン定義は保存されません。ステートマシン定義を変更した場合、最新の定義を使用した実行のステートマシン定義のみを表示できます。

例えば、ステートマシン定義から 1 つ以上のステップを削除すると、Step Functions は定義と以前の実行イベントとの不一致を検出します。以前の定義は Express ワークフローには保存されないため、Step Functions は以前のバージョンのステートマシン定義で実行された実行のステートマシン定義を表示できません。そのため、以前のバージョンのステートマシン定義で実行された実行には、[実行の入力と出力]、[定義]、[グラフビュー]、および [テーブルビュー] タブは使用できません。

Step Functions コンソールでの実行の表示とデバッグ

Step Functions コンソールの [実行の詳細] ページには、Standard ワークフローと Express ワークフローの過去および進行中のステートマシン実行に関する情報が表示されます。この情報はダッシュボード形式で表示されます。例えば、ステートマシンの Amazon States Language 定義、その実行ステータス、ARN、および状態遷移の総数を確認できます。ステートマシン内の個々のステートの実行の詳細も表示できます。

コンテンツ

- [実行の詳細] ページ - インターフェイスの概要
  - 実行の概要
  - エラーメッセージ
  - 表示モード
  - ステップの詳細
  - イベント
- チュートリアル: Step Functions コンソールを使用したステートマシン実行の調査
  - ステップ 1: 必要な Lambda 関数を作成およびテストする
  - ステップ 2: ステートマシンを作成および実行する
  - ステップ 3: ステートマシンの実行の詳細を表示する
  - ステップ 4: さまざまな表示モードを詳しく見る

[実行の詳細] ページ - インターフェイスの概要

[実行の詳細] ページでは、Standard ワークフローと Express ワークフローの両方で進行中および過去のステートマシン実行の詳細を確認できます。実行の開始時に実行 ID を指定した場合、このページにはその実行 ID が付けられます。それ以外の場合は、Step Functions が自動的に生成する固有の実行 ID でタイトルが付けられます。

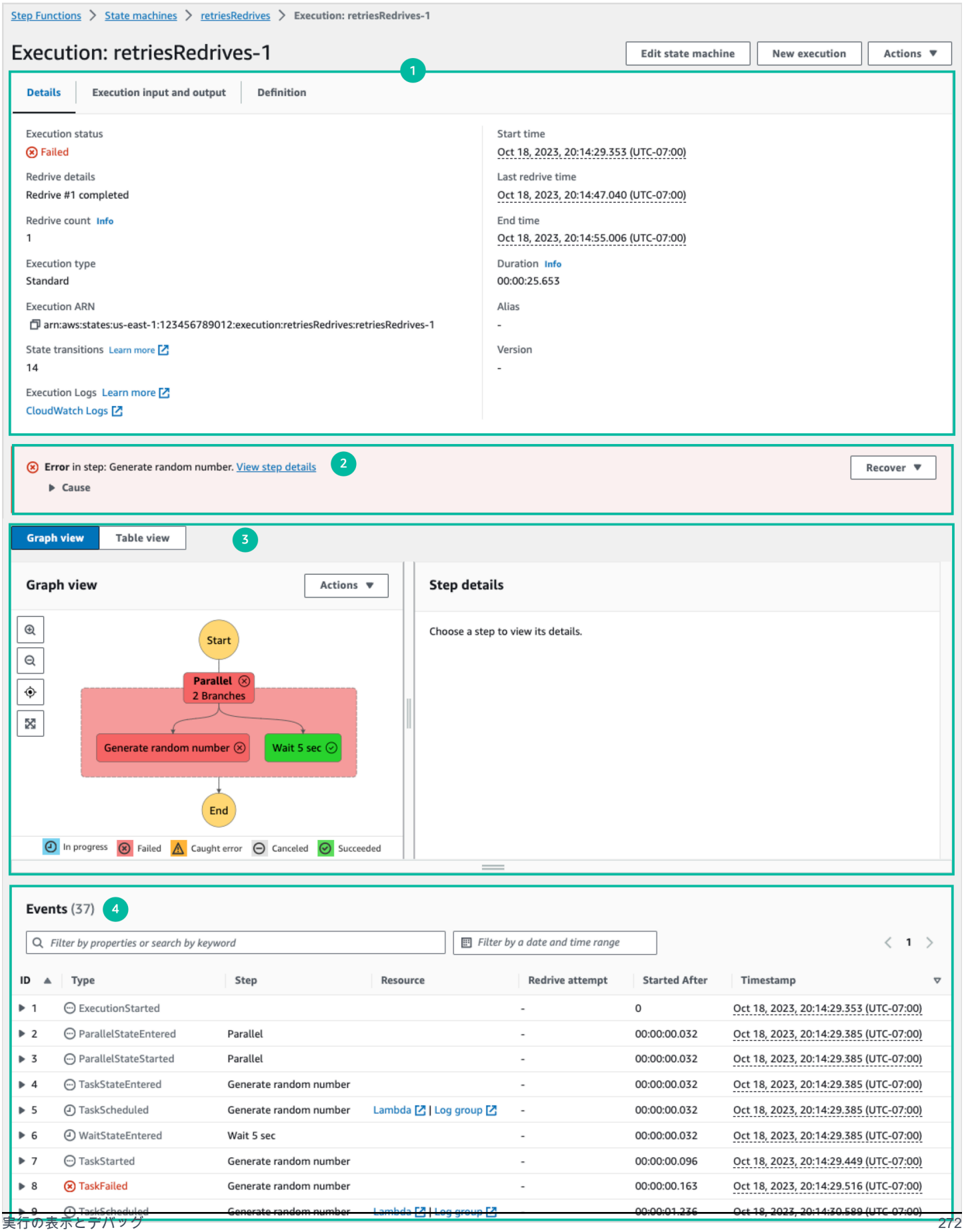
実行メトリクスの他に、[実行の詳細] ページにはステートマシンとその実行を管理するための以下のオプションがあります。

Button	次の操作をするには、このボタンを選択します。
ステートマシンの編集	ステートマシンの Amazon States Language 定義を編集します。
新しい実行	ステートマシンの新しい実行を開始します。
アクション	次のオプションから選択してください。 • [実行を停止] - 進行中の実行を停止します。このオプションは完了した実行には使用できません。 • Redrive - Redrive 過去 14 日間に正常に完了しなかった標準ワークフローの実行。これには、失敗した、中止された、またはタイムアウトした実行が含まれます。詳細については、「Redriving の実行」を参照してください。 • [エクスポート] - 実行の詳細を JSON 形式でエクスポートして、他のユーザーと共有したり、オフライン分析を実行したりできます。 • [フィードバックを送信] - インターフェイスに関するフィードバックを共有します。

バージョンまたはエイリアスで開始された実行の表示

Step Functions コンソールでは、バージョンまたはエイリアスで開始された実行を確認することもできます。詳細については、「Listing executions for versions and aliases」を参照してください。

[実行の詳細] コンソールページには、次のセクションが含まれています。


Step Functions > State machines > retriesRedrives > Execution: retriesRedrives-1
Execution: retriesRedrives-1
Edit state machine
New execution
Actions
1
Details
Execution input and output
Definition
Execution status
Failed
Redrive details
Redrive #1 completed
Redrive count Info
1
Execution type
Standard
Execution ARN
arn:aws:states:us-east-1:123456789012:execution:retriesRedrives:retriesRedrives-1
State transitions Learn more
14
Execution Logs Learn more
CloudWatch Logs
Start time
Oct 18, 2023, 20:14:29.353 (UTC-07:00)
Last redrive time
Oct 18, 2023, 20:14:47.040 (UTC-07:00)
End time
Oct 18, 2023, 20:14:55.006 (UTC-07:00)
Duration Info
00:00:25.653
Alias
-
Version
-
Error in step: Generate random number. View step details
2
Cause
Recover
Graph view
Table view
3
Graph view
Actions
Start
Parallel
2 Branches
Generate random number
Wait 5 sec
End
In progress
Failed
Caught error
Canceled
Succeeded
Step details
Choose a step to view its details.
Events (37)
4
Filter by properties or search by keyword
Filter by a date and time range
1
ID
Type
Step
Resource
Redrive attempt
Started After
Timestamp
1 ExecutionStarted - 0 Oct 18, 2023, 20:14:29.353 (UTC-07:00)
2 ParallelStateEntered Parallel - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
3 ParallelStateStarted Parallel - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
4 TaskStateEntered Generate random number - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
5 TaskScheduled Generate random number Lambda | Log group - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
6 WaitStateEntered Wait 5 sec - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
7 TaskStarted Generate random number - 00:00:00.096 Oct 18, 2023, 20:14:29.449 (UTC-07:00)
8 TaskFailed Generate random number - 00:00:00.163 Oct 18, 2023, 20:14:29.516 (UTC-07:00)
9 TaskScheduled Generate random number Lambda | Log group - 00:00:01.236 Oct 18, 2023, 20:14:30.589 (UTC-07:00)

1. 実行の概要
2. エラーメッセージ
3. 表示モード
4. ステップの詳細
5. イベント

実行の概要

[実行概要] セクションは [実行の詳細] ページの上部に表示されます。このセクションでは、ワークフローの実行の詳細の概要を説明します。この情報は次の 3 つのタブに分かれています。

詳細

実行のステータス、ARN、実行開始時刻と終了時刻のタイムスタンプなどの情報を表示します。ステートマシンの実行中に発生した [状態遷移] の合計数も表示できます。ステートマシンのトレースまたはログを有効にしている場合は、X-Ray トレースマップと Amazon CloudWatch Execution Logs のリンクを表示することもできます。

ステートマシンの実行が別のステートマシンによって開始された場合、このタブで親ステートマシンのリンクを表示できます。

ステートマシンの実行が redriven だった場合、このタブには Redrive カウントなどの redrive 関連情報が表示されます。

実行の入力と出力

ステートマシンの実行の入力と出力を表示します side-by-side。

定義

ステートマシンの Amazon States Language 定義を表示します。

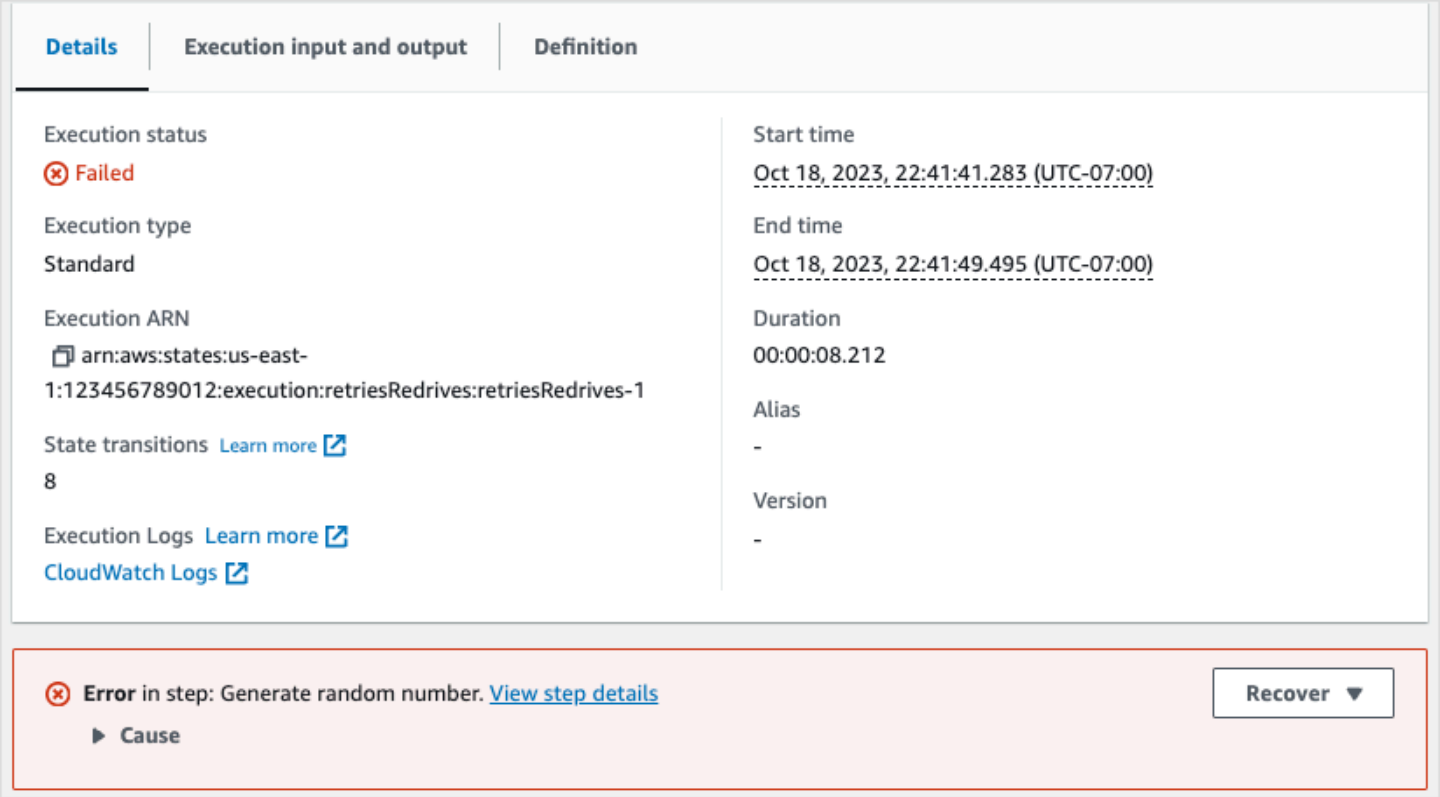
エラーメッセージ

ステートマシンの実行に失敗すると、[実行の詳細] ページにエラーメッセージが表示されます。エラーメッセージの [原因] または [ステップの詳細を表示] を選択すると、実行が失敗した理由またはエラーの原因となったステップが表示されます。

[ステップの詳細を表示] を選択すると、Step Functions は [ステップの詳細]、[グラフビュー]、[テーブルビュー] の各タブでエラーの原因となったステップを強調表示します。ステップがリトライを定義したタスク、マップ、またはパラレルの状態である場合、[ステップの詳細] ペインにはそのステッ

プの [再試行] タブが表示されます。さらに、実行をredrivenした場合、[ステップの詳細] ペインの [再試行とredrives] タブに再試行と実行のredriveの詳細が表示されます。

このエラーメッセージの [復元] ドロップダウンボタンから、失敗した実行をredriveまたは新しい実行を開始できます。詳細については、「Redriving の実行」を参照してください。



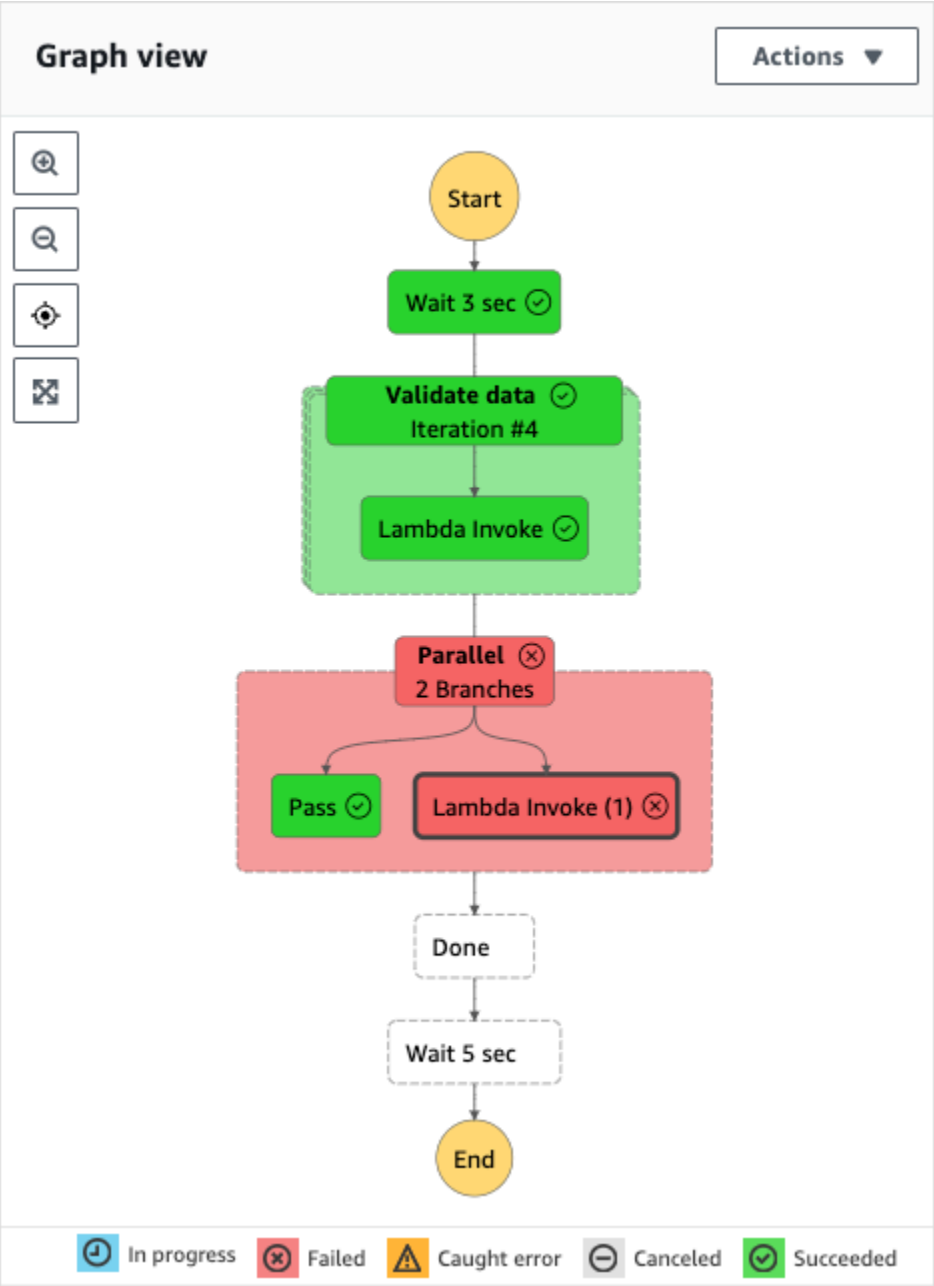
表示モード

[表示モード] セクションには、ステートマシンの 2 つの異なる可視化が含まれています。ワークフローのグラフィック表示、ワークフロー内の状態の概要を示す表、またはステートマシンの実行に関連するイベントのリストのいずれかを選択できます。

> **Note**
>
> タブを選択すると、その内容が表示されます。

Graph view

[グラフビュー] モードでは、ワークフローを図で示したものが表示されます。下部には、ステートマシンの実行ステータスを示す凡例が表示されます。また、ワークフロー全体を拡大/縮小したり、中央そろえにしたり、ワークフローを全画面モードで表示したりできるボタンもあります。



このビューでは、ワークフロー内の任意のステップを選択して、その実行に関する詳細を [ステップの詳細] コンポーネントに表示できます。[グラフビュー] でステップを選択すると、[テーブルビュー] にもそのステップが表示されます。これは逆の場合も同様です。[テーブルビュー] からステップを選択すると、[グラフビュー] にも同じステップが表示されます。

ステートマシンに `Map` 状態、`Parallel` 状態、あるいはその両方が含まれている場合は、ワークフロー内の [グラフビュー] でその名前を確認できます。さらに、`Map` 状態の場合、[グラフビュー] では [マップ] 状態実行データのさまざまな反復間を移動できます。例えば、[マップ] 状

態に 5 回の反復があり、3 回目と 4 回目の反復の実行データを表示する場合、次の操作を行います。

1. 反復データを表示する [マップ] 状態を選択します。
2. [マップイテレーションビューア] で、3 回目の反復のドロップダウンリストから [#2] を選択します。これは、反復回数が 0 からカウントされるためです。同様に、[マップ] 状態の 4 回目の反復では、ドロップダウンリストから #3 を選択します。

   また
   は、△
   と
   ▽
   コントロールを使用して [マップ] 状態の異なる反復間を移動することもできます。

   > **Note**
   >
   > ステートマシンにネストされた Map 状態が含まれている場合、親と子の Map 状態の反復のドロップダウンリストが次の例のように表示されます。
   >
   >
   >

3. (オプション) [マップ] 状態の反復処理が 1 つ以上実行に失敗した場合、または実行が中止された場合は、ドロップダウンリストの [失敗] または [中止] で反復番号を選択すると、そのデータを表示できます。

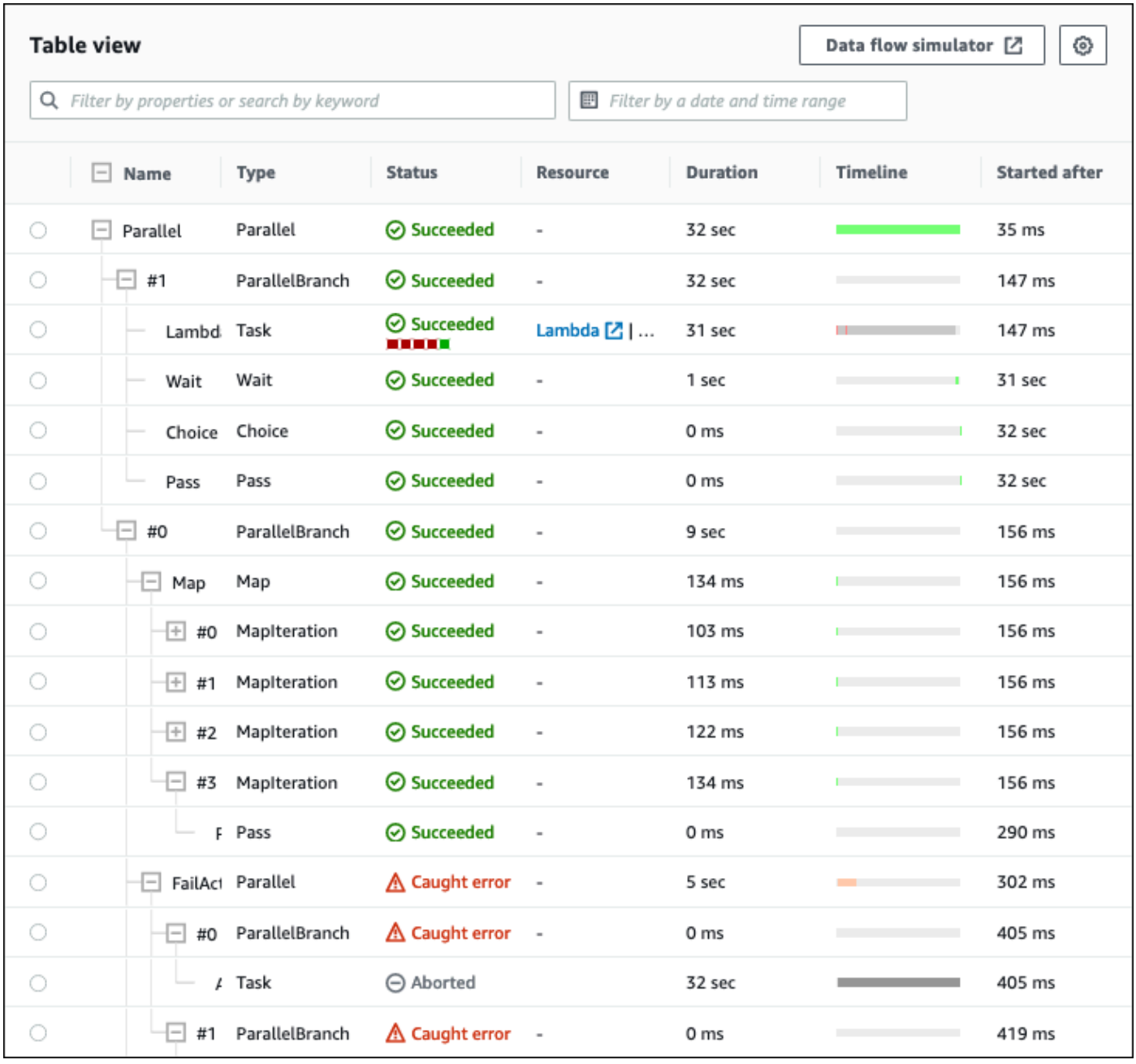
最後に、[エクスポート] ボタンと [レイアウト] ボタンを使用して、ワークフローグラフを SVG または PNG 画像としてエクスポートできます。ワークフローを水平表示と垂直表示間で切り替えることもできます。

Table view

[テーブルビュー] モードでは、ワークフロー内の状態が表形式で表示されます。この表示モードでは、ワークフローで実行された各状態の詳細を確認できます。これには、名前、使用したリソースの名前 (AWS Lambda 関数など) 、および状態が正常に実行されたかどうかが含まれます。

このビューでは、ワークフロー内の任意のステップを選択して、その実行に関する詳細を [ステップの詳細] コンポーネントに表示できます。[テーブルビュー] でステップを選択すると、[グラフビュー] にもそのステップが表示されます。これは逆の場合も同様です。[グラフビュー] からステップを選択すると、[テーブルビュー] にも同じステップが表示されます。

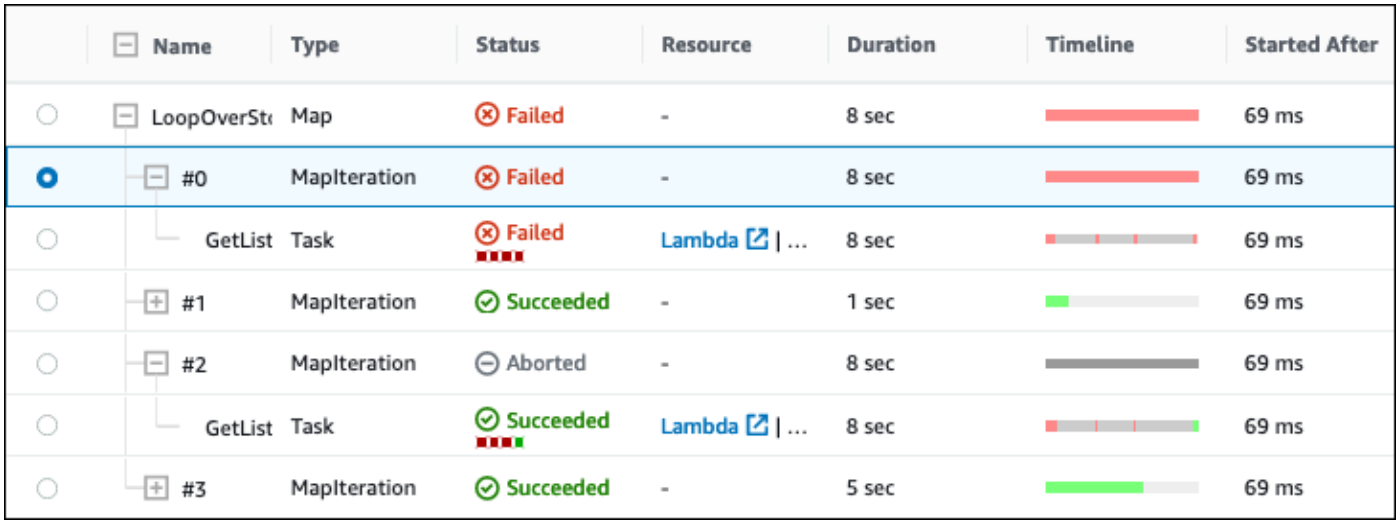
また、ビューにフィルターを適用して、[テーブルビュー] モードで表示されるデータ量を制限することもできます。[ステータス] や [Redrive の試行] など、特定のプロパティに対してフィルターを作成できます。詳細については、「チュートリアル: Step Functions コンソールを使用したステートマシン実行の調査」を参照してください。



デフォルトでは、このモードには、[名前]、[タイプ]、[ステータス]、[リソース]、[開始後] の各列が表示されます。表示する列は、[環境設定] ダイアログボックスを使用して設定できます。このダイアログボックスで行った選択は、再び変更されるまで今後ステートマシンを実行しても保持されます。

[タイムライン] 列を追加すると、実行全体における各状態の実行時間がランタイムを基準にして表示されます。これは色分けされた、線形のタイムラインとして表示されます。これにより、特定の状態の実行に関するパフォーマンス関連の問題を特定しやすくなります。タイムライン上の各状態の色分けされたセグメントは、進行中、失敗、中止などの状態の実行ステータスを識別するのに役立ちます。

例えば、ステートマシンの状態に対して実行リトライを定義した場合、そのリトライはタイムラインに表示されます。赤色のセグメントは失敗した Retry の試行を表し、薄い灰色のセグメントは各 Retry 試行の合間の BackoffRate を表します。



ステートマシンに Map 状態、Parallel 状態、あるいはその両方が含まれている場合は、ワークフロー内の [テーブルビュー] でその名前を確認できます。Map と Parallel 状態の場合、[テーブルビュー] モードでは反復と並列分岐の実行データがツリービュー内のノードとして表示されます。これらの状態の各ノードを選択すると、[ステップの詳細] セクションに個別の詳細を表示できます。例えば、状態が失敗する原因となった特定の [マップ] 状態反復のデータを確認できます。[マップ] 状態のノードを展開し、[ステータス] 列に各反復のステータスが表示されます。

ステップの詳細

[グラフビュー] または [テーブルビュー] で状態を選択すると、右側に [ステップの詳細] セクションが開きます。このセクションには以下のタブがあり、選択した状態に関する詳細な情報が表示されます。

Input (入力)

選択した状態の入力詳細が表示されます。入力内容にエラーがある場合は、タブヘッダーに

が表示されます。また、このタブではエラーの原因を確認できます。

また、[アドバンスドビュー] トグルボタンを選択して、データが選択した状態を通過したときの入力データ転送パスを確認することもできます。これにより、InputPath、Parameters、ResultSelector、OutputPath、ResultPath などの 1 つ以上のフィールドがデータに適用されたときに、入力がどのように処理されたかがわかります。

出力

選択した状態の出力を表示します。出力にエラーがある場合は、タブヘッダーに

が表示されます。また、このタブではエラーの原因を確認できます。

また、[アドバンスドビュー] トグルボタンを選択して、データが選択した状態を通過したときの出力データ転送パスを確認することもできます。これにより、InputPath、Parameters、ResultSelector、OutputPath、ResultPath などの 1 つ以上のフィールドがデータに適用されたときに、入力がどのように処理されたかがわかります。

詳細

状態タイプ、実行ステータス、実行時間などの情報を表示します。

などのリソースを使用するTask状態の場合 AWS Lambda、このタブには、リソース定義ページとリソース呼び出しの Amazon CloudWatch ログページへのリンクが表示されます。また、指定されている場合は、Task 状態の TimeoutSeconds と HeartbeatSeconds フィールドの値も表示されます。

Map 状態の場合、このタブには Map 状態の反復回数の合計に関する情報が表示されます。反復は、失敗、中止、成功、または に分類されますInProgress。

定義

選択した状態に対応する Amazon States Language 定義を表示します。

再試行

> **Note**
>
> このタブは、ステートマシンの `Task` または `Parallel` 状態で `Retry` フィールドを定義した場合にのみ表示されます。

選択した状態での最初の実行試行回数とそれ以降の再試行回数が表示されます。初回およびそれ以降のすべての失敗について、[タイプ] の横にある

を選択すると、ドロップダウンボックスに表示される [失敗の理由] が表示されます。再試行が成功すると、ドロップダウンボックスに表示される [出力] が表示されます。

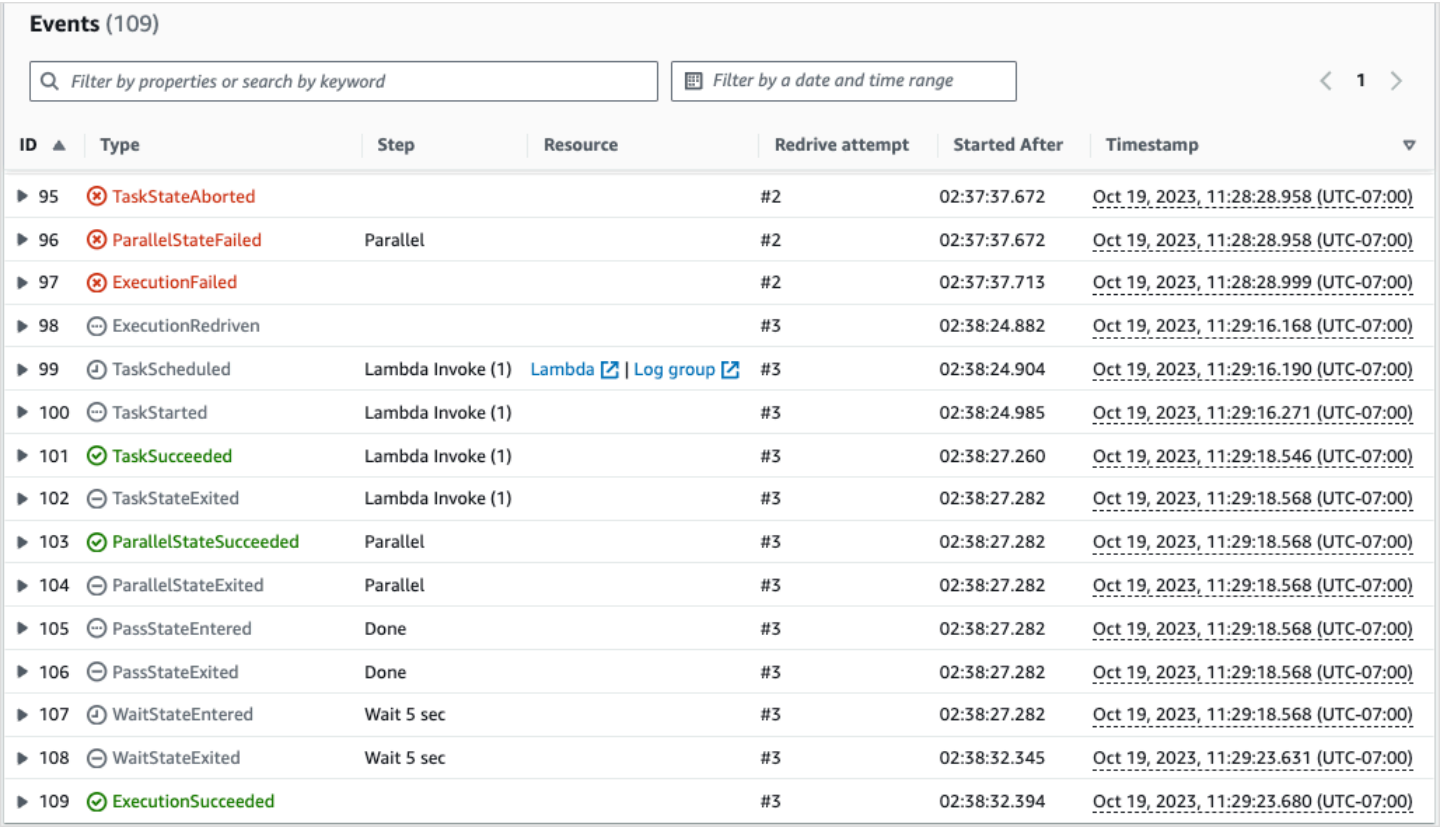
実行をredrivenした場合、このタブヘッダーには [再試行とredrives] という名前が表示され、それぞれのredriveの再試行の詳細が表示されます。

イベント

実行中の選択した状態に関連するイベントを、フィルター処理したリストで表示します。このタブに表示される情報は、[イベント] テーブルに表示される実行イベント履歴全体の一部にすぎません。

イベント

[イベント] テーブルには、選択した実行の全履歴が、複数のページにまたがるイベントのリストとして表示されます。各ページには最大 25 件のイベントが含まれます。このセクションにはイベントの合計数も表示されるので、イベント履歴の最大数である 25,000 イベントを超えているかどうかを判断するのに役立ちます。



Events (109)

Filter by properties or search by keyword | Filter by a date and time range | 1

ID	Type	Step	Resource	Redrive attempt	Started After	Timestamp
95	TaskStateAborted			#2	02:37:37.672	Oct 19, 2023, 11:28:28.958 (UTC-07:00)
96	ParallelStateFailed	Parallel		#2	02:37:37.672	Oct 19, 2023, 11:28:28.958 (UTC-07:00)
97	ExecutionFailed			#2	02:37:37.713	Oct 19, 2023, 11:28:28.999 (UTC-07:00)
98	ExecutionRedriven			#3	02:38:24.882	Oct 19, 2023, 11:29:16.168 (UTC-07:00)
99	TaskScheduled	Lambda Invoke (1)	Lambda | Log group	#3	02:38:24.904	Oct 19, 2023, 11:29:16.190 (UTC-07:00)
100	TaskStarted	Lambda Invoke (1)		#3	02:38:24.985	Oct 19, 2023, 11:29:16.271 (UTC-07:00)
101	TaskSucceeded	Lambda Invoke (1)		#3	02:38:27.260	Oct 19, 2023, 11:29:18.546 (UTC-07:00)
102	TaskStateExited	Lambda Invoke (1)		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
103	ParallelStateSucceeded	Parallel		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
104	ParallelStateExited	Parallel		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
105	PassStateEntered	Done		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
106	PassStateExited	Done		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
107	WaitStateEntered	Wait 5 sec		#3	02:38:27.282	Oct 19, 2023, 11:29:18.568 (UTC-07:00)
108	WaitStateExited	Wait 5 sec		#3	02:38:32.345	Oct 19, 2023, 11:29:23.631 (UTC-07:00)
109	ExecutionSucceeded			#3	02:38:32.394	Oct 19, 2023, 11:29:23.680 (UTC-07:00)

デフォルトでは、[イベント] テーブルの結果は、イベントの [タイムスタンプ] に基づいて昇順に表示されます。[タイムスタンプ] 列のヘッダーをクリックすると、実行イベント履歴のソートを降順に変更できます。

[イベント] テーブルでは、各イベントが実行ステータスを示すように色分けされています。例えば、失敗したイベントは赤で表示されます。イベントに関するその他の詳細情報を表示するには、イベント ID の横にある

を選択します。イベントを開くと、そのイベントの入力、出力、リソース呼び出しがイベント詳細に表示されます。

さらに、[イベント] テーブルでは、フィルターを適用して表示される実行イベント履歴の結果を制限できます。[ID] や [Redrive 再試行] などのプロパティを選択できます。詳細については、「チュートリアル: Step Functions コンソールを使用したステートマシン実行の調査」を参照してください。

チュートリアル: Step Functions コンソールを使用したステートマシン実行の調査

このチュートリアルでは、実行の詳細ページに表示される実行情報を調べて、実行が失敗した理由を確認する方法を学習します。次に、Map ステート実行のさまざまなイテレーションにアクセスする

方法について学習します。最後に、[テーブルビュー] で列を設定し、適切なフィルターを適用して関心のある情報のみを表示する方法を学習します。

このチュートリアルでは、フルーツセットの価格を取得する標準タイプのステートマシンを作成します。これを行うために、ステートマシンは 4 つの果物、各果物価格、果物の平均コストのランダムリストを返す 3 つの AWS Lambda 関数を使用します。Lambda 関数は、果物の価格がしきい値以下の場合にエラーを返します。

> **Note**
>
> 次の手順には、Standard ワークフロー実行の詳細を調べる方法が記載されていますが、Express ワークフロー実行の詳細を調べることもできます。Standard ワークフロータイプと Express ワークフロータイプの実行の詳細の違いについては、コンソールでの Standard ワークフローと Express ワークフローの実行 を参照してください。

コンテンツ

- ステップ 1: 必要な Lambda 関数を作成およびテストする
- ステップ 2: ステートマシンを作成および実行する
- ステップ 3: ステートマシンの実行の詳細を表示する
- ステップ 4: さまざまな表示モードを詳しく見る

ステップ 1: 必要な Lambda 関数を作成およびテストする

1. Lambda コンソールを開き、ステップ 1: Lambda 関数を作成する セクションのステップ 1～4 を実行します。Lambda 関数には必ず **`GetListOfFruits`** と名前を付けてください。
2. Lambda 関数を作成したら、ページの右上隅に表示されている関数の Amazon リソースネーム (ARN) をコピーします。ARN をコピーするには、[ ] をクリックします。ARN の例を次に示します。ここで、*`function-name`* は Lambda 関数の名前 (この場合は `GetListOfFruits`) です。

   ```
   arn:aws:lambda:us-east-1:123456789012:function:function-name
   ```

3. Lambda 関数の次のコードをGetListOfFruitsページのコードソース領域にコピーします。

   ```
   function getRandomSubarray(arr, size) {
   ```

```
    var shuffled = arr.slice(0), i = arr.length, temp, index;
    while (i--) {
        index = Math.floor((i + 1) * Math.random());
        temp = shuffled[index];
        shuffled[index] = shuffled[i];
        shuffled[i] = temp;
    }
    return shuffled.slice(0, size);
}

exports.handler = async function(event, context) {

    const fruits = ['Abiu','Açaí','Acerola','Ackee','African
 cucumber','Apple','Apricot','Avocado','Banana','Bilberry','Blackberry','Blackcurrant','Jos



     const errorChance = 45;

    const waitTime = Math.floor( 100 * Math.random() );

    await new Promise( r => setTimeout(() => r(), waitTime));

    const num = Math.floor( 100 * Math.random() );
    // const num = 51;
     if (num <= errorChance) {
         throw(new Error('Error'));
     }

    return getRandomSubarray(fruits, 4);
};
```

4. [デプロイ]、[テスト] の順に選択して変更をデプロイし、Lambda 関数の出力を確認します。
5. 次のステップで、**GetFruitPrice**、**CalculateAverage** という名前をそれぞれ付けた 2 つの Lambda 関数を追加します。

   a. 次のコードを GetFruitPrice Lambda 関数のコードソース領域にコピーします。

```
exports.handler = async function(event, context) {

    const errorChance = 0;
    const waitTime = Math.floor( 100 * Math.random() );

    await new Promise( r => setTimeout(() => r(), waitTime));
```

```
    const num = Math.floor( 100 * Math.random() );
    if (num <= errorChance) {
        throw(new Error('Error'));
    }

    return Math.floor(Math.random()*100)/10;
};
```

b. 次のコードを CalculateAverage Lambda 関数のコードソース領域にコピーします。

```
function getRandomSubarray(arr, size) {
    var shuffled = arr.slice(0), i = arr.length, temp, index;
    while (i--) {
        index = Math.floor((i + 1) * Math.random());
        temp = shuffled[index];
        shuffled[index] = shuffled[i];
        shuffled[i] = temp;
    }
    return shuffled.slice(0, size);
}

const average = arr => arr.reduce( ( p, c ) => p + c, 0 ) / arr.length;

exports.handler = async function(event, context) {
        const errors = [
        "Error getting data from DynamoDB",
        "Error connecting to DynamoDB",
        "Network error",
        "MemoryError - Low memory"
        ]

    const errorChance = 0;

    const waitTime = Math.floor( 100 * Math.random() );

    await new Promise( r => setTimeout(() => r(), waitTime));

    const num = Math.floor( 100 * Math.random() );
    if (num <= errorChance) {
        throw(new Error(getRandomSubarray(errors, 1)[0]));
    }

    return average(event);
```

```
};
```

c. 必ず、これら 2 つの Lambda 関数の ARN をコピーしてから、[デプロイ] および [テスト] をクリックしてください。

ステップ 2: ステートマシンを作成および実行する

Step Functions コンソールを使用して、ステップ 1 で作成した Lambda 関数を呼び出すステートマシンを作成します。このステートマシンには 3 つの `Map` ステートが定義されています。これらの各 `Map` ステートには、Lambda 関数のいずれかを呼び出す `Task` ステートが含まれています。また、各 `Task` ステートに 1 つの `Retry` フィールドと再試行回数が定義されています。`Task` ステートでランタイムエラーが発生した場合、その `Task` ステートに定義されている再試行回数まで再実行されます。

1. Step Functions コンソールを開き、[コードでワークフローを記述] を選択します。

> **Important**
>
> ステートマシンが、前に作成した Lambda 関数と同じ AWS アカウントとリージョンにあることを確認します。

2. [タイプ] については、デフォルトの [標準] のままにします。
3. 以下の Amazon States Language をコピーして [定義] に貼り付けます。表示されている ARN は、先に作成した Lambda 関数の ARN に必ず置き換えてください。

```
{
    "StartAt": "LoopOverStores",
    "States": {
        "LoopOverStores": {
            "Type": "Map",
            "Iterator": {
                "StartAt": "GetListOfFruits",
                "States": {
                    "GetListOfFruits": {
                        "Type": "Task",
                        "Resource": "arn:aws:states:::lambda:invoke",
                        "OutputPath": "$.Payload",
                        "Parameters": {
                            "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:GetListofFruits:$LATEST",
```

```
                            "Payload": {
                                "storeName.$": "$"
                            }
                        },
                        "Retry": [
                            {
                                "ErrorEquals": [
                                    "States.ALL"
                                ],
                                "IntervalSeconds": 2,
                                "MaxAttempts": 1,
                                "BackoffRate": 1.3
                            }
                        ],
                        "Next": "LoopOverFruits"
                    },
                    "LoopOverFruits": {
                        "Type": "Map",
                        "Iterator": {
                            "StartAt": "GetFruitPrice",
                            "States": {
                                "GetFruitPrice": {
                                    "Type": "Task",
                                    "Resource": "arn:aws:states:::lambda:invoke",
                                    "OutputPath": "$.Payload",
                                    "Parameters": {
                                        "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:GetFruitPrice:$LATEST",
                                        "Payload": {
                                            "fruitName.$": "$"
                                        }
                                    },
                                    "Retry": [
                                        {
                                            "ErrorEquals": [
                                                "States.ALL"
                                            ],
                                            "IntervalSeconds": 2,
                                            "MaxAttempts": 3,
                                            "BackoffRate": 1.3
                                        }
                                    ],
                                    "End": true
                                }
```

```
                            }
                        },
                        "ItemsPath": "$",
                        "End": true
                    }
                }
            },
            "ItemsPath": "$.stores",
            "Next": "LoopOverStoreFruitsPrice",
            "ResultPath": "$.storesFruitsPrice"
        },
        "LoopOverStoreFruitsPrice": {
            "Type": "Map",
            "End": true,
            "Iterator": {
                "StartAt": "CalculateAverage",
                "States": {
                    "CalculateAverage": {
                        "Type": "Task",
                        "Resource": "arn:aws:states:::lambda:invoke",
                        "OutputPath": "$.Payload",
                        "Parameters": {
                            "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:Calculate-average:$LATEST",
                            "Payload.$": "$"
                        },
                        "Retry": [
                            {
                                "ErrorEquals": [
                                    "States.ALL"
                                ],
                                "IntervalSeconds": 2,
                                "MaxAttempts": 2,
                                "BackoffRate": 1.3
                            }
                        ],
                        "End": true
                    }
                }
            },
            "ItemsPath": "$.storesFruitsPrice",
            "ResultPath": "$.storesPriceAverage",
            "MaxConcurrency": 1
        }
```

```
    }
}
```

4. ステートマシンに名前を入力します。このページの他のオプションについては、デフォルトのままにし、[ステートマシンの作成] を選択します。
5. ステートマシン名が記載されたページを開きます。ステップ 4: ステートマシンを実行する セクションのステップ 1～4 を実行しますが、実行入力には以下のデータを使用します。

```
{
    "stores": [
      "Store A",
      "Store B",
      "Store C",
      "Store D"
    ]
}
```

ステップ 3: ステートマシンの実行の詳細を表示する

実行 ID がタイトルに表示されているページでは、実行結果を確認するほか、エラーをデバッグできます。

1. (オプション) 実行の詳細 ページに表示されているタブから選択すると、それぞれのタブに表示される情報を確認できます。例えば、ステートマシンの入力とその実行出力を表示するには、実行概要セクションの [実行の入力と出力] を選択します。
2. ステートマシンの実行に失敗した場合は、エラーメッセージで [原因] または [ステップの詳細を表示] を選択します。エラーの詳細はステップの詳細セクションに表示されます。エラーの原因となったステップは、 という名前Taskの状態でありGetListofFruits、グラフビュー とテーブルビュー で強調表示されていることに注意してください。

> **Note**
>
> GetListofFruits ステップは Map状態内で定義され、ステップが正常に実行されなかったため、Map状態ステップのステータスは失敗と表示されます。

ステップ 4: さまざまな表示モードを詳しく見る

好みのモードを選択して、ステートマシンワークフローまたは実行イベント履歴を表示することができます。これらの表示モードで実行できるタスクには、次のようなものがあります。

[グラフビュー] - 別の **Map** ステートイテレーションに切り替える

[Map] ステートのイテレーションが 5 回あり、3 回目と 4 回目のイテレーションの実行詳細を表示する場合は、次の操作を行います。

1. イテレーションデータを表示する Map ステートを選択します。
2. [マップイテレーションビューア] から、表示するイテレーションを選択します。イテレーション回数はゼロからカウントされます。5 回のうち 3 回目のイテレーションを選択するには、[Map] ステート名の横にあるドロップダウンリストから [#2] を選択します。

> **Note**
>
> ステートマシンにネストされた Map ステートが含まれている場合、Step Functions は親と子の Map ステートイテレーションを別々の 2 つのドロップダウンリストとして表示します。
>
>
>

3. (オプション) 1 つ以上の Map ステートイテレーションの実行に失敗したか、中断した状態で停止された場合は、失敗したイテレーションに関する詳細を確認できます。これらの詳細を確認するには、ドロップダウンリストの [失敗] または [中断] で該当するイテレーション番号を選択します。

[テーブルビュー] - 別の **Map** ステートイテレーションに切り替える

[Map] ステートのイテレーションが 5 回あり、3 回目と 4 回目のイテレーションについて実行の詳細を表示する場合は、次の操作を行います。

1. 別のイテレーションデータを表示する Map ステートを選択します。

2. Map ステートイテレーションのツリービュー表示で、3 回目のイテレーションは #2 という名前が付いたイテレーションの行を選択します。同様に、4 回目のイテレーションは #3 という名前の行を選択します。

[テーブルビュー] - 表示する列を設定する

[ ] を選択します。次に、[環境設定] ダイアログボックスの [表示する列を選択] で表示する列を選択します。

デフォルトでは、このモードには、[名前]、[タイプ]、[ステータス]、[リソース]、[開始後] の各列が表示されます。

[テーブルビュー] - 結果をフィルタリングする

ステータスなどのプロパティや日付と時刻の範囲に基づいて 1 つ以上のフィルターを適用して、表示する情報量を制限します。例えば、実行に失敗したステップを表示するには、次のフィルターを適用します。

1. [プロパティによるフィルタリング、またはキーワードによる検索] を選択し、[プロパティ] で [ステータス] を選択します。
2. [オペレータ] で [ステータス =] を選択します。
3. [ステータス = 失敗] を選択します。
4. (オプション) 適用したフィルターを削除するには、[フィルターをクリア] を選択します。

[イベントビュー] - 結果をフィルタリングする

[タイプ] などのプロパティや日付と時刻の範囲に基づいて 1 つ以上のフィルターを適用して、表示する情報量を制限します。例えば、実行に失敗した Task ステートを表示するには、次のフィルターを適用します。

1. [プロパティによるフィルタリング、またはキーワードによる検索] を選択し、[プロパティ] で [タイプ] を選択します。
2. [オペレータ] で [タイプ =] を選択します。
3. Type = TaskFailed を選択します。
4. (オプション) 適用したフィルターを削除するには、[フィルターをクリア] を選択します。

イベントビュー – TaskFailedイベントの詳細を検査する

TaskFailed イベント ID

の横にある を選択して、ドロップダウンボックスに表示される入力、出力、リソース呼び出しなどの詳細を確認します。

Redriving の実行

redrive を使用して、過去 14 日間に正常に完了しなかった標準ワークフローの実行を再開できます。これには、失敗した、中止された、またはタイムアウトした実行が含まれます。

redrive を実行すると、失敗したステップから失敗した実行が継続され、同じ入力が使用されます。Step Functions は、成功したステップの結果と実行履歴を保存し、redrive の実行時にこれらのステップは再実行されません。例えば、ワークフローに 2 つの状態、1 つは パス 状態、もう 1 つはそれに続く タスク 状態です ワークフローの実行が Task 状態で失敗し、実行を redrive すると、実行のスケジュールが再設定され、Task 状態が再実行されます。

再処理された実行では、元の実行試行に使用されたのと同じステートマシン定義と実行 ARN が使用されます。元の実行試行がバージョン、エイリアス、あるいはその両方に関連付けられていた場合、redriven された実行は同じバージョン、エイリアス、または両方に関連付けられます。エイリアスを別のバージョンを指すように更新しても、再処理された実行には元の実行試行に関連付けられたバージョンが使用され続けます。再処理された実行では同じステートマシン定義が使用されるため、ステートマシン定義を更新した場合は新しい実行を開始する必要があります。

実行を再処理すると、ステートマシンレベルのタイムアウト (定義されている場合) は 0 にリセットされます。ステートマシンレベルのタイムアウトの詳細については、「`TimeoutSeconds`」を参照してください。

実行の再処理は状態遷移と見なされます。状態遷移が課金に及ぼす影響については、Step Functions コストを参照してください。

トピック

- 失敗した実行の再処理対象
- 個々の状態の再処理動作
- 実行を再処理する IAM アクセス許可
- コンソールでの実行の再処理
- API を使用した実行の再処理

- 再処理された実行内容を調べる
- 再処理された実行の再試行動作

失敗した実行の再処理対象

最初に試みた実行が以下の条件を満たしていれば、実行を再処理することができます。

- 2023 年 11 月 15 日以降に実行を開始しました。この日付より前に開始した実行は再処理の対象外です。
- 実行ステータスは SUCCEEDED ではありません。
- ワークフローの実行は 14 日間の再処理可能期間を超えていません。再処理可能期間とは、特定の実行を再処理できる時間を指します。この期間は、ステートマシンが実行を完了した日から始まります。
- ワークフローの実行は、1 年間の最大オープン時間を超えていません。ステートマシン実行クォータの詳細については、「ステートマシンの実行に関連するクォータ」を参照してください。
- 実行イベント履歴数は 24,999 件未満です。再処理された実行によって、イベント履歴が既存のイベント履歴に追加されます。ExecutionRedriven 履歴イベントと少なくとも 1 つのその他の履歴イベントに対応するため、ワークフロー実行に含まれるイベントが 24,999 件未満であることを確認してください。

個々の状態の再処理動作

ワークフローで失敗した状態によって、失敗したすべての状態の再処理動作は異なります。次の表では、すべての状態に対する再処理動作について説明します。

状態名	再処理実行動作
パス	前のステップが失敗したり、ステートマシンがタイムアウトしたりすると、パス状態は終了し、再処理では実行されません。
タスク	タスク状態をスケジュールして再開します。 タスク状態を再実行する実行を再処理すると、その状態の (定義されている場合) TimeoutSeconds は 0 にリセットされます。タイムアウ

状態名	再処理実行動作
	ト値の詳細については、「タスク状態」を参照してください。
選択	Choice ステートルールを再評価します。
待機	状態が過去のタイムスタンプを参照する Timestamp または TimestampPath を指定している場合、再処理は Wait ステートを終了させ、Next フィールドで指定されたステートに入ります。
成功	Succeed ステートになるステートマシン実行を再処理しません。
失敗	再び Fail ステートになり、再び失敗します。
並行	失敗または中止されたブランチのみを再スケジュールし、再処理します。 States.DataLimitExceeded エラーが原因でステートが失敗した場合、最初の実行試行で成功したブランチを含めて、Parallel ステートが再実行されます。
インラインマップステート	スケジュールを変更し、失敗または中止されたイテレーションのみを再スケジュールし、再処理します。 States.DataLimitExceeded エラーが原因でステートが失敗した場合、最初の実行試行で成功した反復を含めて、インラインマップステートが再実行されます。

状態名	再処理実行動作
分散マップ状態	マップ実行で失敗した子ワークフローの実行を再処理します。詳細については、「マップ実行の再処理」を参照してください。 States.DataLimitExceeded エラーが原因でステートが失敗すると、分散マップ状態が再実行されます。これには、最初の実行試行で成功した子ワークフローも含まれます。

実行を再処理する IAM アクセス許可

Step Functions には、実行を再処理するための適切なアクセス許可が必要です。次の IAM ポリシー例では、実行を再処理するためにステートマシンに必要な最小限の権限を付与します。######のテキストを、リソース固有の情報に必ず置き換えてください。

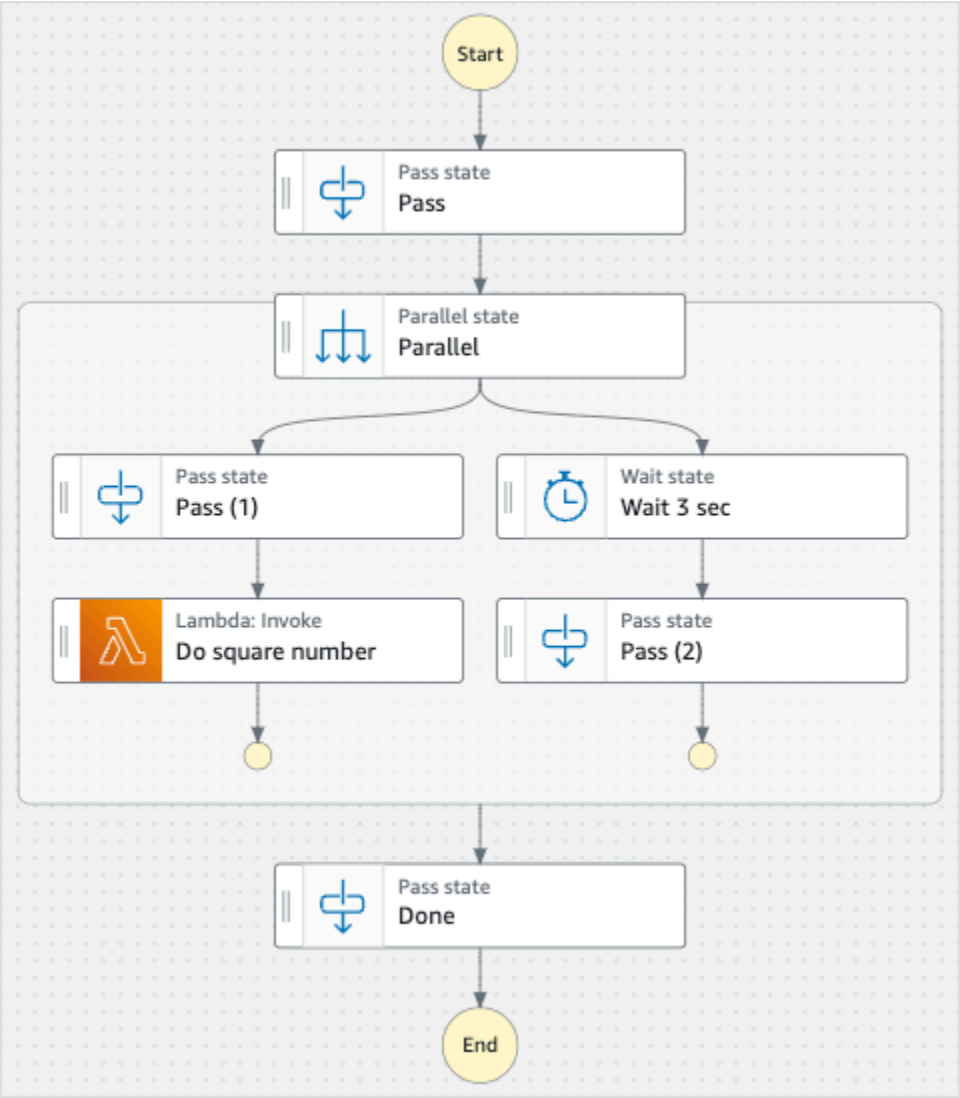
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:RedriveExecution"
            ],
            "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:*"
        }
    ]
}
```

マップ実行の再処理に必要なアクセス許可の例については、「分散マップの IAM redriving ポリシーの例」を参照してください。

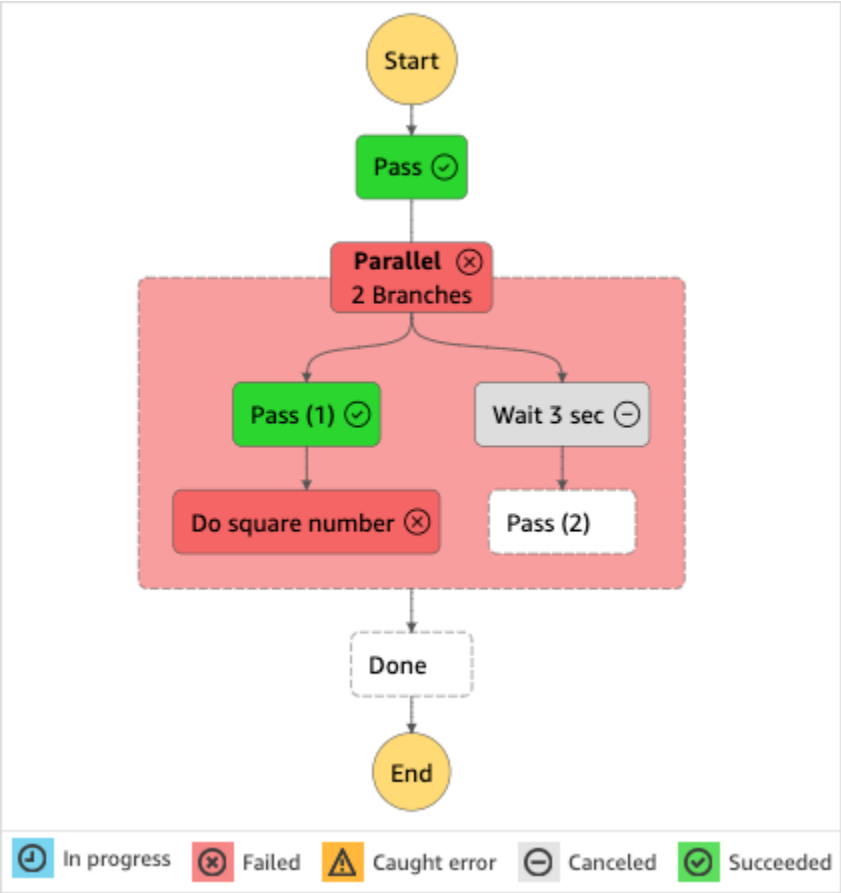
コンソールでの実行の再処理

Step Functions コンソールから対象となる実行を再処理できます。

例えば、以下のイメージがステートマシンのワークフローグラフを表しているとしましょう。



このステートマシンを実行するとします。次のイメージに、ステートマシンの実行のグラフを示します。



この図に示すように、[Parallel] ステート内の [平方数を実行する] という名前の [Lambda 呼び出し] ステップがエラーを返しました。このため、[Parallel] ステートは失敗しました。実行が進行中または開始されていないブランチは停止し、ステートマシンの実行は失敗します。

コンソールから実行を再処理するには

1. Step Functions コンソールを開き、実行に失敗した既存のステートマシンを選択します。
2. ステートマシンの詳細ページの [実行] で、失敗した実行インスタンスを選択します。
3. [Redrive] を選択します。
4. [Redrive] ダイアログボックスで [Redrive 実行] を選択します。

> **Tip**
>
> 失敗した実行の[実行の詳細] ページを開いている場合は、次のいずれかを実行して、実行を再処理します。
>
> - [復旧] を選択し、[障害からのリドライブ] を選択します。

- [アクション]、[Redrive] の順に選択します。

再処理は、同じステートマシン定義と ARN を使用していることに注意してください。最初の実行試行で失敗したステップから実行を継続します。この例では、それは、[Parallel] ステート内の [平方数を実行する] ステップと [3 秒待機] ブランチです。[Parallel] ステートで失敗したこれらのステップの実行を再開した後は、[完了] ステップまで実行を継続します。

5. 実行を選択して、[実行の詳細] ページを開きます。

   このページでは、redriven 実行の結果を表示できます。例えば、実行の概要 セクションには、実行が再処理された回数を表す [リドライブ回数] が表示されます。[イベント] セクションでは、元の実行試行のイベントに追加された再処理関連の実行イベントを確認できます。例えば、`ExecutionRedriven` イベントです。

API を使用した実行の再処理

RedriveExecution API を使用して、実行をredrive適格なものにすることができます。この API は、標準ワークフローの実行に失敗した場合に、失敗、中止、またはタイムアウトになったステップから再開します。

AWS Command Line Interface（ AWS CLI) で、失敗したredriveステートマシンの実行に対して次のコマンドを実行します。`######`のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

再処理された実行内容を調べる

コンソールでredriven実行を調べるか、 APIs: GetExecutionHistoryおよび を使用して実行を調べることができますDescribeExecution。

コンソールで再処理された実行内容を調べる

1. Step Functions コンソールを開き、実行を再処理した既存のステートマシンを選択します。
2. [実行の詳細] ページを開きます。

   このページでは、redriven 実行の結果を表示できます。例えば、実行の概要 セクションには、実行が再処理された回数を表す [リドライブ回数] が表示されます。[イベント] セクションで

は、元の実行試行のイベントに追加された再処理関連の実行イベントを確認できます。例えば、ExecutionRedriven イベントです。

API を使用して再処理された実行内容を調べる

ステートマシンの実行を再処理した、次の API のいずれかを使用して再処理された実行に関する詳細を表示できます。######のテキストを、リソース固有の情報に必ず置き換えてください。

- GetExecutionHistory – 指定された実行の履歴をイベントのリストとして返します。この API は、可能であれば、実行の再処理試行に関する詳細も返します。

  で AWS CLI、次のコマンドを実行します。

```
aws stepfunctions get-execution-history --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

- DescribeExecution – ステートマシンの実行に関する情報を提供します。これには、実行に関連するステートマシン、実行の入出力、再処理実行の詳細 (ある場合)、関連する実行メタデータなどがあります。

  で AWS CLI、次のコマンドを実行します。

```
aws stepfunctions describe-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

再処理された実行の再試行動作

再処理された実行により、再試行を定義した タスク、並行、またはインラインマップステートが再実行される場合、これらのステートの再試行回数は 0 にリセットされます。これにより、再処理の最大試行回数を設定できます。redriven 実行では、コンソールを使用してこれらのステートでの再試行を個別に追跡できます。

コンソールで個々の再試行回数を調べるには

1. Step Functions コンソールの実行の詳細ページで、再処理の再試行されたステートを選択します。
2. [再試行とリドライブ] タブを選択します。

3. 各再試行の横にある

   ▶

   を選択して、その詳細を表示します。再試行が成功すると、ドロップダウンボックスに表示される [出力] に結果が表示されます。

以下のイメージは、元の実行試行時とその実行の再処理時のステートに対して実行された再試行の例を示しています。この図では、元の試行と実行の再処理試行で 3 回の再試行が行われています。4 回目の再処理試行で成功し、16 という出力が返されます。

Input | Output | Details | Definition | **Retries & redrives** | Events

Type	Status	Resource	Duration	Time
▶ Original execution	Failed	Logs | Lambda | Log group	00:00:00.151	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.139	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.164	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.149	
▶ Redrive #1	Failed	Logs | Lambda | Log group	00:00:00.187	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.147	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.154	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.170	
▶ Redrive #2	Failed	Logs | Lambda | Log group	00:00:00.206	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.184	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.188	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.219	
▶ Redrive #3	Failed	Logs | Lambda | Log group	00:00:00.198	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.142	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.174	
▶ Retry	Failed	Logs | Lambda | Log group	00:00:00.208	
▼ Redrive #4	Succeeded	Logs | Lambda | Log group	00:00:00.195	

Output Learn more

```
1 ▾ {
2     "Squared": 16
3   }
```

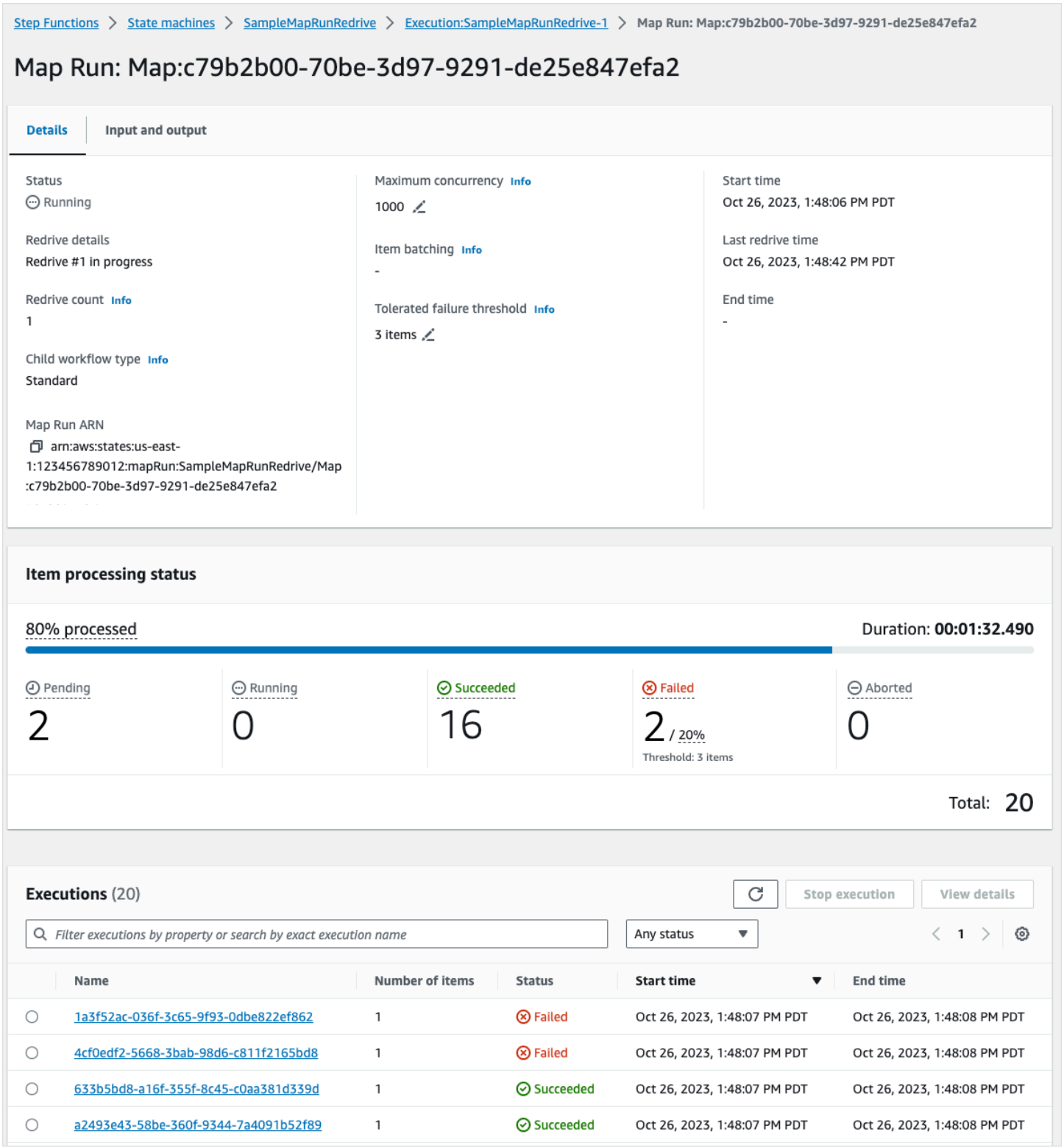
Formatted

分散マップ状態実行のマップ実行の検証

分散モードで Map 状態を実行すると、Step Functions はマップ実行リソースを作成します。マップ実行とは、分散マップ状態によって開始する一連の子ワークフロー実行、およびこれらの実行をコントロールするランタイム設定を指します。Step Functions は、マップの実行に Amazon リソースネーム (ARN) を割り当てます。マップ実行は、Step Functions コンソールで確認できます。`DescribeMapRun` API アクションを呼び出すこともできます。マップ実行は、 にメトリクスも出力します CloudWatch。

Step Functions コンソールには、分散マップ状態実行に関連するすべての情報を表示する [マップ実行の詳細] ページがあります。例えば、分散マップ状態の実行ステータス、マップ実行の ARN、および分散マップ状態によって開始された子ワークフロー実行で処理されたアイテムのステータスを表示できます。また、すべての子ワークフロー実行のリストを表示したり、その詳細にアクセスしたりすることもできます。さらに、マップ実行が redriven だった場合は、[マップ実行] 実行の概要 セクションにマップ実行の redrive の詳細が表示されます。例えば、「前回の redrive」などです。コンソールには、この情報がダッシュボード形式で表示されます。

[マップ実行の詳細] ページには、次のセクションがあります。



コンテンツ

- [マップ実行] 実行の概要
- エラーメッセージ

- アイテム処理ステータス
- 実行リスト
- マップ実行の再処理
  - マップ実行での子ワークフローの対象を再処理する
  - 子ワークフローの実行再処理動作
  - マップ実行再処理で使用される入力のシナリオ
  - マップ実行を再処理する IAM アクセス許可
  - コンソールでのマップ実行の再処理
  - RedrivingAPI を使用したマップ実行の再処理

[マップ実行] 実行の概要

[マップ実行の概要] セクションは、[マップ実行の詳細] ページの上部に表示されます。このセクションでは、分散マップ状態の実行の詳細の概要を説明します。この情報は次のタブに分かれています。

詳細

分散マップ状態の実行ステータス、マップ実行 ARN、分散マップ状態によって開始された子ワークフロー実行の種類などの情報を表示します。マップ実行の許容障害しきい値や、子ワークフロー実行に指定された最大同時実行数など、その他の設定も表示できます。これらの設定を編集することも可能です。

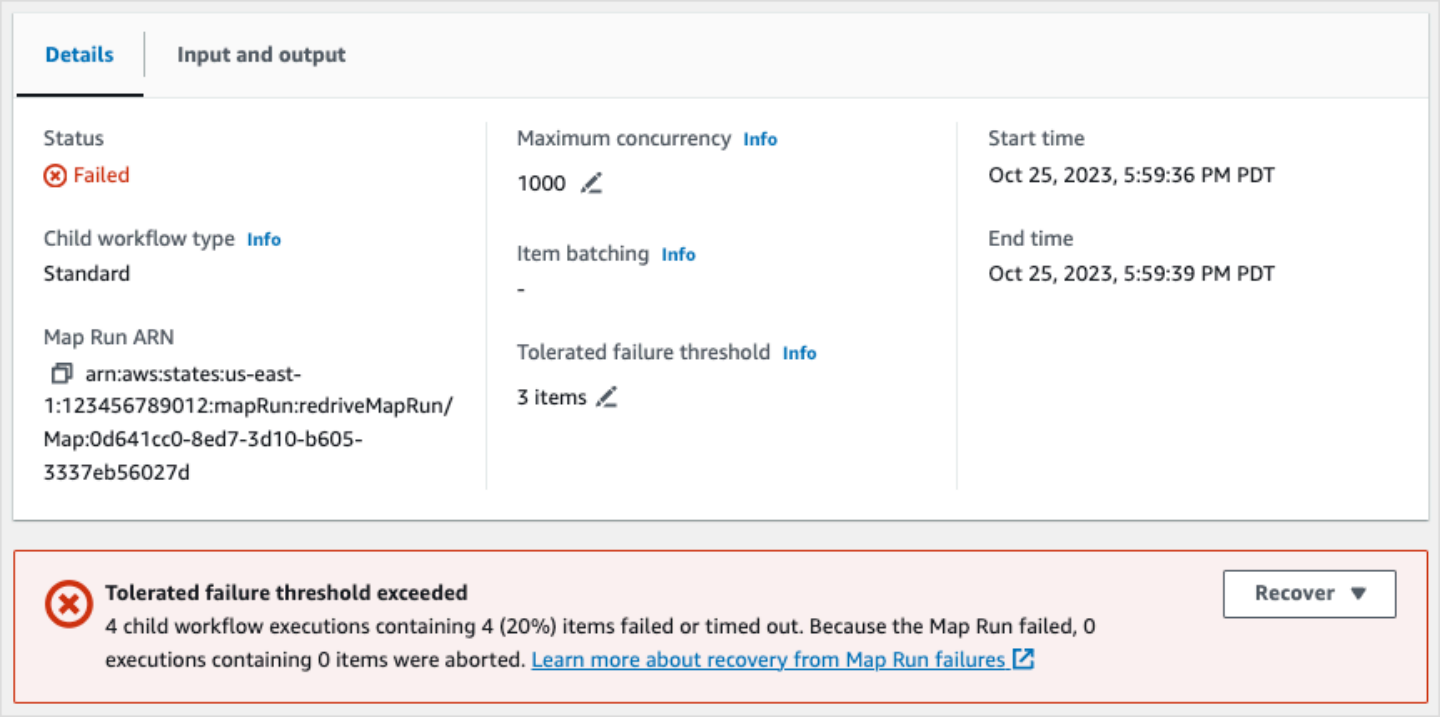
入力と出力

分散マップ状態が受信した入力と、ステートが生成する対応する出力を表示します。例えば、入力データセットとその場所、およびそのデータセット内の個々のデータアイテムに適用された入力フィルターを表示できます。分散マップ状態実行の出力をエクスポートする場合、このタブには実行結果を含む Amazon S3 バケットへのパスが表示されます。それ以外の場合は、親ワークフローの [実行の詳細] ページに移動して実行出力が表示されます。

エラーメッセージ

マップ実行が失敗した場合、[マップ実行の詳細] ページには失敗の理由を示すエラーメッセージが表示されます。

このエラーメッセージの [復旧] ドロップダウンボタンから、このマップ実行によって開始された失敗した子ワークフローの実行をredriveするか親ワークフローの新規実行を開始できます。詳細については、「マップ実行の再処理」を参照してください。



アイテム処理ステータス

[アイテム処理ステータス] セクションには、マップ実行で処理されたアイテムのステータスが表示されます。例えば、[保留中] は、子ワークフロー実行がまだアイテムの処理を開始していないことを示します。

アイテムのステータスは、アイテムを処理している子ワークフロー実行のステータスによって異なります。子ワークフローの実行が失敗したり、タイムアウトになったり、ユーザーが実行をキャンセルしたりした場合、Step Functions はその子ワークフロー実行内の項目の処理結果に関する情報を受け取りません。その実行によって処理されたすべてのアイテムは、子ワークフロー実行のステータスを共有します。

例えば、2 回の子ワークフロー実行で 100 項目を処理し、各実行で 50 項目のバッチを処理するとします。実行の 1 つが失敗し、もう 1 つが成功した場合、成功したアイテムが 50 個、失敗したアイテムが 50 個になります。

次の表では、すべてのアイテムで利用できる処理ステータスのタイプを説明しています。

ステータス	説明
[保留中]	子ワークフロー実行がまだ処理を開始していない項目を示します。アイテムの処理が開始される前にマップ実行が停止、失敗、またはユーザーが実行をキャンセルした場合、アイテムは [保留中] ステータスのままになります。 例えば、処理待ちのアイテムが 10 個ある状態でマップ実行が失敗した場合、この 10 個のアイテムは [保留中] ステータスのままになります。
実行中	子ワークフロー実行によって現在処理中のアイテムを示します。
成功	子ワークフロー実行が項目を正常に処理したことを示します。 成功した子ワークフローの実行が、失敗したアイテムを持つことはできません。実行中にデータセット内の 1 つのアイテムに障害が発生すると、子ワークフローの実行全体が失敗します。
[失敗]	子ワークフローの実行がアイテムの処理に失敗したか、実行がタイムアウトしたことを示します。子ワークフロー実行によって処理された項目が 1 つでも失敗すると、子ワークフローの実行全体が失敗します。 例えば、1000 項目を処理した子ワークフローの実行の場合を考えます。そのデータセット内の 1 つのアイテムが実行中に失敗した場合、Step Functions は子ワークフローの実行全体が失敗したと見なします。 マップ実行をredriveすると、このステータスのアイテムの数は 0 にリセットされます。

ステータス	説明
Aborted	子ワークフロー実行によってアイテムの処理が開始されたが、ユーザーが実行をキャンセルしたか、マップ実行が失敗したために Step Functions が実行を停止したことを示します。 例えば、50 個のアイテムを処理している [実行中] の子ワークフロー実行を考えてみましょう。障害またはユーザーが実行をキャンセルしたためにマップ実行が停止した場合、子ワークフローの実行と 50 項目すべてのステータスが [中断] に変わります。 Express タイプの子ワークフロー実行を使用する場合、実行を停止することはできません。 Express タイプの子ワークフロー実行を開始するマップ実行を redrive すると、このステータスのアイテムの数は 0 にリセットされます。これは、Express の子ワークフローが、 ではなく StartExecution API アクションを使用して再起動されるためですredriven。

実行リスト

[実行] セクションには、特定のマップ実行のすべての子ワークフロー実行が一覧表示されます。[正確な実行名で検索] フィールドを使用して、特定の子ワークフロー実行を検索します。[任意のステータス] ドロップダウンを使用して、子ワークフローの実行履歴をステータスでフィルタリングすることもできます。特定の実行に関する詳細を表示するには、リストから子ワークフロー実行を選択し、[詳細を表示] ボタンを選択して [実行の詳細] ページを開きます。

> **Important**
>
> 子ワークフロー実行の保持ポリシーは 90 日間です。この保持期間を過ぎて完了した子ワークフロー実行は、[実行] テーブルには表示されません。これは、分散マップ状態または親ワークフローが保持期間を超えて実行され続けている場合でも同様です。`ResultWriter`

を使用して分散マップ状態出力を Amazon S3 バケットにエクスポートすると、これらの子ワークフローの実行の詳細 (結果を含む) を表示できます。

> **Tip**
>
> 更新ボタン
>
>
>
> を選択すると、すべての子ワークフロー実行の最新リストが表示されます。

マップ実行の再処理

失敗した子ワークフローは、親ワークフローのマップ実行で redriving できます。redriven された親ワークフローは、分散マップを含むすべての失敗状態を redrives します。親ワークフローが実行を完了したときに、ステートの `<stateType>Entered` イベントに対応する `<stateType>Exited` イベントがない場合、親ワークフローは失敗した状態を再処理します。例えば、イベント履歴に `MapStateEntered` イベントの `MapStateExited` イベントが含まれていない場合は、親ワークフローを再処理して、マップ実行で失敗した子ワークフローの実行をすべて再処理できます。

ステートマシンに ItemReader、ResultWriter、またはその両方にアクセスするために必要な権限がない場合、マップ実行は開始されないか、最初の実行試行に失敗します。親ワークフローの最初の実行時にマップ実行が開始されなかった場合、親ワークフローを再処理すると初めてマップ実行が開始されます。これを解決するには、必要なアクセス許可をステートマシンロールに追加し、次に親ワークフローを再処理します。必要なアクセス許可を追加せずに親ワークフローを再処理すると、新しいマップ実行の実行を開始しようとしますが、再び失敗します。必要なアクセス許可については、「分散マップ状態を使用するための IAM ポリシー」を参照してください。

トピック

- マップ実行での子ワークフローの対象を再処理する
- 子ワークフローの実行再処理動作
- マップ実行再処理で使用される入力のシナリオ
- マップ実行を再処理する IAM アクセス許可
- コンソールでのマップ実行の再処理
- RedrivingAPI を使用したマップ実行の再処理

マップ実行での子ワークフローの対象を再処理する

次の条件が満たされる場合、マップ実行で失敗した子ワークフローの実行を再処理できます。

- 2023 年 11 月 15 日以降に親ワークフローの実行を開始しました。この日付より前に開始した実行は再処理の対象外です。
- 特定のマップ実行の再処理のハードリミットである 1000 件を超えていません。この制限を超えると、`States.Runtime` エラーが表示されます。
- 親ワークフローは再処理可能です。親ワークフローが再処理可能でない場合、マップ実行で子ワークフローを再処理することはできません。ワークフローの再処理の対象については、「失敗した実行の再処理対象」を参照してください。
- マップ実行でのタイプ Standard の子ワークフロー実行は、実行イベント履歴の上限である 25,000 件を超えていません。イベント履歴の制限を超えた子ワークフローの実行は、許容される失敗しきい値にカウントされ、失敗したと見なされます。実行の再処理の対象については、「失敗した実行の再処理対象」を参照してください。

次の場合、元の実行試行でマップ実行が失敗しても、新しいマップ実行が開始され、既存のマップ実行は再処理されません。

- `States.DataLimitExceeded` エラーにより、マップ実行が失敗しました。
- JSON データ補間エラー、`States.Runtime` のため、マップ実行が失敗しました。例えば、OutputPath で存在しない JSON ノードを選択したとします。

マップ実行は、親ワークフローが停止またはタイムアウトした後も引き続き実行できます。これらのシナリオでは、再処理はすぐには実行されません。

- マップ実行は、進行中のタイプ Standard の子ワークフロー実行をキャンセルしているか、タイプ Express の子ワークフロー実行が完了するのを待っている可能性があります。
- 結果をエクスポートするように設定した場合、マップ実行は結果をまだ ResultWriter に書き込んでいる可能性があります。

このような場合、実行中のマップ実行は、再処理を試行する前に操作を完了します。

子ワークフローの実行再処理動作

マップ実行での再処理された子ワークフローの実行は、次の表に示すような動作を示します。

エクスプレス子ワークフロー	標準子ワークフロー
元の実行試行で失敗またはタイムアウトしたすべての子ワークフロー実行は、 StartExecution API アクションを使用して開始されます。ItemProcessor で最初に実行されたステートが最初に実行されます。	元の実行試行で失敗、タイムアウト、またはキャンセルされたすべての子ワークフロー実行は、RedriveExecution API アクションを使用して再処理されます。これらの子ワークフローは、 の最後の状態redrivenであり ItemProcessor 、実行に失敗しました。
失敗した実行はいつでも再処理できます。これは、Express 子ワークフローの実行は常に StartExecution API アクションを使用して新しい実行として開始されるためです。	失敗した標準子ワークフローの実行は、常に再処理できるわけではありません。実行が再処理可能でない場合、再試行されることはありません。実行の最後のエラーまたは出力は永続的です。これは、実行の履歴イベントが 25,000 件を超えるか、14 日間の再処理可能な期間が過ぎた場合に発生することがあります。 親ワークフローの実行が 14 日以内に終了し、子ワークフロー実行が 14 日より前に終了した場合、標準子ワークフローの実行は再処理できない可能性があります。
エクスプレス子ワークフローの実行では、元の実行試行と同じ実行 ARN が使用されますが、個々の再処理を明確に識別することはできません。	標準子ワークフロー実行では、元の実行試行と同じ実行 ARN が使用されます。やなどの APIsを使用して、 コンソールおよび GetExecutionHistory redrivesで個人を明確に識別できますDescribeExecution。詳細については、「再処理された実行内容を調べる」を参照してください。

マップ実行を再処理していて、同時実行数の制限に達した場合、そのマップ実行での子ワークフロー実行は保留ステートに移行します。マップ実行の実行ステータスも [保留中の redrive] ステートに移行します。指定した同時実行数の制限により子ワークフローの実行回数が増えるまで、実行は、[保留中の redrive] ステートのままになります。

例えば、ワークフロー内の分散マップの同時実行数の上限が 3000 件で、再実行される子ワークフローの数が 6000 件だとします。これにより、3000 件の子ワークフローが並行して実行され、残り

の 3000 件のワークフローは [保留中のリドライブ] ステートのままになります。3000 件の子ワークフローの最初のバッチが実行を完了すると、残りの 3000 件の子ワークフローが実行されます。

マップ実行の実行が完了するか中止されると、[保留中の redrive] ステートの子ワークフロー実行数は 0 にリセットされます。

マップ実行再処理で使用される入力のシナリオ

最初の実行時に分散マップにどのように入力したかに応じて、再処理されたマップ実行では次の表に示すように入力を使用します。

最初の実行試行時の入力	マップ実行再処理で使用された入力
前のステートから渡された入力または実行入力。	再処理されたマップ実行は同じ入力を使用します。
入力は ItemReader を使用して渡され、次の条件のいずれかが true であるため、マップ実行は子ワークフローの実行を開始しませんでした。 • マップ実行が States.ItemReaderFailed エラーで失敗しました。 • マップ実行が States.ResultWriterFailed エラーで失敗しました。 • マップ実行が開始される前に、親ワークフローの実行がタイムアウトになったか、キャンセルされました。	再処理されたマップ実行では Amazon S3 バケット内の入力を使用します。
を使用して渡された入力 ItemReader。子ワークフローの実行を開始または開始しようとした後にマップ実行が失敗しました。	再処理されたマップ実行は、元の実行時と同じ入力を使用します。

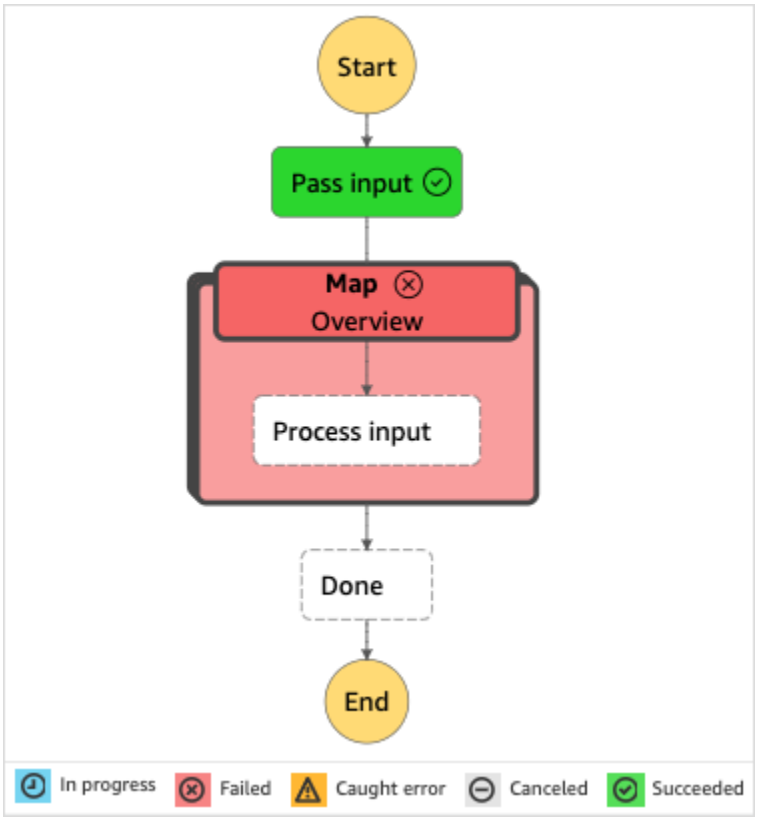
マップ実行を再処理する IAM アクセス許可

Step Functions には、マップ実行を再処理するための適切なアクセス許可が必要です。次の IAM ポリシーの例では、マップ実行の再処理に必要な最小限の権限をステートマシンに付与しています。#####のテキストを、リソース固有の情報に必ず置き換えてください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
}
```

コンソールでのマップ実行の再処理

以下のイメージは、分散マップを含むステートマシンの実行グラフを示しています。マップ実行が失敗したため、この実行は失敗しました。マップ実行を再処理するには、親ワークフローを再処理する必要があります。



コンソールからマップ実行を再処理するには

1. Step Functions コンソールを開き、実行に失敗した分散マップを含む既存のステートマシンを選択します。
2. ステートマシンの詳細ページの [実行] で、このステートマシンの失敗した実行インスタンスを選択します。
3. [再処理] を選択します。
4. [再処理] ダイアログボックスで [リドライブの実行] を選択します。

> **Tip**
>
> 実行の詳細ページまたはマップ実行の詳細ページからマップ実行を再処理することもできます。
> 実行の詳細ページを開いている場合は、次のいずれかを実行して実行を再処理します。
>
> - [復旧] を選択し、[障害からのリドライブ] を選択します。
> - [アクション]、[再処理] の順に選択します。
>
> マップ実行の詳細ページを開いている場合は、[復旧] を選択し、[障害からのリドライブ] を選択します。

再処理は、同じステートマシン定義と ARN を使用していることに注意してください。最初の実行試行で失敗したステップから実行を継続します。この例では、[マップ] という名前の分散マップステップとその中の [入力を処理する] ステップです。失敗したマップ実行の子ワークフロー実行を再開した後も、再処理は、[完了] ステップの実行を継続します。

5. 実行の詳細ページから [マップ実行] を選択すると、再処理されたマップ実行の詳細が表示されます。

   このページでは、redriven 実行の結果を表示できます。例えば、[マップ実行] 実行の概要 セクションには、マップ実行が再処理された回数を表す [リドライブ回数] が表示されます。[イベント] セクションでは、元の実行試行のイベントに追加された再処理関連の実行イベントを確認できます。例えば、`MapRunRedriven` イベントです。

redriven マップ実行が完了したら、コンソールで、または GetExecutionHistoryと DescribeExecution API アクションを使用して、そのredrive詳細を確認できます。redriven 実行の確認の詳細については、実行の詳細については、「再処理された実行内容を調べる」を参照してください。

RedrivingAPI を使用したマップ実行の再処理

親ワークフローの RedriveExecution API を使用して対象のマップ実行を再処理できます。この API は、マップ実行で失敗した子ワークフローの実行を再開します。

AWS Command Line Interface (AWS CLI) で、以下のコマンドを実行して、失敗したステートマシンの実行を再処理します。######のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

redriven マップ実行が完了したら、コンソールまたは DescribeMapRun API アクションを使用してそのredrive詳細を確認できます。マップ実行の標準ワークフロー実行redriveの詳細を調べるには、GetExecutionHistoryまたは DescribeExecution API アクションを使用できます。redriven 実行の確認の詳細については、実行の詳細については、「再処理された実行内容を調べる」を参照してください。

親ワークフローでログ記録を有効にしている場合は、Step Functions コンソールのマップ実行でエクスプレスワークフロー実行の再処理の詳細を確認できます。詳細については、「CloudWatch Logs を使用したログ記録」を参照してください。

Step Functions のエラー処理

Pass と Wait 状態を除くすべての状態でランタイムエラーが発生する可能性があります。エラーは、以下の例のようにさまざまな理由で発生する可能性があります。

- ステートマシンの定義の問題 (例えば、Choice 状態に一致するルールがない)。
- タスクの失敗 (例えば、AWS Lambda 関数の例外)。
- 一時的な問題 (例えば、ネットワークパーティションイベント)。

デフォルトでは、状態でエラーが報告されると、AWS Step Functions の実行全体が失敗します。

> **Tip**
>
> エラー処理を含むワークフローの例を AWS アカウント にデプロイするには、AWS Step Functions ワークショップの「モジュール 8 - エラー処理」を参照してください。

トピック

- エラー名
- エラー後の再試行
- Fallback 状態
- Retry と Catch を使用するステートマシンの例

エラー名

Step Functions は大文字と小文字を区別する文字列を使用して Amazon States Language でエラーを識別します。これをエラー名といいます。Amazon States Language は、よく知られているエラー名を付ける一連の組み込み文字列を定義します。すべて `States.` プレフィックスが付いています。

`States.ALL`

あらゆる既知のエラー名に一致するワイルドカード。

> **Note**
>
> このエラータイプは、`States.DataLimitExceeded` ターミナルエラータイプとランタイムエラータイプをキャッチできません。これらのエラータイプについては、「`States.DataLimitExceeded`」および「`States.Runtime`」を参照してください。

`States.DataLimitExceeded`

Step Functions では、以下の条件で `States.DataLimitExceeded` 例外が報告されます。

- コネクタの出力がペイロードサイズクォータより大きい場合。
- ステートの出力がペイロードサイズクォータより大きい場合。
- `Parameters` 処理の後、状態の入力がペイロードサイズクォータよりも大きい場合。

クォータに関する詳細については、クォータ を参照してください。

> **Note**
>
> これは States.ALL エラータイプでは検出できないターミナルエラーです。

States.ExceedToleratedFailureThreshold

失敗したアイテムの数がステートマシン定義で指定されたしきい値を超えたため、Map 状態が失敗しました。詳細については「分散マップ状態の許容される失敗しきい値」を参照してください。

States.HeartbeatTimeout

HeartbeatSeconds 値より長時間実行されたため ハートビートの送信に失敗した Task 状態。

> **Note**
>
> このエラーは、Catch および Retry フィールド内でのみ表示されます。

States.IntrinsicFailure

ペイロードテンプレート内で組み込み関数を呼び出そうとしましたが、失敗しました。

States.ItemReaderFailed

ItemReader フィールドに指定されたアイテムソースから読み取れなかったため、Map 状態が失敗しました。詳細については「ItemReader」を参照してください。

States.NoChoiceMatched

Choice 状態が選択ルールで定義された条件と入力を一致させることができず、デフォルト遷移も指定されていません。

States.ParameterPathFailure

状態の Parameters フィールド内で、名前の末尾がパスを使用する .$ になっているフィールドを置換しようとしても失敗します。

States.Permissions

指定されたコードの実行に必要な特権が足りないため Task 状態が失敗しました。

States.ResultPathMatchFailure

Step Functions は、 状態が受け取った入力に状態の ResultPath フィールドを適用できませんでした。

States.ResultWriterFailed

ResultWriter フィールドに指定された宛先に結果を書き込めなかったため、Map 状態が失敗しました。詳細については「ResultWriter」を参照してください。

States.Runtime

処理できない例外のため、実行に失敗しました。多くの場合、これらは実行時のエラー (null JSON ペイロードに InputPath または OutputPath を適用しようとしたなど) によって発生します。States.Runtime エラーは再試行可能ではなく、常に実行が失敗する原因になります。States.ALL での再試行またはキャッチでは States.Runtime エラーはキャッチされません。

States.TaskFailed

実行中に失敗した Task 状態。リトライやキャッチで使用すると、States.TaskFailed は、States.Timeout 以外のあらゆる既知のエラー名に一致するワイルドカードとして機能します。

States.Timeout

TimeoutSeconds 値より長時間実行されたか、HeartbeatSeconds 値より長い間隔ハートビートの送信に失敗した Task 状態。

さらに、ステートマシンが指定された TimeoutSeconds 値より長く実行されると、実行は States.Timeout エラーで失敗します。

状態は別の名前でエラーを報告することがあります。ただし、これらのエラー名は、States. プレフィックスから始めることはできません。

ベストプラクティスとして、本番稼働用コードが AWS Lambda のサービス例外 (Lambda.ServiceException および Lambda.SdkClientException) を処理できることを確認します。詳細については「Lambda サービスの例外を処理する」を参照してください。

> **Note**
>
> Lambda での未処理のエラーは、エラー出力で Lambda.Unknown として報告されます。これには、メモリ不足エラーと関数のタイムアウトが含まれま

す。Lambda.Unknown、States.ALL、または States.TaskFailed を一致させて、こういったエラーに処理できます。Lambda が最大呼び出し数に達すると、エラーは Lambda.TooManyRequestsException となります。Lambda 関数のエラーの詳細については、「AWS Lambda デベロッパーガイド」の「エラー処理と自動再試行」を参照してください。

エラー後の再試行

Task、Parallel、Map の状態は Retry という名前のフィールドを持つことができます。その値は Retrier と呼ばれるオブジェクトの配列にする必要があります。個々の retrier は特定回数の再試行を表し、通常は時間間隔が増加していきます。

いずれかの状態がエラーを報告し、Retry フィールドがある場合、Step Functions は配列にリストされた順序で Retrier をスキャンします。Retrier の ErrorEquals フィールドの値にエラー名が表示されると、ステートマシンは Retry フィールドで定義されているとおりに再試行します。

再試行を定義した、タスク、並行、または インラインマップステートを redriven で再実行すると、redrive での再試行回数が最大になるように、これらのステートの再試行回数は 0 にリセットされます。redriven 実行では、コンソールを使用してこれらのステートでの再試行を個別に追跡できます。詳細については、「」の 再処理された実行の再試行動作 を参照してください。

retrier には以下のフィールドがあります。

> **Note**
>
> 再試行は状態遷移として扱われます。状態遷移が課金に及ぼす影響については、Step Functions コストを参照してください。

ErrorEquals (必須)

エラー名に一致する空でない文字列の配列です。状態がエラーを報告すると、Step Functions は再試行全体をスキャンします。エラー名がこの配列に表示されている場合、この retrier に記述されている再試行ポリシーを実装します。

IntervalSeconds (オプション)

最初の再試行前の秒数を表す整数 (デフォルトで 1)。IntervalSeconds には 99999999 の最大値があります。

MaxAttempts (オプション)

再試行の最大回数を表す正の整数 (デフォルトでは 3)。エラーが指定された回数を超えて再発する場合は、再試行が停止され通常のエラー処理が再開されます。0 の値は、エラーが再試行されないことを指定します。MaxAttempts には 99999999 の最大値があります。

BackoffRate (オプション)

各試行間の後に IntervalSeconds で表される再試行間隔が増加する乗数。デフォルトでは、BackoffRate 値は 2.0 ずつ増加します。

例えば、IntervalSeconds が 3、MaxAttempts が 3、BackoffRate が 2 だとします。最初の再試行は、エラーが発生した 3 秒後に行われます。2 回目の再試行は、1 回目の再試行の 6 秒後に行われます。また、3 回目の再試行は 2 回目の再試行の 12 秒後に行われます。

MaxDelaySeconds (オプション)

再試行間隔を延長できる最大値を秒単位で設定する正の整数。このフィールドは BackoffRate フィールドと併用すると便利です。このフィールドで指定する値によって、連続する再試行ごとにバックオフ率の乗数が適用されることによる指数関数的な待機時間を制限できます。MaxDelaySeconds には 0 より大きく 31622401 より小さい値を指定する必要があります。

この値を指定しない場合、Step Functions は再試行間の待機時間を制限しません。

JitterStrategy (オプション)

連続する再試行間の待機時間にジッターを含めるかどうかを決定する文字列。ジッターは、ランダムな遅延間隔にわたって再試行回数を分散させて同時再試行回数を減らします。この文字列は、FULL または NONE を値として受け入れます。デフォルト値は、「NONE」です。

例えば、MaxAttempts を 3、IntervalSeconds を 2、BackoffRate を 2 に設定したとします。最初の再試行は、エラーが発生した 2 秒後に行われます。2 回目の再試行は 1 回目の再試行の 4 秒後に行われ、3 回目の再試行は 2 回目の再試行の 8 秒後に行われます。JitterStrategy を FULL に設定した場合、1 回目の再試行間隔は 0～2 秒の間でランダム化され、2 回目の再試行間隔は 0～4 秒の間でランダム化され、3 回目の再試行間隔は 0～8 秒の間でランダム化されます。

再試行フィールドの例

このセクションには、次の Retry フィールドの例が含まれます。

- Retry with BackoffRate
- Retry with MaxDelaySeconds
- Retry all errors except States.Timeout
- Complex retry scenario

> **Tip**
>
> エラー処理ワークフローの例を AWS アカウント にデプロイするには、AWS Step Functions ワークショップの「エラー処理」を参照してください。

例 1 - BackoffRate を使用して再試行する

次の Retry の例では、2 回の再試行を行い、3 秒待ってから最初の再試行します。Step Functions は、指定した BackoffRate に基づいて、再試行の最大回数に達するまで各再試行の間隔を増やします。次の例では、1 回目の再試行から 3 秒待ってから 2 回目の再試行が開始されます。

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "IntervalSeconds": 3,
   "MaxAttempts": 2,
   "BackoffRate": 1
} ]
```

例 2 - MaxDelaySeconds を指定して再試行する

次の例では、再試行を 3 回行い、BackoffRate から生じる待機時間を 5 秒に制限しています。最初の再試行は 3 秒待ってから行われます。2 回目と 3 回目の再試行は、MaxDelaySeconds で設定されている最大待機時間制限により、前回の再試行から 5 秒間待ってから行われます。

```
"Retry": [ {
    "ErrorEquals": [ "States.Timeout" ],
    "IntervalSeconds": 3,
    "MaxAttempts": 3,
    "BackoffRate":2,
    "MaxDelaySeconds": 5,
    "JitterStrategy": "FULL"
} ]
```

MaxDelaySeconds を使用しない場合、2 回目の再試行は 1 回目の再試行の 6 秒後に行われ、3 回目の再試行は 2 回目の再試行の 12 秒後に行われます。

例 3 - States.Timeout を除くすべてのエラーを再試行する

retrier の ErrorEquals フィールドにあらかじめ表示される名前 States.ALL は、すべてのエラー名に一致するワイルドカードです。ErrorEquals 配列では単独で表示される必要があり、Retry 配列では最後の retrier に表示される必要があります。名前 States.TaskFailed もワイルドカードの役割を果たし、States.Timeout を除くあらゆるエラーに一致します。

Retry フィールドの例では、States.Timeout を除くすべてのエラーを再試行します。

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "MaxAttempts": 0
}, {
   "ErrorEquals": [ "States.ALL" ]
} ]
```

例 4 - 複雑な再試行シナリオ

retrier のパラメータは、単一状態実行のコンテキストの retrier に対するすべてのアクセスに適用されます。

次の Task 状態の場合を考えてみます。

```
"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"
   } ]
}
```

このタスクは 4 回連続で失敗し、エラー名 `ErrorA`、`ErrorB`、`ErrorC`,`ErrorB` を出力します。結果として以下が発生します。

- 最初の 2 つのエラーが最初の retrier に一致し、1 秒および 2 秒の待機が発生します。
- 3 番目のエラーが 2 番目の retrier に一致し、5 秒の待機が発生します。
- 4 番目のエラーも最初の retrier に一致します。ただし、その特定のエラーに対する再試行の最大回数 2 回 (`MaxAttempts`) に達しています。そのため、その retrier は失敗し、実行によってワークフローが `Catch` フィールドを通じて `Z` 状態にリダイレクトされます。

Fallback 状態

`Task`、`Map`、`Parallel` 状態はそれぞれ `Catch` というフィールドを持つことができます。このフィールドの値は、catchers と言われるオブジェクトの配列である必要があります。

catcher には以下のフィールドがあります。

`ErrorEquals` (必須)

同じ名前の retrier フィールドで指定されたエラー名と正確に一致する空ではない文字列。

`Next` (必須)

ステートマシンの状態名のいずれかに正確に一致する必要がある文字列。

`ResultPath` (オプション)

`Next` フィールドに指定された状態に catcher が送信する入力を決定するパス。

状態がエラーを報告し、`Retry` フィールドがないか、再試行でエラーが解決できなかった場合、Step Functions は配列にリストされた順序で catcher をスキャンします。エラー名が catcher の `ErrorEquals` フィールドの値に表示される場合、ステートマシンは `Next` フィールドに名前がある状態に移行します。

catcher の `ErrorEquals` フィールドにあらかじめ表示される名前 `States.ALL` は、すべてのエラー名に一致するワイルドカードです。`ErrorEquals` 配列では単独で表示される必要があ

り、Catch 配列では最後の catcher に表示される必要があります。名前 States.TaskFailed もワイルドカードの役割を果たし、States.Timeout を除くあらゆるエラーに一致します。

Catch フィールドの次の例では、Lambda 関数が処理されない Java 例外を出力した場合に RecoveryState という名前の状態に移行します。それ以外の場合は、フィールドは EndState 状態に移行します。

```
"Catch": [ {
   "ErrorEquals": [ "java.lang.Exception" ],
   "ResultPath": "$.error-info",
   "Next": "RecoveryState"
}, {
   "ErrorEquals": [ "States.ALL" ],
   "Next": "EndState"
} ]
```

> **Note**
>
> 各 catcher には処理するエラーを複数指定できます。

エラー出力

Step Functions が catch 名に指定された状態に移行する場合、オブジェクトには通常 Cause というフィールドが含まれます。このフィールドの値は人間が読んで理解できるエラーの説明です。このオブジェクトはエラー出力といいます。

この例では、最初の catcher に ResultPath フィールドが含まれています。これは状態の最上位の ResultPath フィールドに似た動作を行い、次のいずれかの結果になります。

- 状態の実行の結果を取得して、状態の入力のすべてまたは一部を上書きします。
- 結果を取得して入力に追加します。catcher によって処理されたエラーの場合、状態の実行の結果はエラー出力です。

したがって、この例の最初の catcher の場合、catcher がエラー出力が error-info という名前のフィールドとして入力に追加します (入力にこの名前のフィールドが存在しない場合)。次に、catcher は入力全体を RecoveryState に送信します。2 番目 catcher では、エラー出力によって入力が上書きされ、catcher はエラー出力のみを EndState に送信します。

> **Note**
>
> ResultPath フィールドを指定しない場合、入力全体を選択して上書きする $ がデフォルトで設定されます。

状態に Retry と Catch フィールドの両方がある場合、Step Functions は最初に適切な Retrier を使用します。リトライポリシーがエラーを解決できなかった場合、Step Functions は一致する catcher の移行を適用します。

ペイロードとサービス統合の原因

キャッチャーは、文字列ペイロードを出力として返します。Amazon Athena や AWS CodeBuild のようなサービス統合を使って作業する場合、Cause 文字列を JSON に指定します。組み込み関数を使用した Pass 状態の次の例は、Cause 文字列を JSON に変換する方法を示しています。

```
"Handle escaped JSON with JSONtoString": {
  "Type": "Pass",
  "Parameters": {
    "Cause.$": "States.StringToJson($.Cause)"
  },
  "Next": "Pass State with Pass Processing"
},
```

Retry と Catch を使用するステートマシンの例

次の例で定義されたステートマシンには、2 つの Lambda 関数があるとします。1 つは常に失敗し、1 つはステートマシンで定義されたタイムアウトが発生するのに十分な時間待機する許可をだします。

これは、常に失敗してメッセージ error を返す Node.js Lambda 関数の定義です。続くステートマシンの例では、この Lambda 関数の名前は FailFunction です。Lambda 関数の作成の詳細については、「ステップ 1: Lambda 関数を作成する」セクションを参照してください。

```
exports.handler = (event, context, callback) => {
    callback("error");
};
```

これは、10 秒間スリープする Node.js Lambda 関数の定義です。続くステートマシンの例では、この Lambda 関数の名前は sleep10 です。

> **Note**
>
> Lambda コンソールでこの Lambda 関数を作成するときは、[Advanced settings] (アドバンスト設定) セクションの [Timeout] (タイムアウト) 値を 3 秒 (デフォルト) から 11 秒に変更してください。

```
exports.handler = (event, context, callback) => {
    setTimeout(function(){
    }, 11000);
};
```

Retry を使用して失敗を処理する

このステートマシンは Retry フィールドを使用して、失敗してエラー名 HandledError を出力する関数を再試行します。この関数は 2 回再試行され、再試行間にはエクスポネンシャルバックオフが使用されます。

```
{
   "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
      }
   }
}
```

このバリエーションでは、Lambda 関数が出力するあらゆるエラーに一致する事前定義されたエラーコード States.TaskFailed を使用します。

```
{
   "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**
>
> ベストプラクティスとしては、Lambda 関数をリファレンスするタスクが Lambda サービス例外を処理する必要があります。詳細については「Lambda サービスの例外を処理する」を参照してください。

Catch を使用して失敗を処理する

この例では、Catch フィールドを使用します。Lambda 関数がエラーを出力すると、そのエラーをキャッチし、ステートマシンは fallback 状態に移行します。

```
{
   "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
      }
   }
}
```

このバリエーションでは、Lambda 関数が出力するあらゆるエラーに一致する事前定義されたエラーコード States.TaskFailed を使用します。

```
{
   "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
      }
   }
}
```

Retry を使用してタイムアウトを処理する

このステートマシンは Retry フィールドを使用して、TimeoutSeconds で指定されたタイムアウト値に基づいてタイムアウトした Task 状態を再試行します。Step Functions は、この Task 状態で

Lambda 関数の呼び出しを 2 回再試行し、再試行間にはエクスポネンシャルバックオフが使用されます。

```
{
   "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
      }
   }
}
```

Catch を使用してタイムアウトを処理する

この例では、Catch フィールドを使用します。タイムアウトが発生すると、ステートマシンは fallback 状態に移行します。

```
{
   "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**
>
> 状態入力とエラーを保持するには、ResultPath を使用します。ResultPath を使用して、エラーと入力の両方を に含める Catch を参照してください。

他のサービスから AWS Step Functions の呼び出し

ステートマシンを呼び出すため他の複数のサービスを構成できます。ステートマシンのワークフロータイプに基づいて、ステートマシンを非同期または同期的で呼び出すことができます。ステートマシンを同期で呼び出すには、StartSyncExecution API コールを使用するか、Amazon API Gateway を Express ワークフローと統合します。非同期呼び出しでは、Step Functions はタスクトークンが返されるまでワークフローの実行を一時停止します。ただし、タスクトークンを待っていると、ワークフローが同期します。

Step Functions を呼び出すように構成できるサービスには、次のものがあります。

- AWS Lambda、StartExecution を呼び出しを使います。
- Amazon API Gateway
- Amazon EventBridge
- AWS CodePipeline
- AWS IoT ルールエンジン
- AWS Step Functions

Step Functions 呼び出しには、StartExecution クォータが適用されます。詳細については、以下を参照してください。

- クォータ

Step Functions 読み取り整合性

AWS Step Functions でのステートマシンの更新は、結果的に整合します。数秒以内のすべての `StartExecution` 呼び出しでは、更新された定義と `roleArn` (IAM ロールの Amazon リソースネーム (ARN)) が使用されます。`UpdateStateMachine` を呼び出したすぐ後に開始された実行では、前のステートマシン定義と `roleArn` が使用されることがあります。

詳細については、次を参照してください。

- `UpdateStateMachine` (AWS Step Functions API リファレンス)
- AWS Step Functions の開始方法 でワークフローを更新します。

Step Functions でのタグ付け

AWS Step Functions は、ステートマシン (標準と Express の両方) とアクティビティのタグ付けをサポートしています。これにより、リソースに関連するコストを追跡して管理したり、AWS Identity and Access Management (IAM) ポリシーのセキュリティを向上させることができます。Step Functions リソースにタグ付けすると、それらを AWS Resource Groups で管理することができます。Resource Groups 詳細については、AWS Resource Groups ユーザーガイドを参照してください。

次の例のように、タグベースの認証の場合、ステートマシンの実行リソースはステートマシンに関連付けられたタグを継承します。

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

実行リソース ARN を指定した DescribeExecution またはその他の API を呼び出すと、Step Functions はステートマシンに関連付けられたタグを使用して、タグベースの認証を実行している間にリクエストを許可または拒否します。これにより、ステートマシンレベルの実行へのアクセスを許可または拒否できます。

リソースのタグ付けに関連する制限を確認するには、タグ付けに関連する制限 を参照してください。

トピック

- コスト割り当てのタグ付け
- セキュリティのためのタグ付け
- Step Functions コンソールでのタグの表示と管理

- Step Functions API アクションを使用してタグを管理

コスト割り当てのタグ付け

コスト割り当てのために Step Functions リソースを整理して識別するために、ステートマシンやアクティビティの目的を識別するメタデータタグを追加できます。これはリソースが多数ある場合に特に便利です。コスト配分タグを使用してAWSの請求書を整理し、自分のコスト構造に反映できます。そのためには、サインアップして AWS アカウントの請求書にタグキーおよび値を含めます。詳細については、AWS Billing ユーザーガイドの月別コスト配分レポートの設定を参照してください。

例えば、次のようにコストセンターと Step Functions リソースの目的を表すタグを追加できます。

リソース	キー	値
StateMachine1	Cost Center	34567
	Application	Image processing
StateMachine2	Cost Center	34567
	Application	Rekognition processing
Activity1	Cost Center	12345
	Application	Legacy database

このタグ付けスキームを使用すると、関連するタスクを実行している 2 つのステートマシンを同じコストセンターでグループ化しながら、関連のないアクティビティに異なるコスト割り当てタグでタグ付けすることができます。

セキュリティのためのタグ付け

IAM は、タグに基づくリソースへのアクセスの制御をサポートしています。タグに基づいてアクセスを制御するには、IAM ポリシーの条件要素でリソースタグに関する情報を提供します。

例えば、キー `environment` および値 `production` のタグを含むすべての Step Functions リソースへのアクセスを制限できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "states:TagResource",
                "states:DeleteActivity",
                "states:DeleteStateMachine",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

詳細については、IAM ユーザーガイドの [Controlling Access Using Tags] (タグを使用したアクセス制御) を参照してください。

Step Functions コンソールでのタグの表示と管理

Step Functions を使用すると、Step Functions コンソールでステートマシンのタグを表示および管理できます。ステートマシンの [Details] (詳細) ページから、[Tags] (タグ) を選択します。ここでは、ステートマシンに関連付けられた既存のタグが表示されます。

> **Note**
>
> アクティビティのタグを管理するには、Step Functions API アクションを使用してタグを管理 を参照してください。

ステートマシンに関連付けられたタグを追加または削除するには、[Manage Tags] (タグの管理) ボタンを選択します。

1. ステートマシンの詳細ページを参照します。
2. [Tags] (タグ) 次に、[Executions] (実行) そして [Definition] (定義) を選択します。
3. [Manage tags] (タグの管理) を選択します。

- 既存のタグを変更するには、[キー] と [値] フィールドを編集します。
- 既存のタグを削除するには、[Remove tag] (タグの削除) を選択します。
- 新しいタグを追加するには、[Add tag] (タグの追加) を選択し、[Key] (キー) と [Value] (値) を入力します。

4. [Save] (保存) を選択します。

Step Functions API アクションを使用してタグを管理

Step Functions API を使用してタグを管理するには、次の API アクションを使用します。

- ListTagsForResource
- TagResource
- UntagResource

AWS Step Functions ワークフロースタジオ

Workflow Studio for AWS Step Functions は、 AWS サービスをオーケストレーションしてサーバーレスワークフローを作成できる、ローコードのビジュアルワークフローデザイナーです。 drag-and-drop その機能または組み込みのコードエディターを使用して、ワークフローの作成と編集、状態ごとの入力と出力のフィルターまたは変換方法の制御、エラー処理の設定を行うことができます。状態をドラッグアンドドロップしてワークフローを構築すると、Workflow Studio によって作業が検証され、コードが自動生成されます。コードエディタ内で生成されたコードを確認またはステートマシンの定義を更新できます。完了したら、ワークフローを保存して実行し、Step Functions コンソールで結果を調べることができます。ワークフローを視覚的に追加および変更して、アプリケーション内の複数のサービスを調整できます。

Step Functions Workflow Studio を使用するには AWS アカウント、使用するリソースに対して適切な権限を提供する認証情報、、および認証情報が必要です。詳細については、「AWS Step Functions のスタート方法の前提条件」を参照してください。

> **Note**
>
> Workflow Studio は Internet Explorer 11 をサポートしていません。Internet Explorer 11 を使用しており、Workflow Studio を使用して問題が発生した場合は、別のブラウザをお試しください。

Workflow Studio には Step Functions コンソールからアクセスして、Step Functions でワークフローを作成または編集します。AWS Application Composer 内のワークフロースタジオにアクセスすることもできます。Application Composer の Workflow Studio にはビジュアル IaC 環境が用意されているため、AWS CloudFormation テンプレートなどの IaC ツールを使用して構築されたサーバーレスアプリケーションにワークフローを簡単に組み込むことができます。Application Composer の Workflow Studio を使用すると、AWS CloudFormation テンプレートを使用してワークフローを構築できます。Application Composer では、新しいワークフローを追加したり、既存のワークフローを変更したり、個々のワークフローステップを他のアプリケーションリソースに接続したりできます。Application Composer は必要な CloudFormation リソースと設定を自動的に作成して更新します。これにより、ワークフローで使用されるすべてのリソースを 1 か所で作成、管理できます。また、ワークフローのプロトタイピングから本番環境へのデプロイまでの時間を短縮するのにも役立ちます。

Application Composer で Workflow Studio を使用すると、ローカルプロジェクトに直接接続することもできます。これにより、ビジュアルキャンバスと並行して統合開発環境 (IDE) で作業できます。詳細については、「Application Composer の Workflow Studio を使用する」を参照してください。

トピック

- インターフェースの概要
- Workflow Studio を使用する
- 状態の入力と出力を構成する
- Workflow Studio での実行ロール
- エラー処理
- チュートリアル: AWS Step Functions Workflow Studio 使用を学ぶ

インターフェースの概要

Workflow Studio for AWS Step Functions は、オーケストレーションによってサーバーレスワークフローを作成できる、ローコードのビジュアルワークフローデザイナーです。 AWS のサービス Workflow Studio にはドラッグアンドドロップ機能があり、ワークフロープロトタイプの構築、編集、可視化を簡単に行うことができます。Workflow Studio には、Step Functions コンソール内の Amazon ステートメント言語 (ASL) を使用してワークフロー定義を記述および編集するための組み込みコードエディタも用意されています。

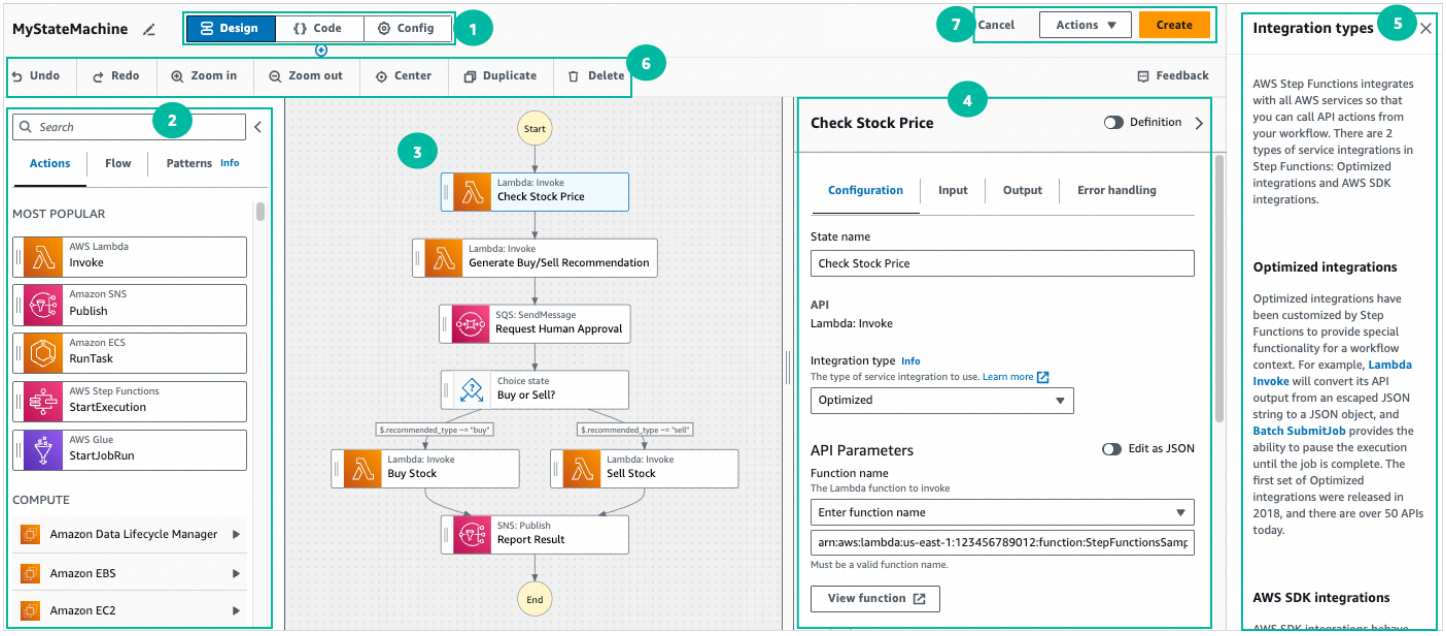
ワークフローの構築と可視化、定義の編集、構成の管理に役立つように、Workflow Studio には[デザイン]、[コード]、[設定] の 3 つのモードがあります。以下のセクションで、これらのモデルについて詳しく説明します。

このトピックの内容

- デザインモード
- コードモード
- 設定モード
- キーボードショートカット

デザインモード

Workflow Studio の [デザイン] モードには、プロトタイプを作成する際にワークフローを可視化するグラフィカルインターフェイスがあります。以下のイメージは、[デザイン] モードで使用できるさまざまなコンポーネントを示しています。

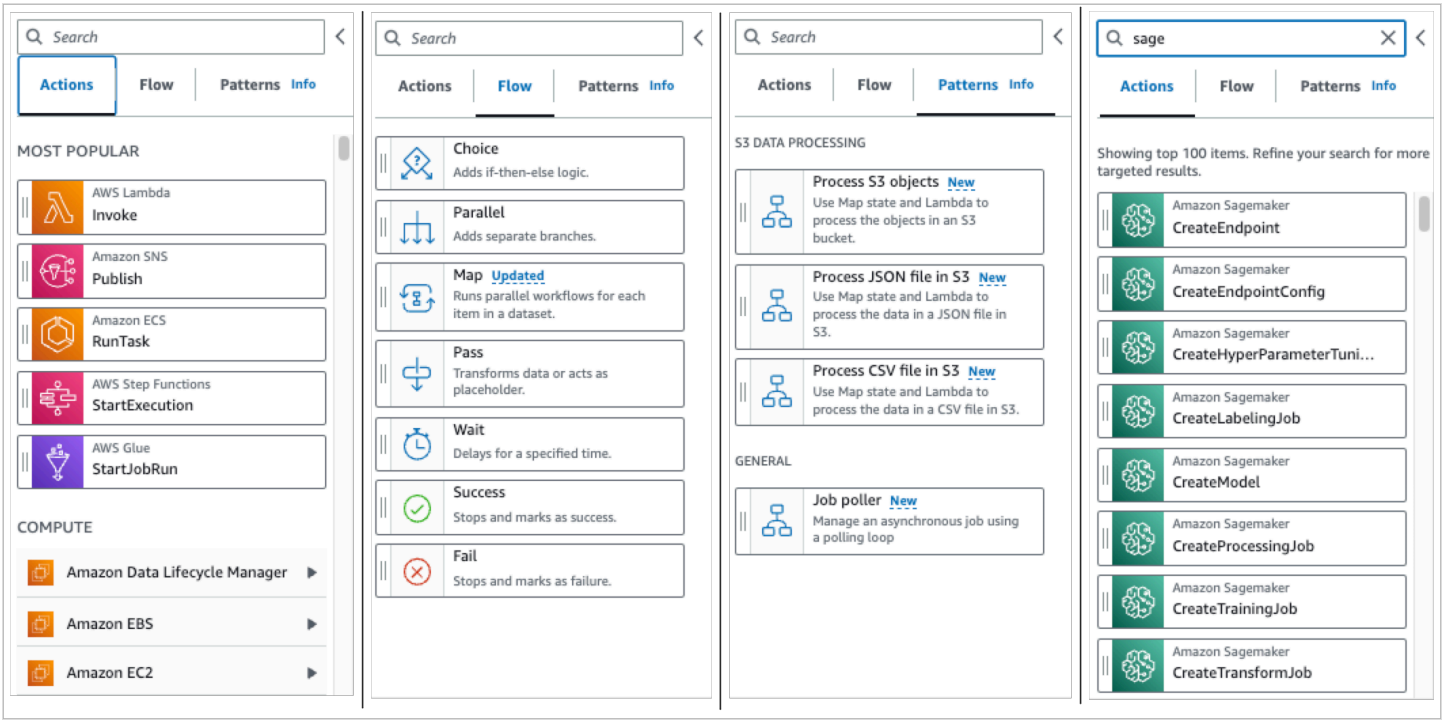


1. モードボタン - モードボタンを使用して Workflow Studio の [デザイン] モード、[コード] モード、および [設定] モードを切り替えます。ワークフローの ASL 定義内の JSON が無効な場合、モードを切り替えることはできません。
2. [State browser] (状態ブラウザ) には次の 3 つのタブがあります。
   - Actions タブには AWS API のリストが表示され、キャンバスのワークフローグラフにドラッグアンドドロップできます。各アクションは タスク 状態を示します。
   - [フロー] タブは、キャンバス内のワークフローのグラフにドラッグアンドドロップできるフロー状態のリストを表示します。
   - 「パターン」タブには ready-to-use、さまざまなユースケースに使用できる再利用可能なビルディングブロックがいくつか用意されています。例えば、これらのパターンを使用して Amazon S3 バケットのデータを繰り返し処理できます。
3. キャンバス は、状態をワークフローのグラフにドラッグアンドドロップし、状態の順序を変更します。状態を選択して、状態を設定または表示に選択します。
4. Inspector パネルでは、キャンバス上で選択した任意の状態のプロパティを表示および編集できます。[定義] トグルをオンにすると、現在選択されているステートが強調表示された状態で、ワークフローの Amazon States Language コードが表示されます。

5. [Info] (情報) リンクは、ヘルプが必要になるとコンテキスト情報を含むパネルを開きます。これらのパネルには、Step Functions ドキュメントの関連トピックへのリンクも含まれます。
6. デザインツールバー - 元に戻す、削除、ズームインなどの一般的なアクションを実行するボタンのセットが含まれています。
7. ユーティリティボタン - ワークフローの保存や、ASL 定義を JSON ファイルや YAML ファイルにエクスポートするなどのタスクを実行するためのボタンのセットです。

[State browser] (状態ブラウザ)

[State browser] (状態ブラウザ) は、ワークフローのグラフにドラッグアンドドロップする状態を選択する場所です。「アクション」タブには AWS API のリストが表示され、「フロー」タブにはフロー状態のリストが表示されます。一方、「パターン」タブには ready-to-use、さまざまなユースケースに使用できる再利用可能なビルディングブロックがいくつか用意されています。上部の検索ボックスを使用して、[状態ブラウザ] ですべての状態を検索できます。



ワークフローの指示と制御に使用できるフロー状態は 7 つあります。そのすべては前の状態から入力を受け取りますが、前の状態からの入力と後続の状態への出力をフィルターできないものが一部あります。フロー状態:

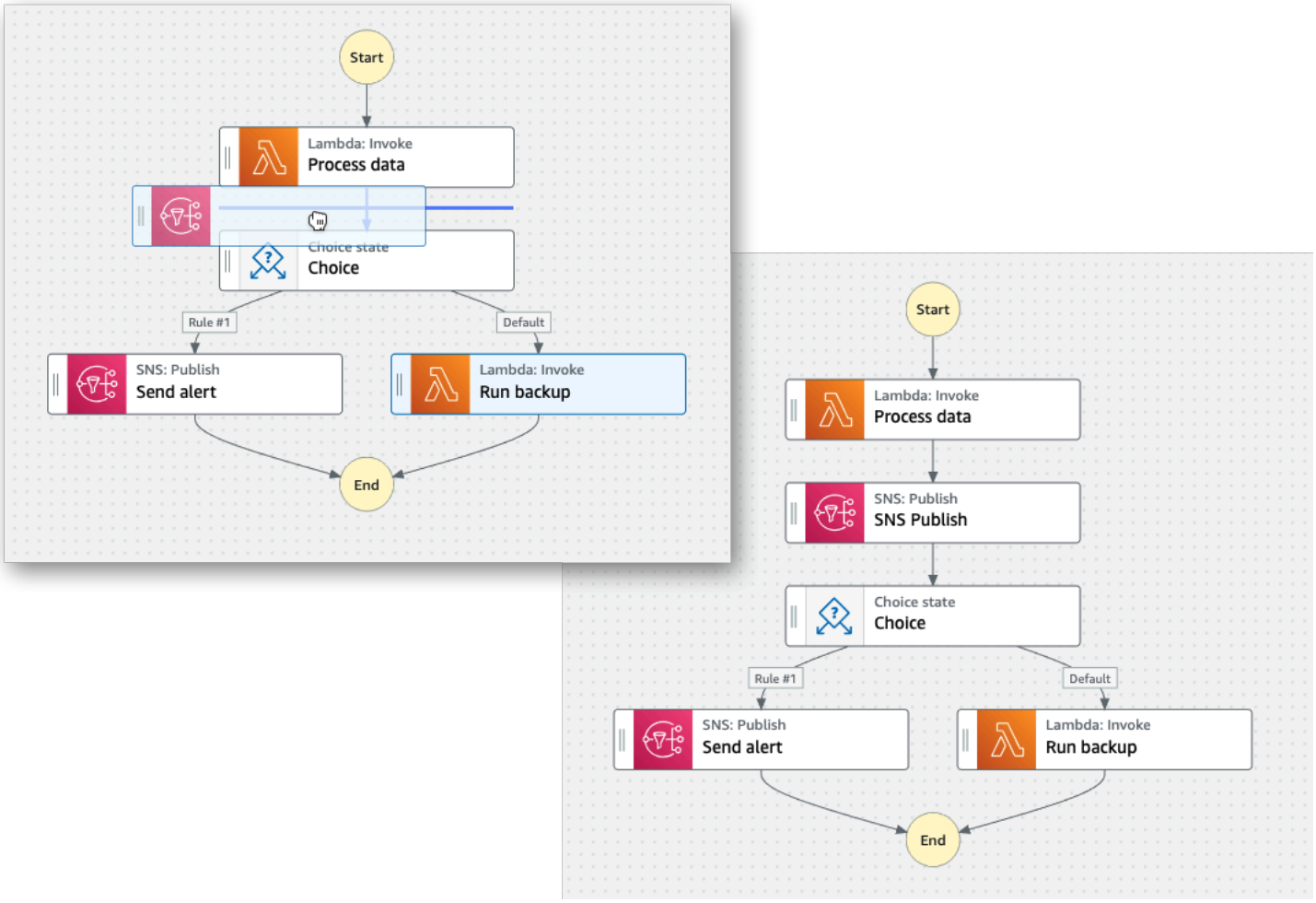
- 選択: ワークフローに実行の分岐点で行う選択を追加します。Inspector の [Configuration] (設定) タブでは、どの状態にワークフローを移行するのかを決定するルールを設定できます。

- 並行: ワークフローに実行の並列分岐を追加します。
- マッピング: 入力配列の各要素のステップを動的に反復します。Parallel フロー状態とは違い、Map 状態は、状態入力にある配列の複数エントリに対して同じステップを実行します。
- パス: 入力を出力に渡すことができます。(オプション) 出力に固定データを追加できます。
- 待機: ワークフローを一定の時間、指定した時刻または日付まで一時停止します。
- 成功: 成功したためワークフローを停止します。
- 失敗: エラーがでたため、ワークフローを停止します。

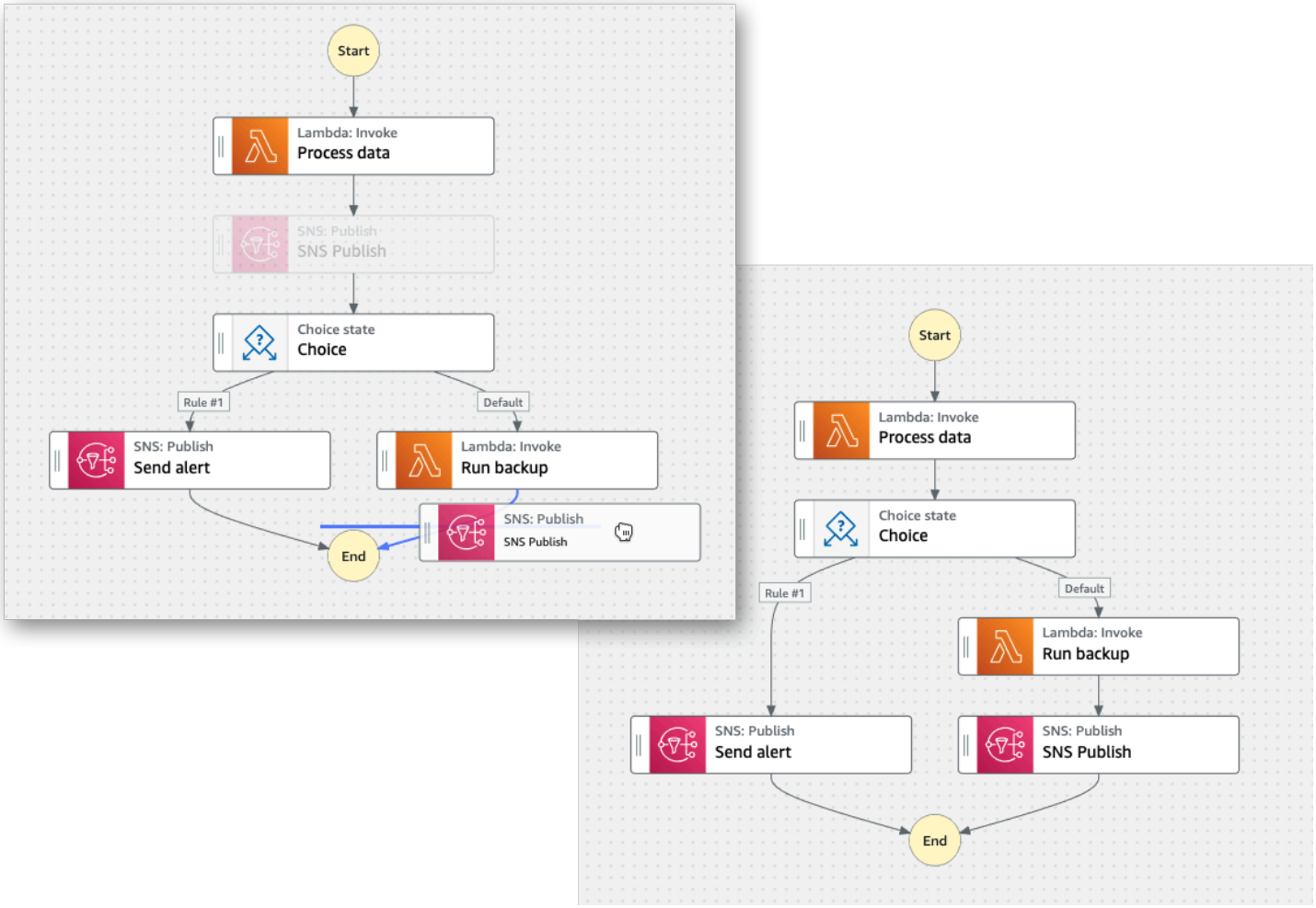
キャンバス

ワークフローに追加する状態を選択した後、キャンバスにドラッグし、ワークフローのグラフにドロップします。状態をドラッグアンドドロップして、ワークフローの別の場所に移動することもできます。ワークフローが複雑な場合は、キャンバスパネルでワークフローをすべて表示できない場合があります。キャンバスの上部にあるコントロールを使用して、拡大/縮小します。ワークフローのグラフのさまざまな部分を表示するには、キャンバスでワークフローのグラフをドラッグします。

ワークフローの状態を [アクション] または [フロー] パネルからドラッグして、ワークフローにドロップします。ラインはワークフローのどこに配置されるかを示します。新しいワークフロー状態がワークフローに追加され、そのコードが自動生成されます。

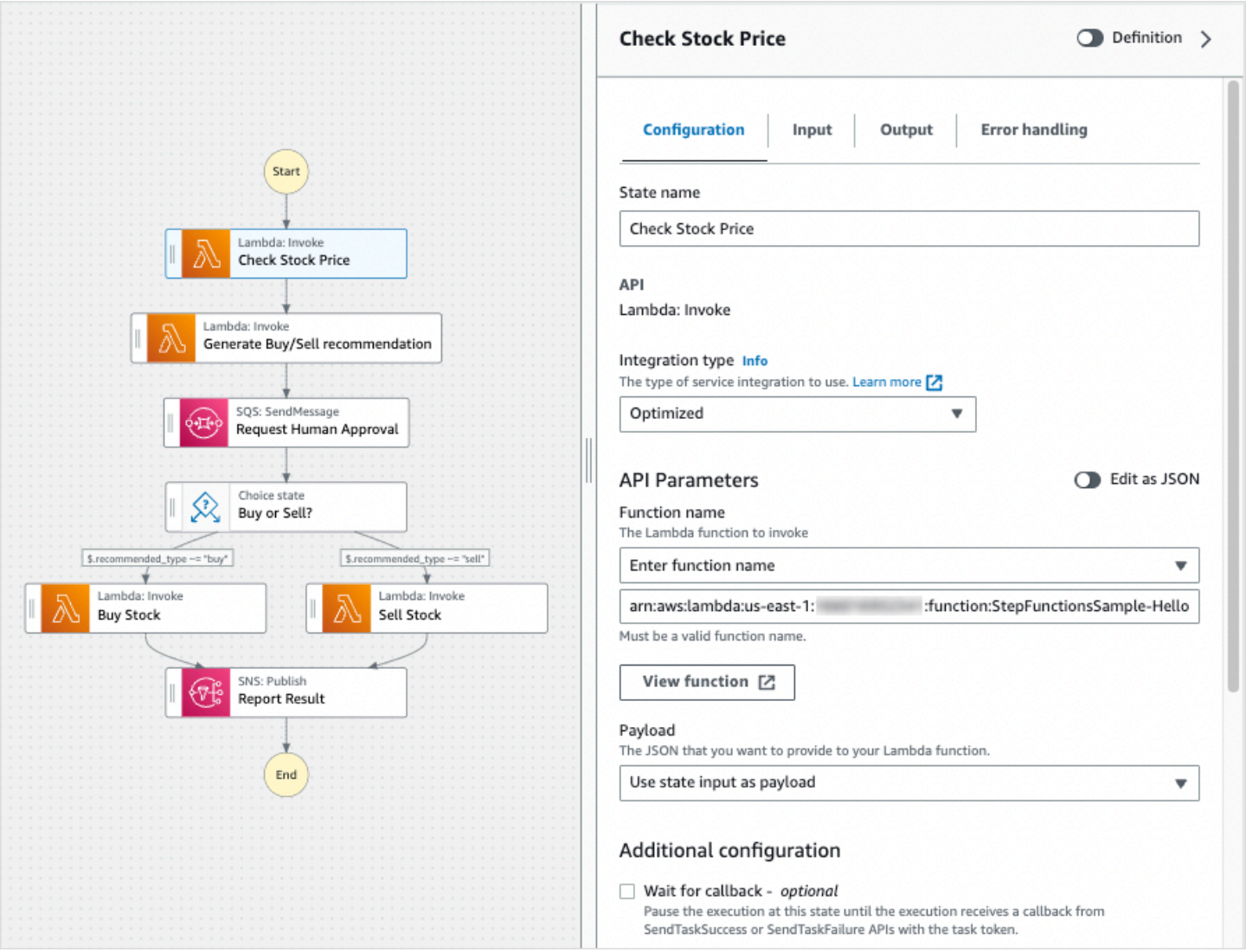


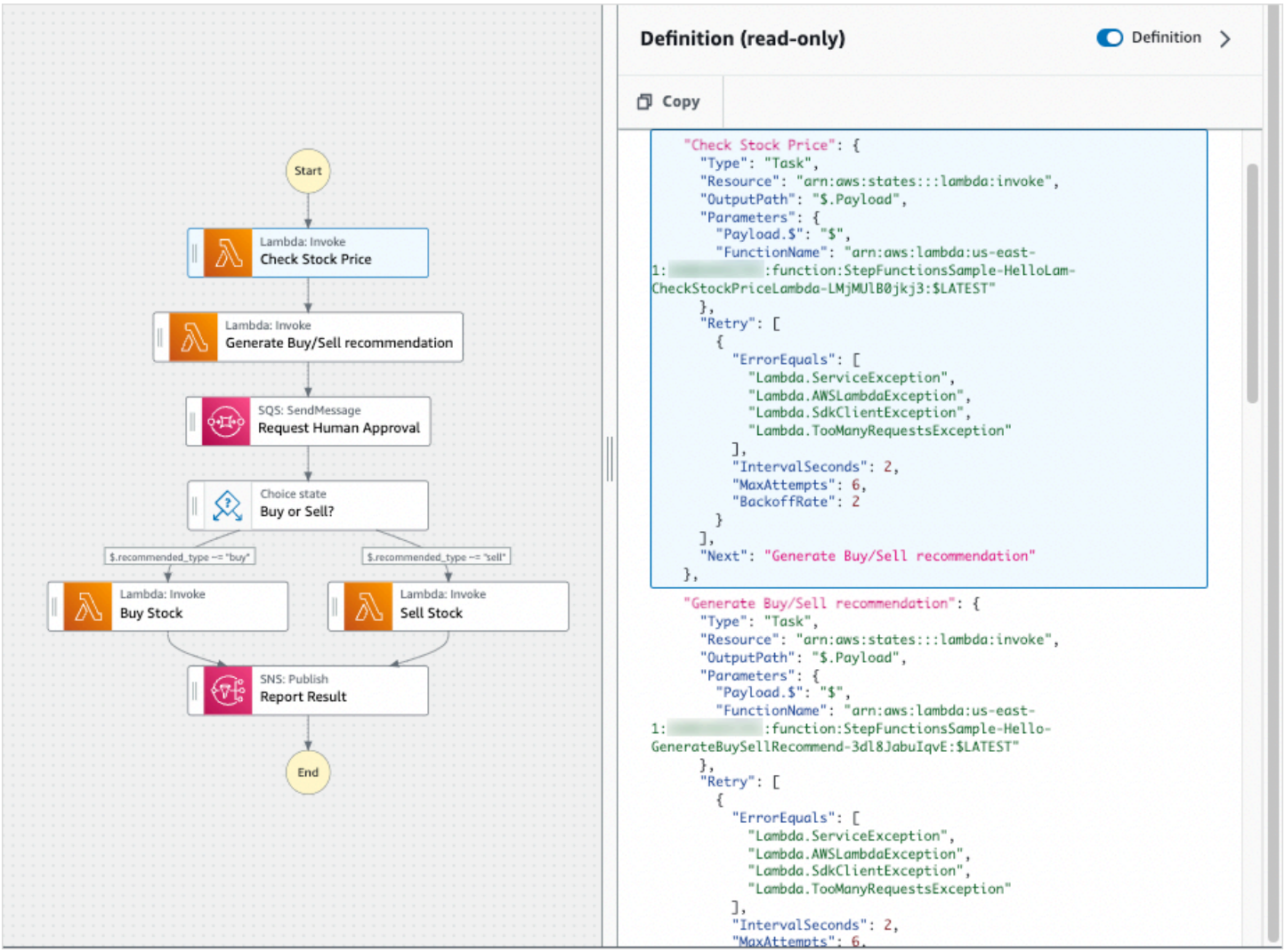
状態の順序を変更するには、ワークフロー内の別の場所にドラッグします。



Inspector

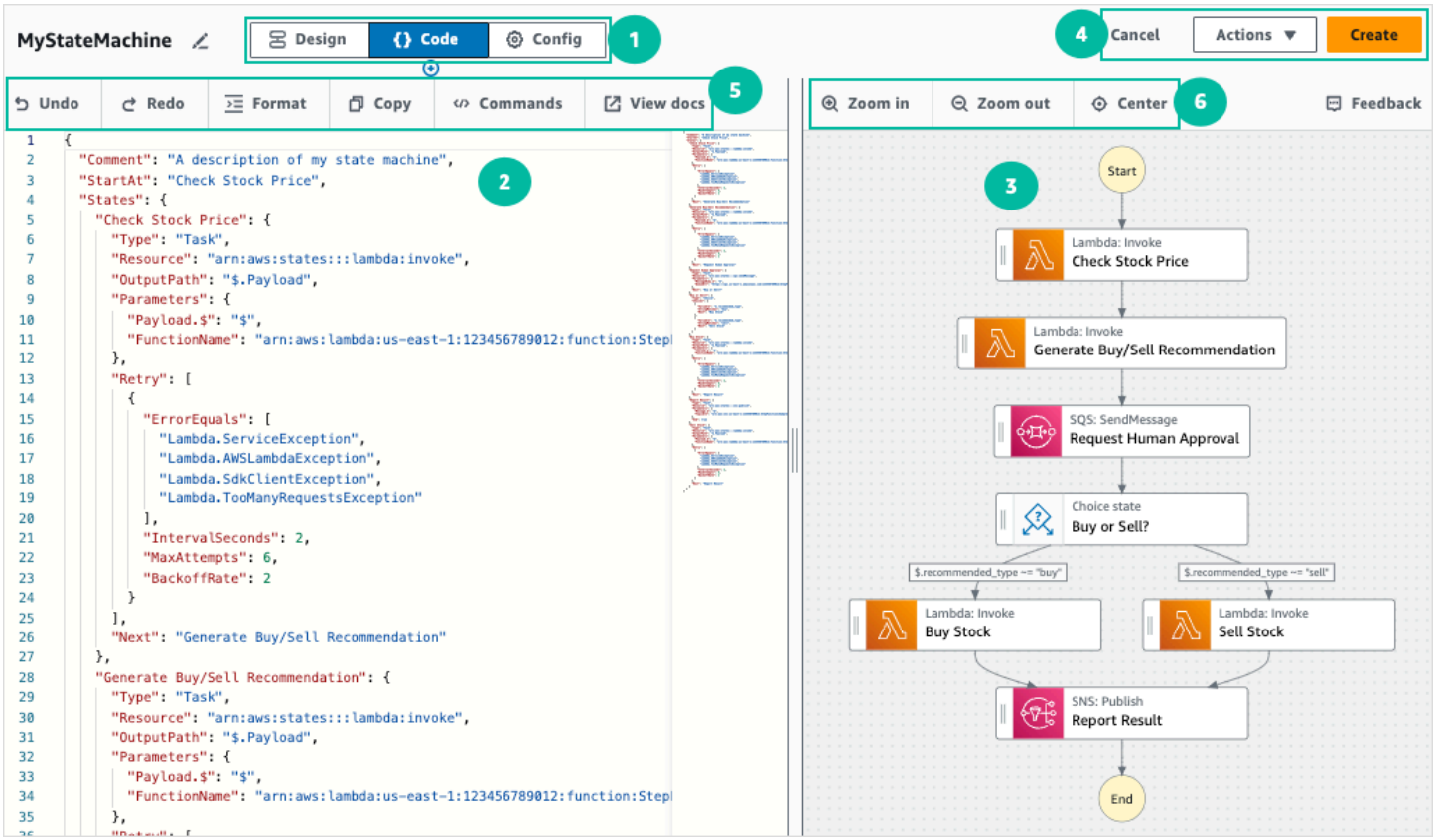
ワークフローに追加するステートはどれでも設定できます。設定する状態を選択すると、[Inspector] パネルにその設定オプションが表示されます。ワークフローコード用に自動生成された ASL 定義を確認するには、[定義] トグルをオンにします。選択した状態に関連付けられている ASL 定義がハイライトされます。


Start
Lambda: Invoke
Check Stock Price
Lambda: Invoke
Generate Buy/Sell recommendation
SQS: SendMessage
Request Human Approval
Choice state
Buy or Sell?
$.recommended_type == "buy"
$.recommended_type == "sell"
Lambda: Invoke
Buy Stock
Lambda: Invoke
Sell Stock
SNS: Publish
Report Result
End
Check Stock Price
Definition
Configuration
Input
Output
Error handling
State name
Check Stock Price
API
Lambda: Invoke
Integration type Info
The type of service integration to use. Learn more
Optimized
API Parameters
Edit as JSON
Function name
The Lambda function to invoke
Enter function name
arn:aws:lambda:us-east-1:
:function:StepFunctionsSample-Hello
Must be a valid function name.
View function
Payload
The JSON that you want to provide to your Lambda function.
Use state input as payload
Additional configuration
Wait for callback - optional
Pause the execution at this state until the execution receives a callback from SendTaskSuccess or SendTaskFailure APIs with the task token.



コードモード

Workflow Studio の[コード] モードには、Step Functions コンソール内でワークフローの Amazon ステートメント言語 (ASL) 定義を表示、記述、編集するための統合コードエディタが用意されています。以下のイメージは、[コード] モードで使用できるさまざまなコンポーネントを示しています。



1. モードボタン - モードボタンを使用して Workflow Studio の [デザイン] モード、[コード] モード、および [設定] モードを切り替えます。ワークフローの ASL 定義内の JSON が無効な場合、モードを切り替えることはできません。
2. ここで コードエディタ は、Workflow Studio 内でワークフローの ASL 定義を記述して編集します。コードエディタには、構文の強調表示やオートコンプリートなどの機能もあります。
3. グラフ可視化 - ワークフローをリアルタイムでグラフィカルに可視化します。
4. ユーティリティボタン - ワークフローの保存や、ASL 定義を JSON ファイルや YAML ファイルにエクスポートするなどのタスクを実行するためのボタンのセットです。
5. コードツールバー - アクションの取り消しやコードのフォーマットなど、一般的なアクションを実行するためのボタンのセットが含まれています。
6. グラフツールバー - ワークフローグラフのズームインやズームアウトなど、一般的なアクションを実行するためのボタンセットが含まれています。

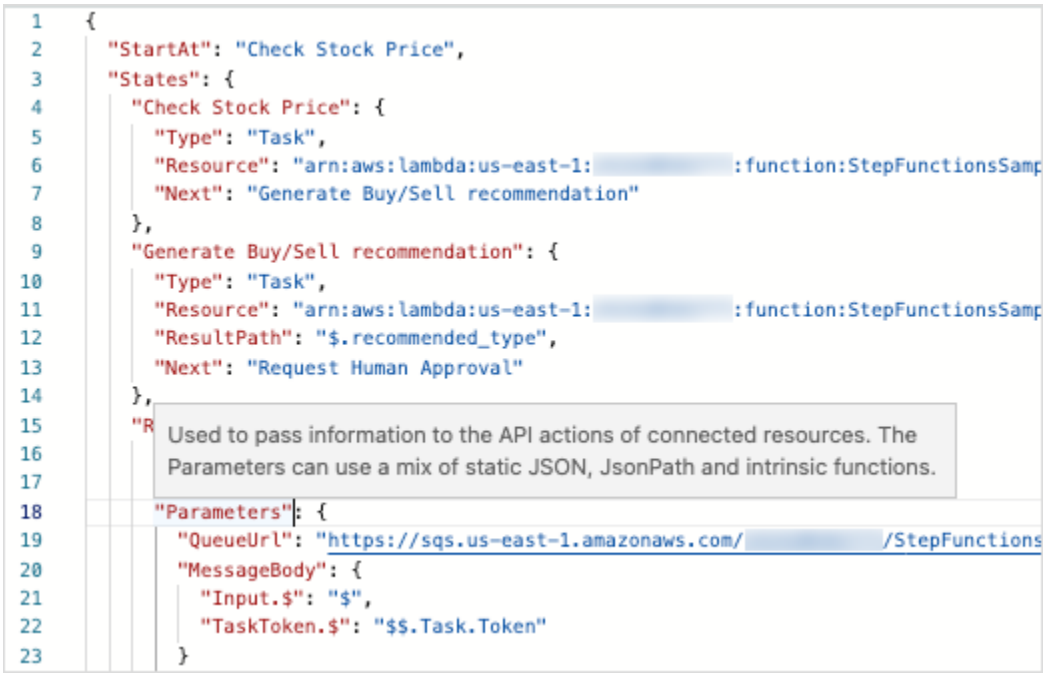
コードエディタ

コードエディタは IDE と同様の操作性で、Workflow Studio 内で JSON を使用してワークフロー定義を記述および編集できます。コードエディタには、構文の強調表示、オートコンプリートの提

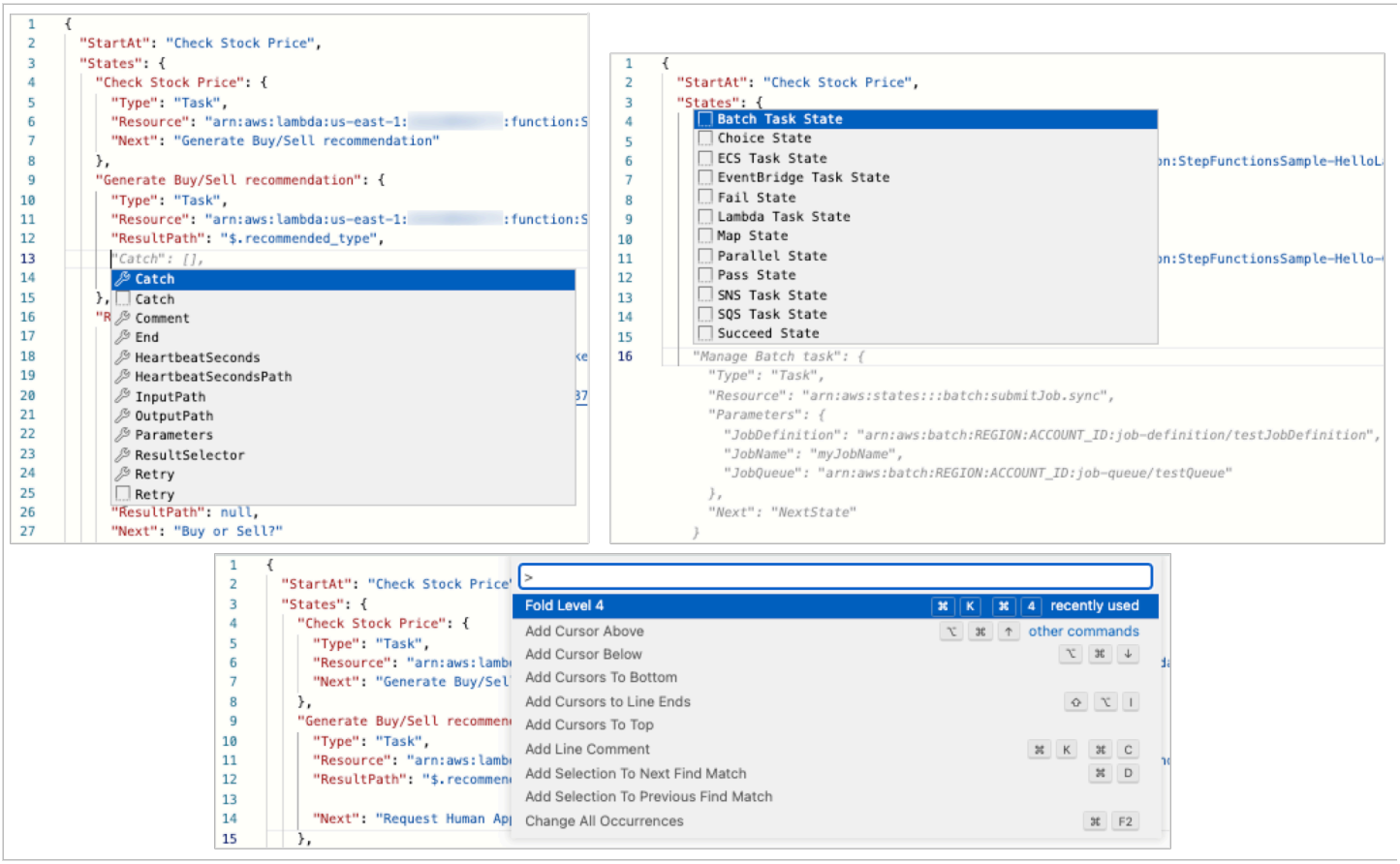
案、ASL 定義の検証、状況依存ヘルプ表示など、いくつかの機能が含まれています。ワークフロー定義を更新すると、グラフ可視化 はワークフローのリアルタイムグラフをレンダリングします。更新されたワークフローグラフは、デザインモード でも確認できます。

デザインモード またはグラフ表示ペインでステートを選択すると、そのステートの ASL 定義がコードエディタで強調表示されます。[デザイン] モードまたはグラフ可視化ペインでステートの並べ替え、削除、または追加を行うと、ワークフローの ASL 定義が自動的に更新されます。

ワークフロー定義内の任意のフィールドにカーソルを置くと、コンテキスト依存ヘルプがツールチップとして表示されます。



オートコンプリート候補には、ワークフローに含めることができるフィールドまたはステートのコードスニペットが表示されます。特定のステートに含めることができるフィールドのリストを表示するには、**Ctrl+Space** を押します。ワークフロー内の新しいステートのコードスニペットを生成するには、現在のステートの定義の後に **Ctrl+Space** を押します。**F1** を押して、使用できるコマンドのリストを表示することもできます。



グラフ可視化

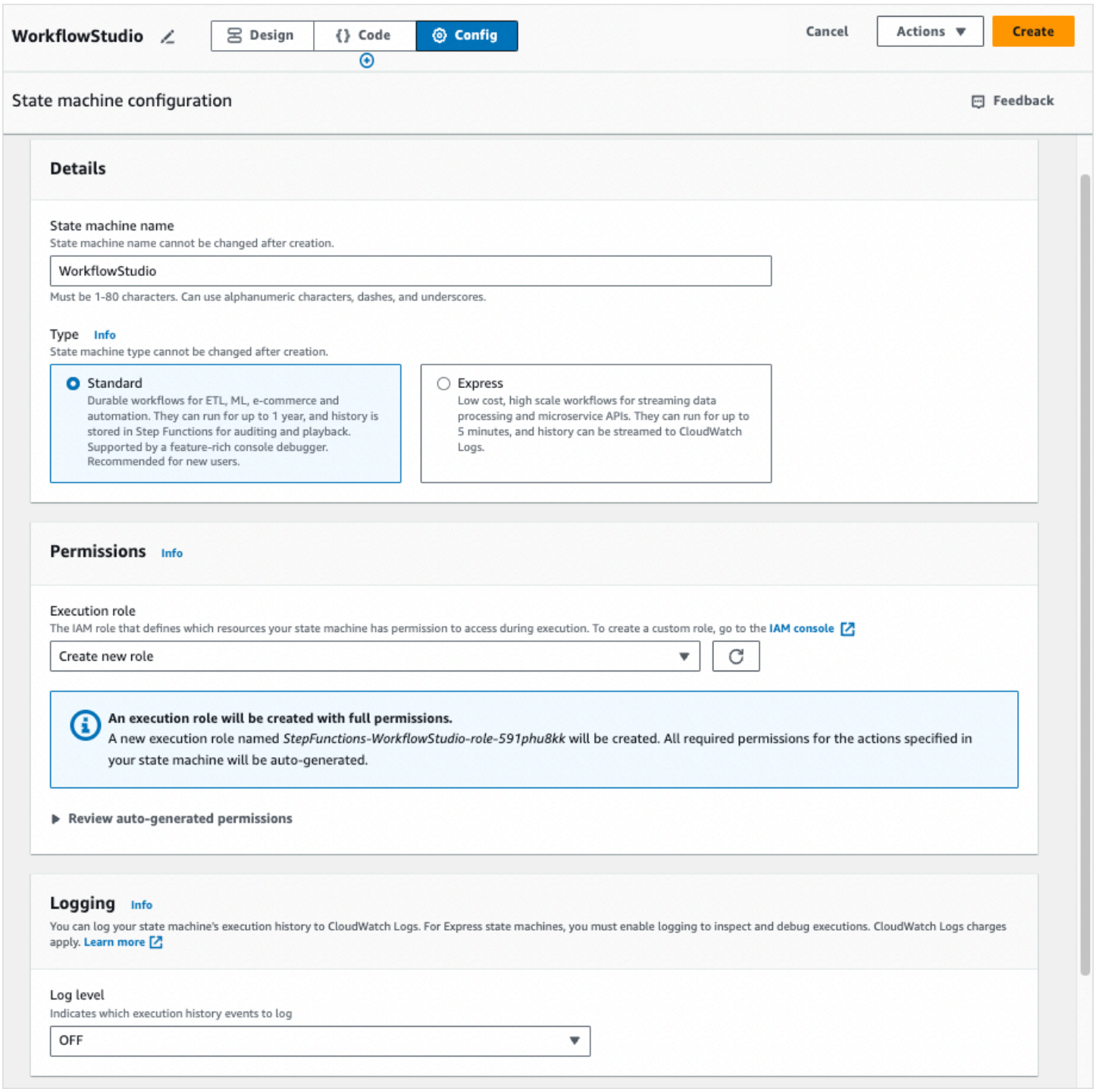
グラフ可視化を使用すると、ワークフローの外観をグラフ形式で確認できます。Workflow Studio の コードエディタ でワークフロー定義を記述すると、グラフ表示ペインにワークフローのリアルタイムグラフが表示されます。可視化ペインでステートの順序を変更、削除、または複製すると、コードエディタのワークフロー定義が自動的に更新されます。同様に、コードエディタでワークフロー定義の更新、順序の変更、削除、ステートの追加を行うと、可視化も自動的に更新されます。

ワークフローの ASL 定義内の JSON が無効な場合、グラフ可視化ペインはレンダリングを一時停止し、ペインの下部にステータスメッセージを表示します。

設定モード

Workflow Studio の設定 モードでは、ステートマシンの設定を管理できます。このモードでは、ステートマシンの名前とタイプ、IAM アクセス許可、ステートマシンのログ記録設定などの詳細を指定できます。このモードで指定できるその他の設定には、 AWS X-Ray ステートマシンの作成時にトレースを有効にしたり、バージョンを公開したりすることが含まれます。ステートマシンを作成した

ら、ステートマシンの名前とタイプを除くすべてのステートマシン設定オプションを編集できます。次のイメージは、[設定] モードで指定できる設定の一部を示しています。



ステートマシンの設定の管理

ステートマシンの設定を管理するには、以下を実行します。

1. ステートマシンの名前を [ステートマシン名] ボックスに入力します。

> **Tip**
>
> または、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシン設定] で名前を指定します。

> **Important**
>
> ステートマシンを作成した後でステートマシン名を編集することはできません。

2. [タイプ] で、ステートマシンのタイプを [スタンダード] または [エクスプレス] から選択します。ステートマシンの詳細のタイプについては、「標準ワークフロー対 Express ワークフロー」を参照してください。

> **Important**
>
> ステートマシンを作成した後でステートマシンのタイプを編集することはできません。

3. [アクセス許可] で、ステートマシンの実行ロールとして使用する IAM ロールを選択します。

   - [新しいロールを作成] (推奨): このオプションを選択すると、Step Functions はステートマシンの作成時に必要最小限の権限でステートマシンの実行ロールを自動的に作成します。これらの自動生成された IAM ロールは、 AWS リージョン ステートマシンを作成した時点で有効です。

> **Tip**
>
> Step Functions がステートマシンに対して自動的に生成するアクセス許可を確認するには、[自動生成されたアクセス許可をレビューする] を選択します。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

- [既存のロールを選択]: ステートマシン用に独自の IAM ロールを作成し、[既存のロールを選択] の下に表示されているオプションから選択します。ロールのポリシーに、ステートマシンに引き継がせたいアクセス許可が含まれていることを確認してください。

  IAM ポリシーの作成の詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

- [ロールの ARN を入力]: このステートマシンに使用する既存の IAM ロールの Amazon リソースネーム (ARN) を指定します。例えば `arn:aws:iam::123456789012:role/service-role/StepFunctions-WorkflowStudio-role-777f4027` です。

4. [ログ記録] で、ステートマシンのログレベルを設定します。Step Functions は、選択内容に基づいて実行履歴イベントを記録します。次のいずれかのオプションを選択できます。

   - [すべて]: すべてのイベントタイプが記録されます。
   - ERROR: TaskFailed ExecutionFailed やなど、すべてのエラーイベントタイプが記録されます。
   - FATAL: やなど ExecutionAborted 、すべての致命的エラーイベントタイプが記録されます。ExecutionFailed
   - [オフ]: イベントタイプはログに記録されません。

   ログレベルの詳細については、「ログレベル」を参照してください。

5. [追加設定] で、以下のオプション設定を 1 つ以上設定します。

   - [X-Ray トレースの有効化]: Step Functions は、トレース ID がアップストリームサービスによって渡されない場合でも、ステートマシンを実行するためにトレースを X-Ray に送信します。詳細については、「AWS X-Ray および Step Functions」を参照してください。
   - [作成時にバージョンを発行]: バージョンとは、実行可能なステートマシンの番号が付けられた変更不可能なスナップショットです。このチェックボックスを選択すると、ステートマシンの作成時にステートマシンのバージョンを発行できます。Step Functions は、ステートマシンの最初のリビジョンとしてバージョン 1 を発行しています。

     バージョンの詳細については、「ステートマシンバージョン」を参照してください。

   - [新しいタグを追加]: ステートマシンにタグを追加するには、このボックスを選択します。タグを追加すると、リソースに関連するコストを追跡して管理し、IAM ポリシーのセキュリティを向上させることができます。タグの詳細については、Step Functions でのタグ付けを参照してください。

6. [作成] を選択します。

7. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

[ロールの設定を表示] を選択して、[設定] モードに戻ることもできます。

キーボードショートカット

Workflow Studio では、以下のキーボードショートカットキーがサポートされています。

キーボードショートカット	機能
Shortcuts for the Code mode	
Ctrl+space	Auto-complete suggestions
F1	Display a list of available commands
Common shortcuts for the Design and Code modes	
Ctrl+Z	Undo the last operation
Ctrl+Shift+Z	Redo the last operation
Alt+C	Center the workflow in the canvas
Backspace	Remove all selected states
Delete	Remove all selected states
Ctrl+D	Duplicate selected state

Workflow Studio を使用する

Step Functions Workflow Studio を使用してワークフローを作成、編集、実行する方法を学習します。ワークフローの準備ができたら、ワークフローをエクスポートできます。Workflow Studio を使用してラピッドプロトタイプ作成を行うこともできます。

このトピックの内容

- ワークフローの作成
- ワークフローの設計
- ワークフローを実行

- ワークフローの編集
- ワークフローのエクスポート
- ワークフロープロトタイプの作成

ワークフローの作成

Workflow Studio では、スターターテンプレートを選択するか、空白のテンプレートを選択してワークフローを最初から作成できます。空白のテンプレートの場合は、[デザイン] モードまたは [コード] モードを使用してワークフローを作成できます。

スターターテンプレートは、ワークフローのプロトタイプと定義を自動的に作成し、 ready-to-run AWS プロジェクトに必要なすべての関連リソースをデプロイするサンプルプロジェクトです。AWS アカウントこれらのスターターテンプレートをそのままデプロイして実行することも、ワークフロープロトタイプを使用してその上に構築することもできます。スターターテンプレートの詳細については、「Step Functions サンプルプロジェクト」を参照してください。

スターターテンプレートを使用してワークフローを作成する

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで、次のいずれかを実行してサンプルプロジェクト ([タスクタイマー] サンプルプロジェクトなど) を選択します。
   - [キーワードで検索] ボックスに **`Task Timer`** と入力し、返された検索結果から[タスクタイマー] を選択します。
   - 右側のペインの [すべて] に一覧表示されているサンプルプロジェクトを参照し、[タスクタイマー] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。
5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。デザインモード では、[State browser] (状態ブラウザ) からステートをドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、コードモード に切り替えてワークフローの Amazon ステートメント言語 (ASL) 定義を更新してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  > **Note**
  >
  > これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用されます。

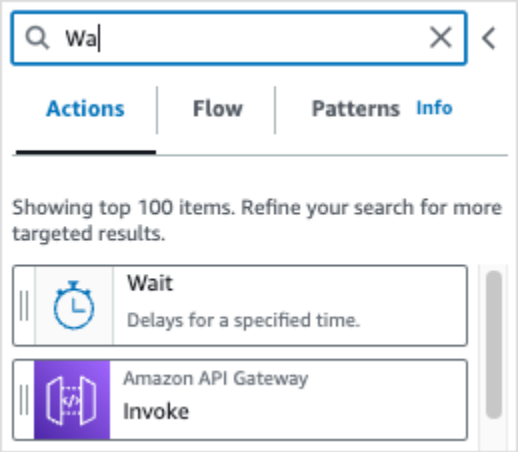
空白のテンプレートを使用してワークフローを作成します。

1. Step Functions コンソールを開きます。
2. Create State Machine (ステートマシンの作成) を選択します。
3. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
4. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。

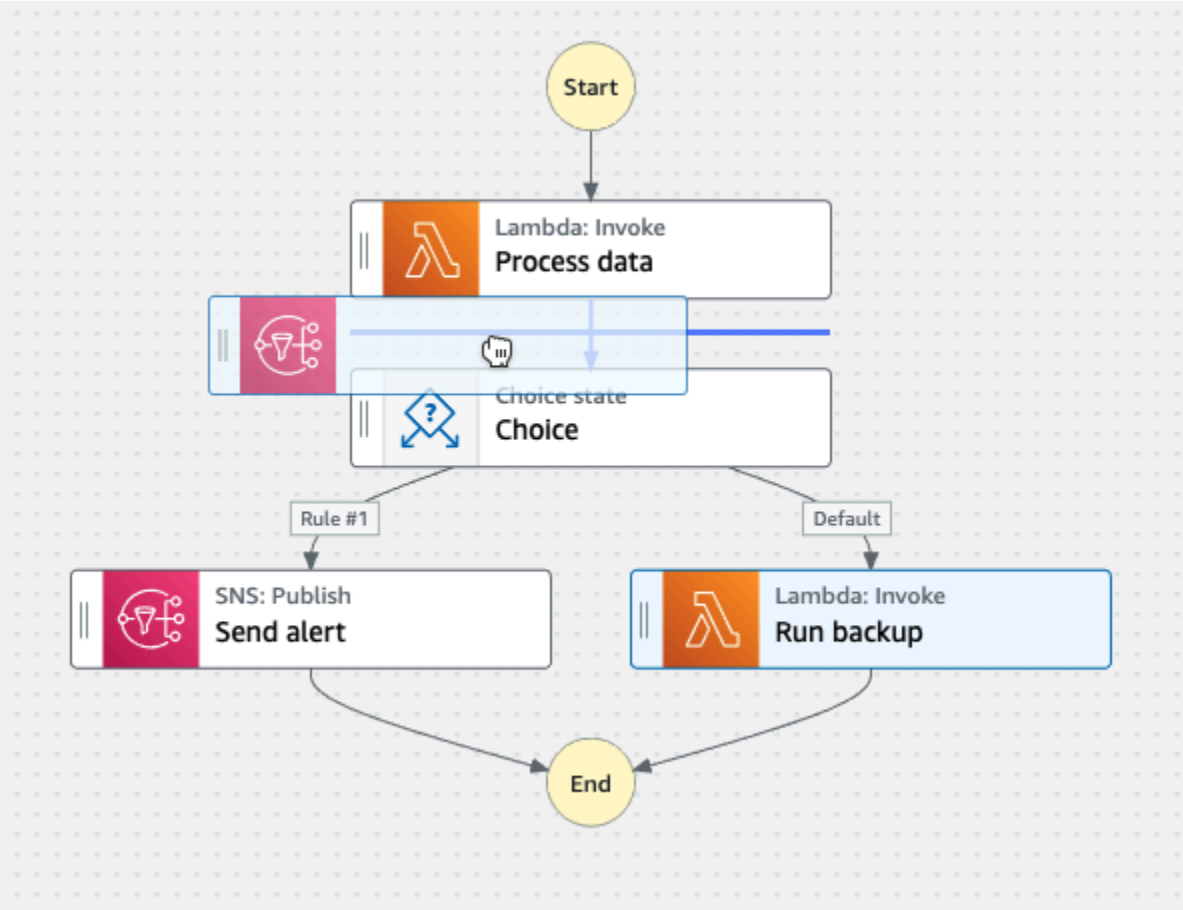
   これで、デザインモード でワークフローの設計を開始したり、コードモード へのワークフロー定義の書き込みを開始できます。
5. [設定] を選択して、設定モード のワークフローの設定を管理します。例えば、ワークフローの名前を入力し、そのタイプを選択します。

ワークフローの設計

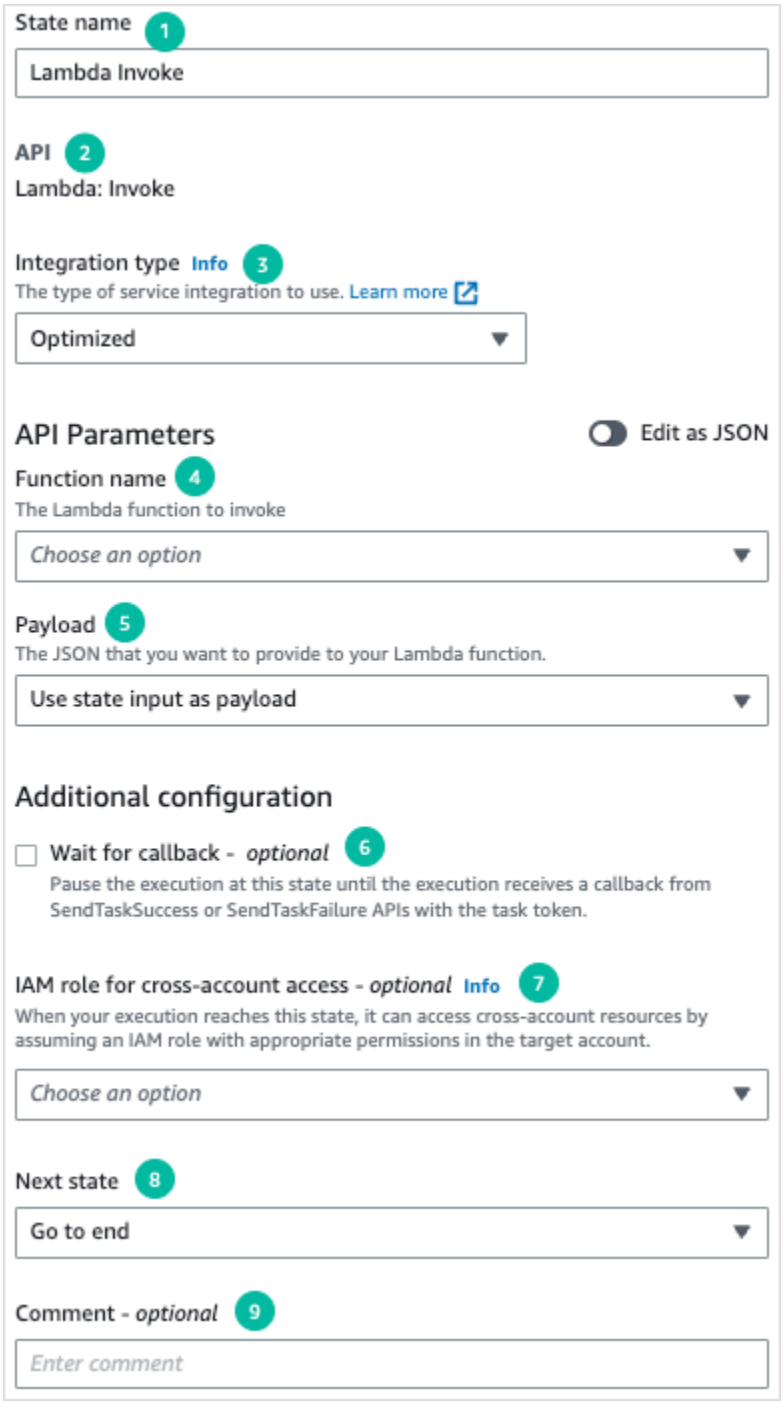
追加する状態の名前がわかっている場合は、[State browser] (状態ブラウザ) の上部にある検索ボックスを使用して、デザインモード の[アクション] タブと [フロー] タブでその状態を見つけます。



それ以外の場合は、ステートブラウザから状態を選択し、キャンバスにドラッグアンドドロップして、ワークフロー内の目的の場所に配置します。また、ワークフロー内の別の場所にドラッグして、ワークフローの状態を並べ替えることができます。状態をキャンバスにドラッグすると、ワークフロー内でドロップできる場所にラインが表示されます。状態がキャンバスにドロップされると、そのコードが自動生成され、ワークフロー定義内に追加されます。定義を確認するには、インスペクターパネルの[定義] トグルをオンにします。ワークフロー定義を編集するには、統合コードエディタが搭載されている コードモード を選択してください。



ステートをキャンバスにドロップすると、右側の Inspector パネルでそのステートを設定できます。このパネルには、キャンバスに配置した各ステートまたは API アクションの[設定]、[入力]、[出力]、および [エラー処理] タブがあります。ワークフローに含めるステートは、[設定] タブで設定します。例えば、Lambda 呼び出し API アクションの [設定] タブは次のオプションで構成されています。

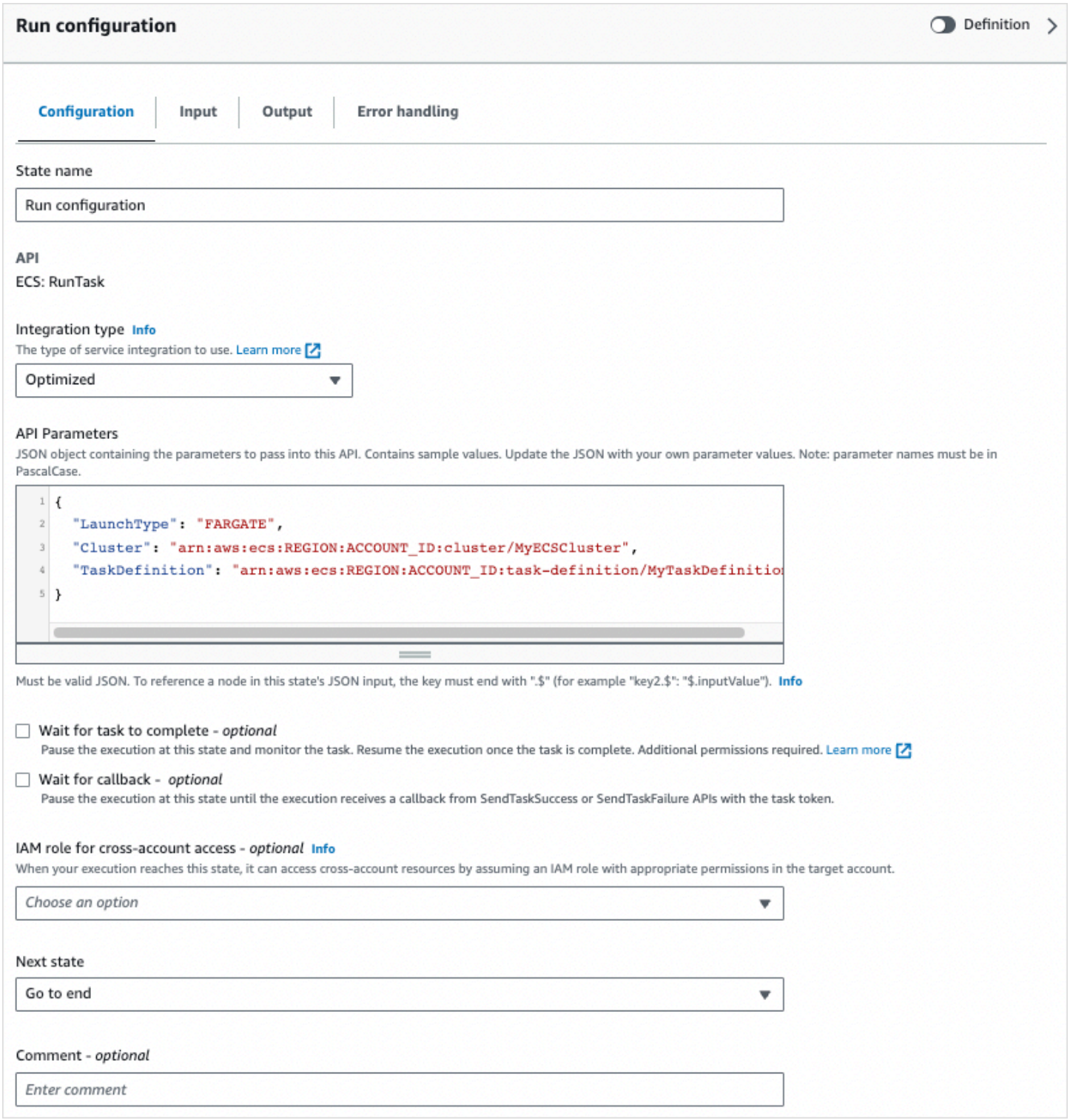


1. [State name] (状態名) は状態を示します。独自の名前を使用するか、デフォルトの生成名をそのまま受け入れます。
2. [API] には、ステートが使用する API アクションが表示されます。
3. [統合タイプ] ドロップダウンリストには、Step Functions で使用可能なサービス統合のタイプを選択するオプションが表示されます。選択した統合タイプは、 AWS のサービス ワークフローから特定の API アクションを呼び出すために使用されます。

4. [Function name] (関数名) は、以下にオプションを提供します。
   - [Enter a function name] (関数名を入力): 関数名またはその ARN を入力できます。
   - [Get function name at runtime from state input] (状態入力からランタイムに関数名を取得): このオプションを使用すると、指定したパスに基づいて、状態入力から関数名を動的に取得できます。
   - [Select function name] (関数名を選択): アカウントとリージョンで利用可能な機能から直接選択できます。
5. [ペイロード] では、次のオプションから選択します。
   - [状態入力をペイロードとして使用]: このオプションを使用して、Lambda 関数に提供されるペイロードとして状態の入力を渡すことができます。
   - [独自のペイロードを入力]: このオプションを使用して、ペイロードとして Lambda 関数に渡す JSON オブジェクトを構築できます。この JSON には、静的な値と状態入力から選択した値を両方とも含めることができます。
   - [ペイロードなし]: Lambda 関数にペイロードを渡さない場合は、このオプションを使用できます。
6. (オプション) 一部の状態には、[タスクが完了するまで待機] または [コールバックを待つ] を選択するオプションがあります。使用可能な場合、これらのオプションを使用して次の サービス統合パターン の一つを選択します。
   - [No option selected] (オプションの選択なし): Step Functions は レスポンスのリクエスト 統合パターンを使用します。Step Functions は HTTP 応答を待ってから、次の状態に進みます。Step Functions はジョブが完了するまで待機することはしません。使用可能なオプションがない場合、状態はこのパターンを使用します。
   - [Wait for task to complete] (タスクの完了を待つ): Step Functions は ジョブの実行 (.sync) 統合パターンを使用します。
   - [Wait for callback] (コールバックを待つ): Step Functions は タスクトークンのコールバックまで待機する 統合パターンを使用します。
7. (オプション) AWS アカウント ワークフロー内のさまざまな設定のリソースにアクセスするために、Step Functions はクロスアカウントアクセスを提供します。クロスアカウントアクセスのための IAM ロールには、以下のオプションがあります。
   - IAM ロールの ARN を指定: 適切なリソースアクセス許可を含む IAM ロールを指定します。これらのリソースは、クロスアカウント呼び出しを行うターゲットアカウントで使用できます。AWS アカウント
   - 状態入力からランタイムに IAM ロール ARN を取得する: IAM ロールを含むステートの JSON 入力内の既存のキーと値のペアへの参照パスを指定します。

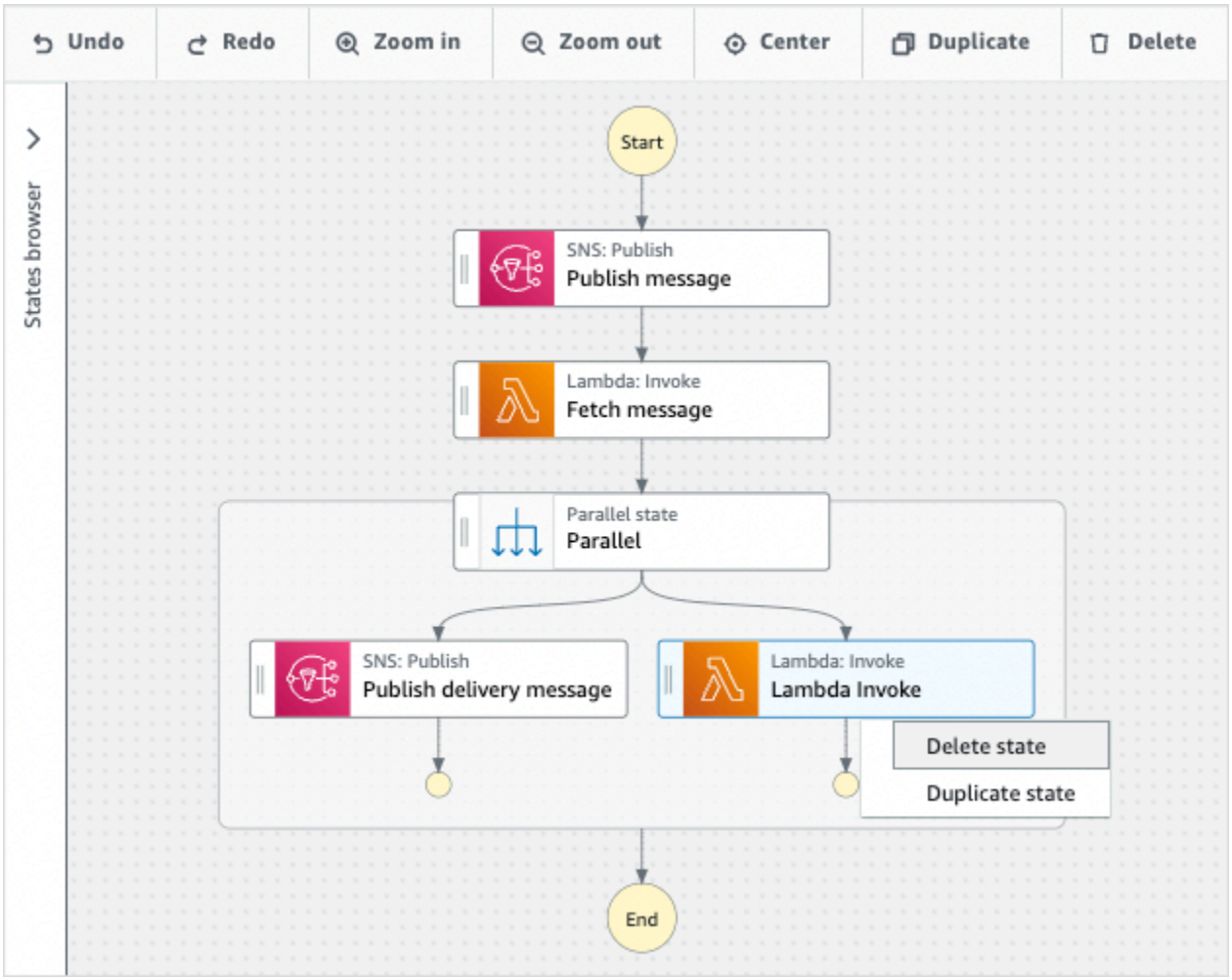
8. [Next state] (次の状態) 次に移行したい状態を選択できます。
9. (オプション) [Comment] (コメント) フィールドを使用して、独自のコメントを追加できます。
   ワークフローには影響しませんが、ワークフローに注釈を付けるために使用できます。

一部の状態には、さらに包括的な設定オプションがあります。例えば、Amazon ECS RunTask 状態の設定には、プレースホルダー値が入力された `API Parameters` フィールドを含みます。



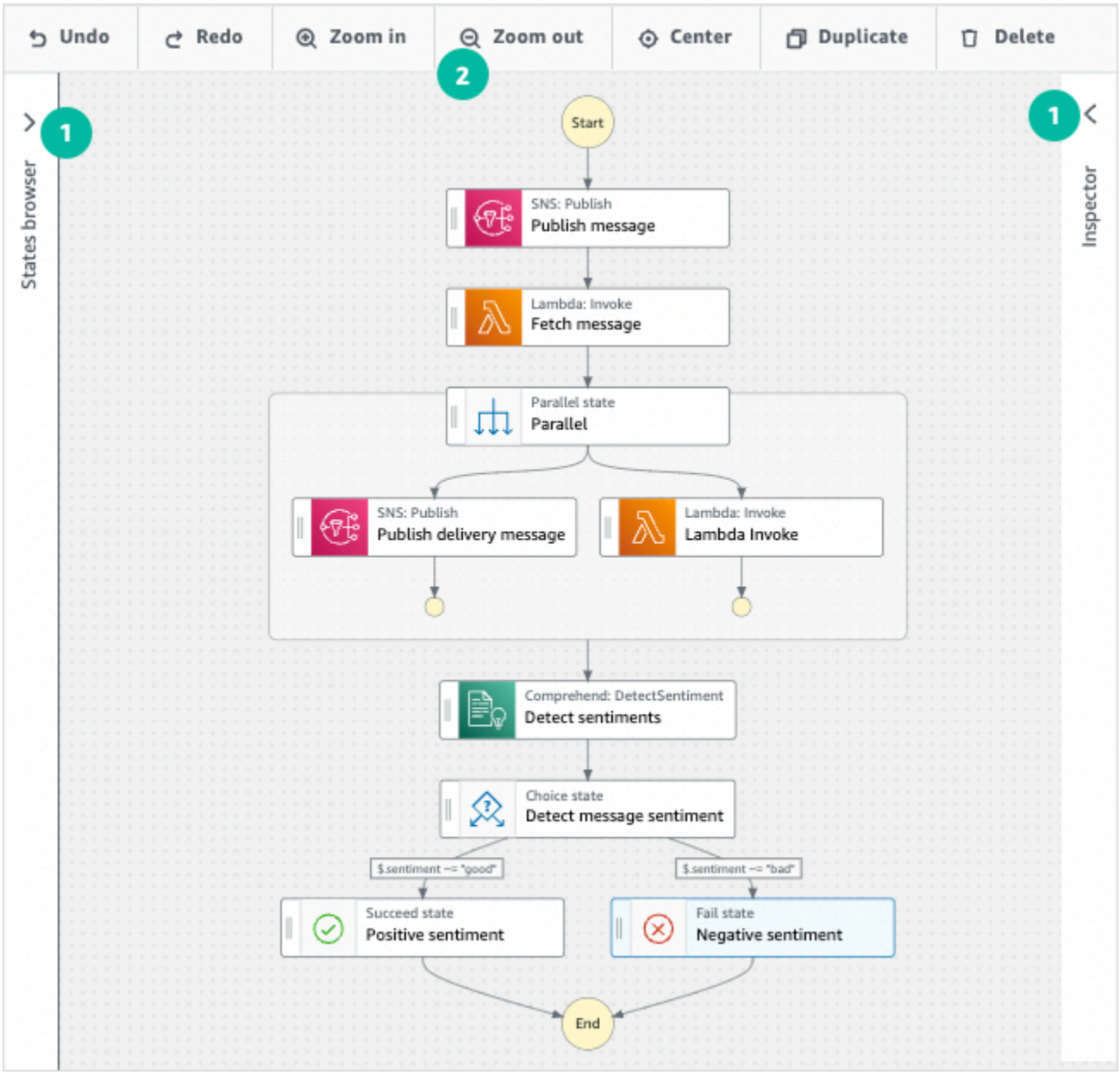
これらの状態では、プレースホルダー値をニーズに適した構成と置き換えることができます。

状態を削除するには、バックスペースを使用して右クリックし、[状態の削除] または [デザインツールバー] の [削除] を選択します。



ワークフローが大きくなると、キャンバスに収まらない場合があります。次のようにできます。

1. サイドパネルのコントロールを使用して、パネルのサイズを変更するか、パネルを閉じます。
2. キャンバス の上部にあるデザインツールバーを使用して、ワークフローのグラフをズームインまたはズームアウトします。



ワークフローを実行

Workflow Studio を使用してワークフローを作成または編集した後、ワークフローを実行すると、Step Functions コンソールでそれが表示できます。

Workflow Studio でワークフローを実行するには

1. [デザイン]、[コード]、または[設定] モードで、[実行]を選択します。

   [実行を開始] ダイアログが新しいタブに開きます。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。
3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

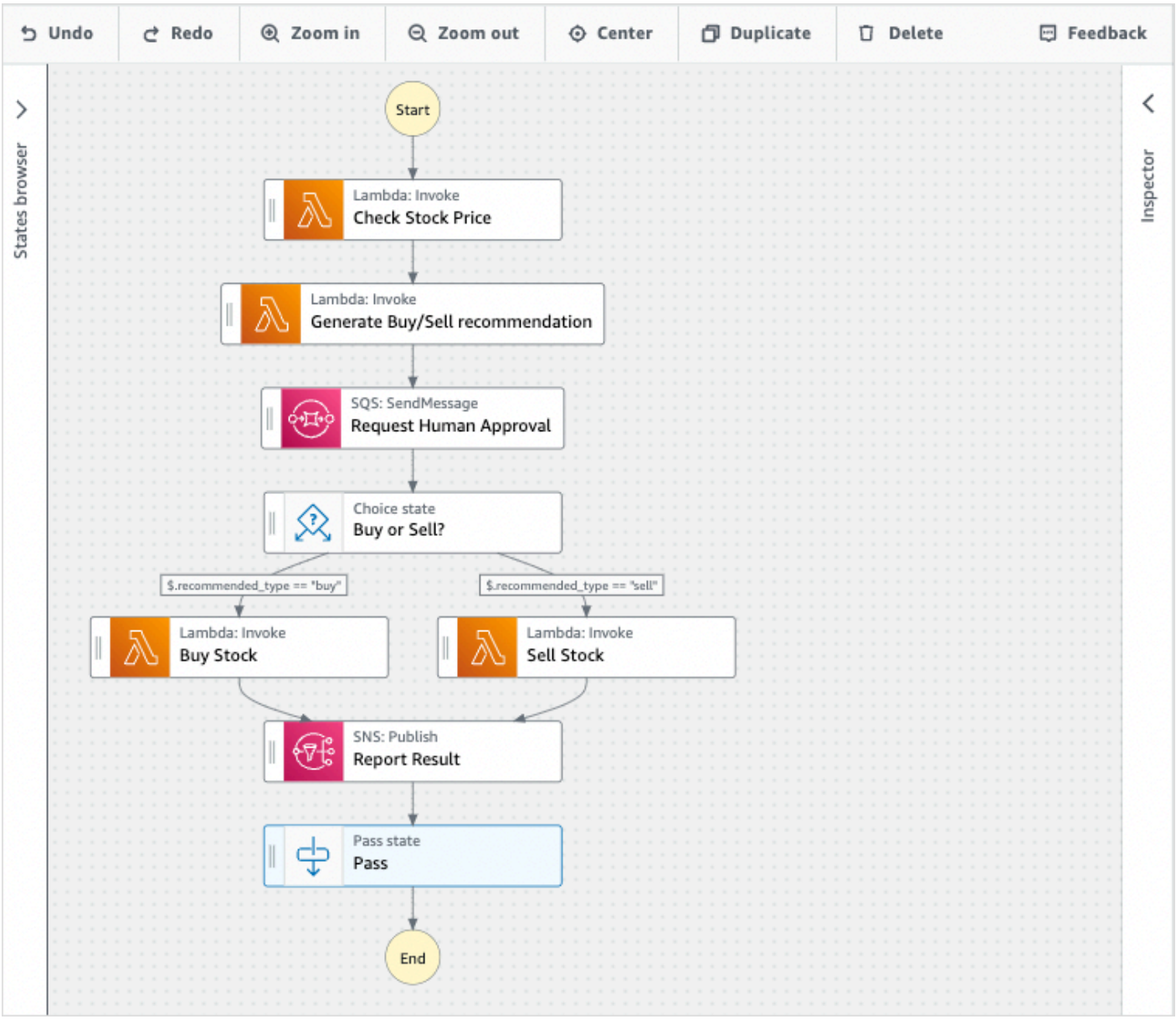
   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ワークフローの編集

Workflow Studio の デザインモード では、既存のワークフローを視覚的に編集できます。Workflow Studio の コードモード でワークフロー定義を編集することもできます。

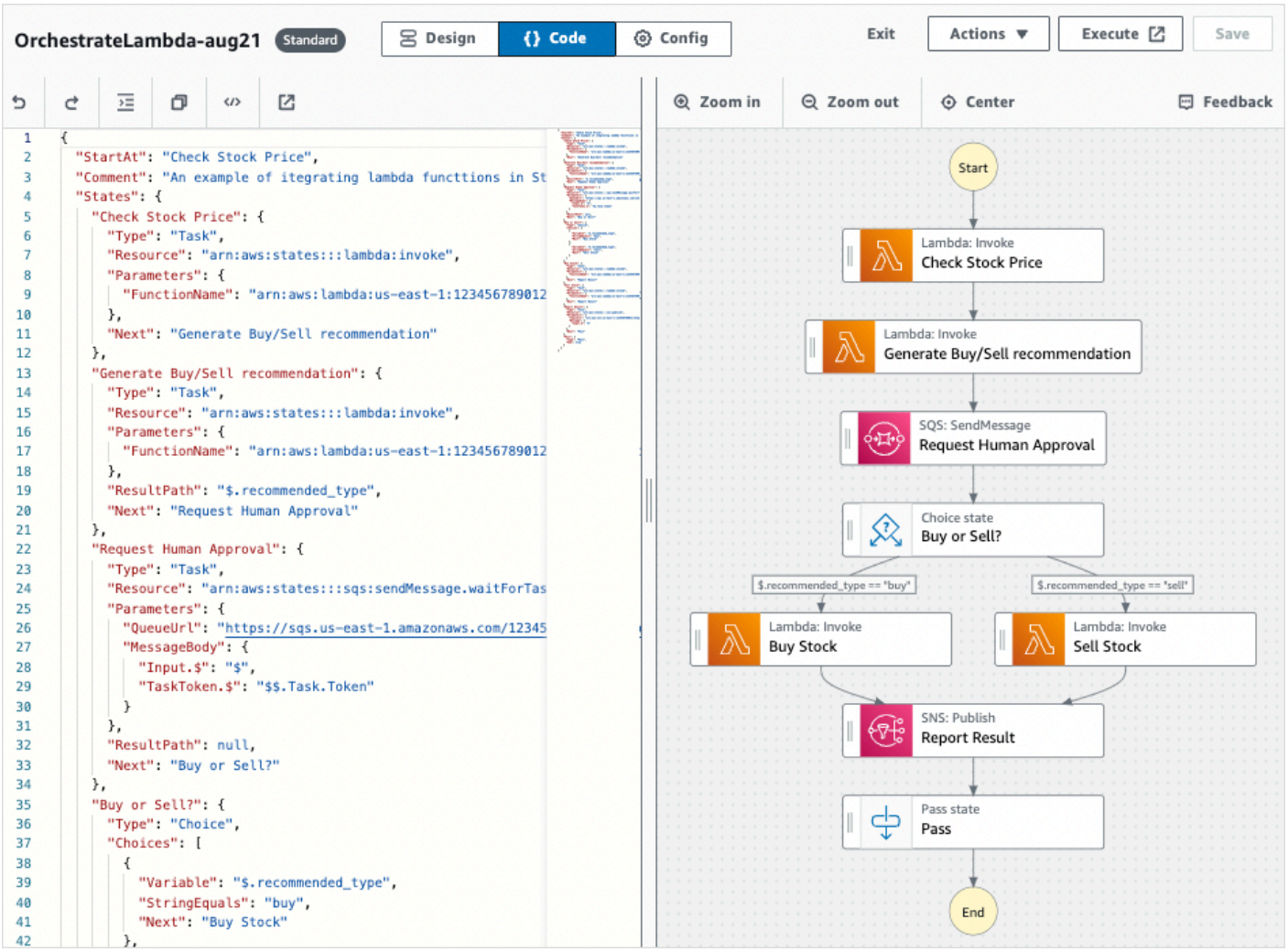
既存のワークフローを編集するには:

1. Step Functions コンソールを開きます。
2. [ステートマシン] ページで、編集するワークフローを選択します。
3. [ステートマシンの詳細] ページで、[編集] を選択します。
4. ワークフローは Workflow Studio の[デザイン] モードで開きます。必要に応じてワークフローを編集します。



> **Note**
>
> ワークフローにエラーが見つかった場合は、[デザイン] モードで修正する必要があります。ワークフローにエラーがある場合は、[コード] モードまたは [設定] モードに切り替えることはできません。

5. (オプション) [コード] ボタンを選択して、Workflow Studio でワークフロー定義を表示または編集します。



6. 完了したら、[保存] を選択して、更新されたワークフローを保存します。
7. (オプション) 更新したワークフローを実行するには、[実行] を選択します。[実行を開始] ダイアログが新しいタブに開きます。

ワークフローのエクスポート

ワークフローの Amazon ステートメント言語 (ASL) 定義とワークフローのグラフをエクスポートできます。

1. Step Functions コンソールでワークフローを選択します。
2. [ステートマシンの詳細] ページで、[編集] を選択します。
3. (オプション) ワークフローは Workflow Studio の [デザイン] モードで開きます。[デザイン] モードでワークフローを編集するか、[コード] モードに切り替えます。
4. [アクション] ドロップダウンボタンを選択し、次のいずれかまたは両方を実行します。

- ワークフローのグラフを SVG または PNG ファイルにエクスポートするには、[グラフをエクスポート] で、使用する形式を選択します。
- ワークフロー定義を JSON または YAML ファイルとしてエクスポートするには、[定義をエクスポート] で目的の形式を選択します。

ワークフロープロトタイプの作成

Workflow Studio を使用して、プレースホルダーリソースを含む新しいワークフローのプロトタイプを作成します。Application Composer の Workflow Studio を使用してワークフローを構築することもできます。プロトタイプを作成するには:

1. Step Functions コンソールにサインインします。
2. Create State Machine (ステートマシンの作成) を選択します。
3. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
4. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
5. Workflow Studio の [デザイン] モードが開きます。Workflow Studio でワークフローを設計します。プレースホルダーリソースを含めるには、次の手順に従います。
   a. プレースホルダーリソースを含める状態を選択し、[設定] を選択します。
      - Lambda 呼び出し状態の場合、[関数名] を選択し、その後、[関数名を入力] を選択します。関数のカスタム名前を入力することもできます。
      - Amazon SQS Send Message 状態については、[キュー URL] を選択し、その後、[キュー URL を入力] を選択します。プレースホルダーキュー URL を入力します。
      - Amazon SNS 発行ステートの場合は、[トピック] からトピック ARN を選択します。
      - [Actions] (アクション) の下にリストされている他の状況すべてについては、デフォルト設定を使用できます。

      > **Note**
      >
      > ワークフローにエラーが見つかった場合は、[デザイン] モードで修正する必要があります。ワークフローにエラーがある場合は、[コード] モードまたは [設定] モードに切り替えることはできません。

   b. (オプション) ワークフローの自動生成 ASL 定義を表示するには、[定義] を選択します。
   c. (オプション) Workflow Studio でワークフロー定義を更新するには、[コード] ボタンを選択します。

> **Note**
>
> ワークフロー定義にエラーがある場合は、[コード] モードで修正する必要があります。ワークフロー定義にエラーがある場合は、[デザイン] モードまたは [設定] モードに切り替えることはできません。

6. (オプション) ステートマシン名を編集するには、デフォルトのステートマシン名の横にある編集アイコンを選択し、ステートマシン名ボックスに名前を指定します。MyStateMachine

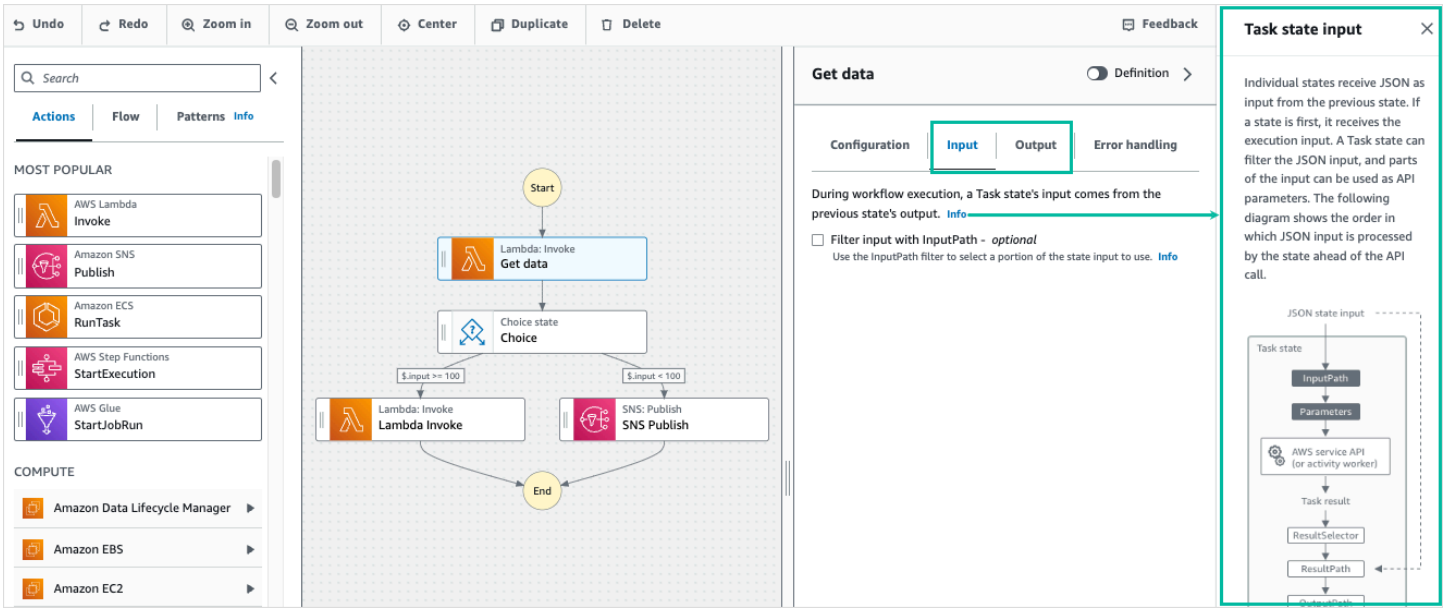
   設定モード に切り替えてデフォルトのステートマシン名を編集することもできます。
7. ステートマシンのタイプや実行ロールなどのワークフロー設定を指定します。
8. [作成] を選択します。

プロトタイプに使用できるプレースホルダーリソースを使って、新しいワークフローが作成されるようになりました。ワークフロー定義とワークフローのグラフをエクスポートできます。

- ワークフロー定義を JSON または YAML ファイルとしてエクスポートするには、[デザイン] モードまたは [コード] モードで [アクション] ドロップダウンボタンを選択します。次に、[定義をエクスポート] で、エクスポートする形式を選択します。このエクスポートされた定義は、AWS Toolkit for Visual Studio Code を使用したローカル開発の開始点として使用できます。
- ワークフローグラフを SVG または PNG ファイルにエクスポートするには、[デザイン] モードまたは [コード] モードで、[アクション] ドロップダウンボタンを選択します。次に、[定義をエクスポート] で、必要な形式を選択します。

状態の入力と出力を構成する

各状態は、受け取った入力に基づいて決定を下すか、アクションを実行します。ほとんどの場合、出力を他の状態に渡します。Workflow Studio では、Inspector パネルの [入力] タブおよび [出力] タブでステートが入力と出力データをフィルタリングし、操作する方法を設定できます。[Info] リンクを使用して、入力と出力を設定する時にコンテキストヘルプにアクセスします。



Step Functions の入出力処理方法の詳細については、Step Functions の入出力処理 を参照してください。

状態へ入力を構成する

各状態は、前の状態からの入力を JSON として受け取ります。入力をフィルターする場合は、Inspector パネルの [入力] タブで `InputPath` フィルターを使用できます。`InputPath` は文字列であり、`$` で始まり、特定の JSON ノードを識別します。これらは参照パスと呼ばれ、 JsonPath 構文に従います。

入力をフィルターするには、次の手順を実行します。

- [入力のフィルター対象] を選択します InputPath。
- JsonPath`InputPath`フィルターに有効な値を入力します。例えば **`$.data`** です。

ご自分の InputPath フィルターがご自分のワークフローに追加されます。

Example 例 1: InputPath ワークフロースタジオでフィルターを使用する

ステートへの入力に次の JSON データが含まれているとします。

```
{
  "comment": "Example for InputPath",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

InputPathフィルターを適用するには、「入力をフィルターする」を選択し InputPath、適切な参照パスを入力します。**$.dataset2.val1** に入ると、次の JSON が状態への入力として渡されます。

```
{"a"}
```

リファレンスパスでは、値の選択もできます。リファレンスが { "a": [1, 2, 3, 4] } で、リファレンスパス$.a[0:2] を InputPath フィルターとして適用した場合、結果は次のようになります。

```
[ 1, 2 ]
```

並行、マッピング、パス フローには、[入力] タブで Parameters と呼ばれる追加の入力フィルタリングオプションがあります。このフィルターはフィルターの後に有効になり、1 つ以上のキーと値のペアで構成されるカスタム JSON オブジェクトを構築するために使用できます。 InputPath 各ペアの値は、静的な値のいずれかとなり、入力から選択するか、パスを使って コンテキストオブジェクト から選択することができます。

> **Note**
>
> パラメータでリファレンスパスを使用して入力内の JSON ノードを参照するように指定するには、パラメータ名の末尾を `.$` で終了します。

Example 例 2: パラレルステート用のカスタム JSON 入力の作成

次の JSON データがパラレルステートへの入力であるとします。

```
{
  "comment": "Example for Parameters",
  "product": {
    "details": {
      "color": "blue",
      "size": "small",
      "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

この入力の一部を選択し、静的な値を使って追加のキーバリューペアを渡すには、[パラメータ] フィールドで [並列] 状態の [入力] タブ で次のように指定できます。

```
{
 "comment": "Selecting what I care about.",
 "MyDetails": {
    "size.$": "$.product.details.size",
    "exists.$": "$.product.availability",
    "StaticValue": "foo"
    }
 }
```

次の JSON データが結果になります。

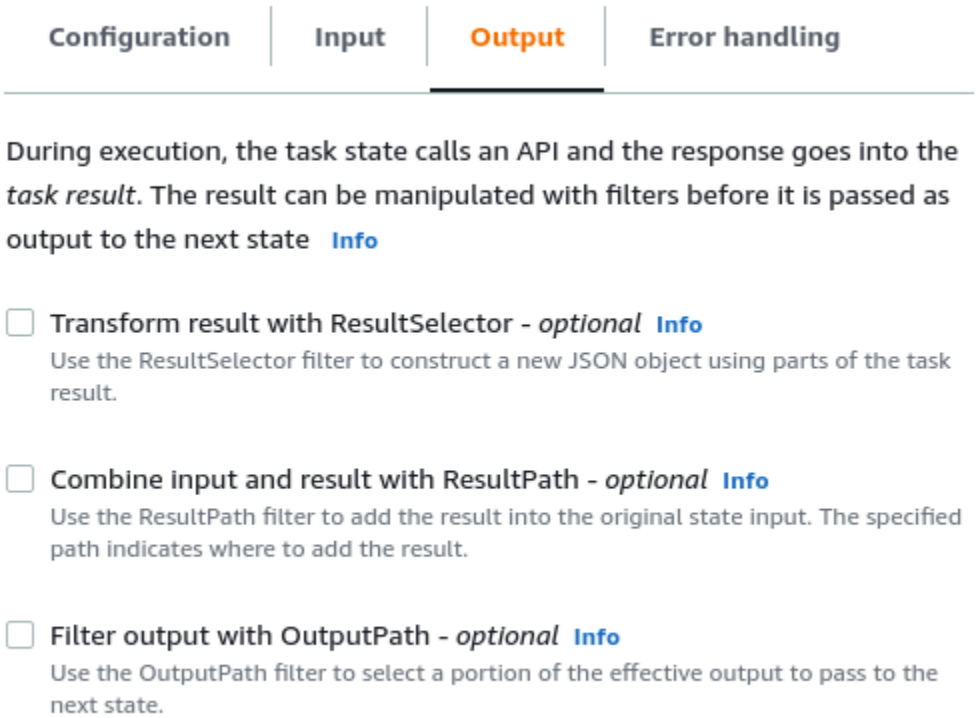
```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
```

```
    "size": "small",
    "exists": "in stock",
    "StaticValue": "foo"
  }
}
```

状態の出力を構成

状態ごとに、次の状態に渡される前にフィルタリングできる JSON 出力を生成します。複数のフィルタが使用でき、それぞれ異なる方法で出力に影響します。各状態で使用可能な出力フィルターは、[Inspector] 内の [出力] タブでリストされます。タスク 状態では、選択した出力フィルターは次の順序で処理されます。

1. ResultSelector: 状態の結果を操作するためにこのフィルターを使用します。結果の一部を使用して新しい JSON オブジェクトを作成できます。
2. ResultPath: このフィルターを使用して、出力に渡す状態入力とタスク結果の組み合わせを選択します。
3. OutputPath: このフィルターを使用して、JSON 出力をフィルターして、結果からどの情報を次の状態に渡すのか選択します。



を使う ResultSelector

`ResultSelector` は、次の状態に対するオプションの出力フィルターです。

- タスク 状態とは、[State browser] (状態ブラウザ) の [アクション] タブにあげられているすべての状態です。
- 状態ブラウザの [フロー] タブにある マッピング 状態。
- 状態ブラウザの [フロー] タブにある 並行 状態。

`ResultSelector` を使用して、1 つ以上のキーバリューペアで構成されるカスタム JSON オブジェクトを作成できます。各ペアの値は、静的な値にすることも、状態の結果からパスで選択することもできます。

> **Note**
>
> パラメータでパスを使用して、結果内の JSON ノードをリファレンスにするように指定するには、パラメータ名の末尾を `.$` で終了します。

Example ResultSelector フィルターの使用例

この例では、`ResultSelector`を使用して Amazon EMR CreateCluster API 呼び出しからのレスポンスを Amazon EMR の状態に合わせて操作します。`CreateCluster`以下は、Amazon EMR `CreateCluster` API コールからの結果です。

```
{
  "resourceType": "elasticmapreduce",
  "resource": "createCluster.sync",
  "output": {
    "SdkHttpMetadata": {
      "HttpHeaders": {
        "Content-Length": "1112",
        "Content-Type": "application/x-amz-JSON-1.1",
        "Date": "Mon, 25 Nov 2019 19:41:29 GMT",
        "x-amzn-RequestId": "1234-5678-9012"
      },
      "HttpStatusCode": 200
    },
    "SdkResponseMetadata": {
```

```
      "RequestId": "1234-5678-9012"
    },
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

この情報の一部を選択し、追加のキーと値のペアを静的な値とともに渡すには、州の [出力] ResultSelectorタブの下のフィールドに以下を指定します。

```
{
 "result": "found",
 "ClusterId.$": "$.output.ClusterId",
 "ResourceType.$": "$.resourceType"
 }
```

ResultSelector を使用すると、次の結果になります。

```
{
 "result": "found",
 "ClusterId": "AKIAIOSFODNN7EXAMPLE",
 "ResourceType": "elasticmapreduce"
}
```

を使用してください。 ResultPath

状態の出力は、入力のコピー、生成される結果、またはその入力と結果の組み合わせです。ResultPath を使用して、上記のうち、状態出力に渡す組み合わせを制御します。その他の ResultPath ユースケースについては、ResultPath を参照してください。

ResultPath は、次の状態に対するオプションの出力フィルターです。

- タスク 状態は、状態ブラウザの [アクション] パネルにあげられているすべての状態です。
- 状態ブラウザの [フロー] タブにある マッピング 状態。
- 状態ブラウザの [フロー] タブにある 並行 状態。
- 状態ブラウザの [フロー] タブにある パス 状態。

ResultPath は、結果を元の状態の入力に追加するために使用できます。指定されたパスは、結果を追加する場所を示します。

Example ResultPath フィルターの使用例

次のものがタスクステートへの入力であるとします。

```
{
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

タスクステートの結果は次のとおりです。

```
Hello, AWS Step Functions
```

ResultPath を適用し、$.taskresult のように、結果を追加する場所を示すリファレンスパスを入力して、状態の入力にこの結果を追加します。

この ResultPath を使って、状態の出力として渡される JSON を次に示します。

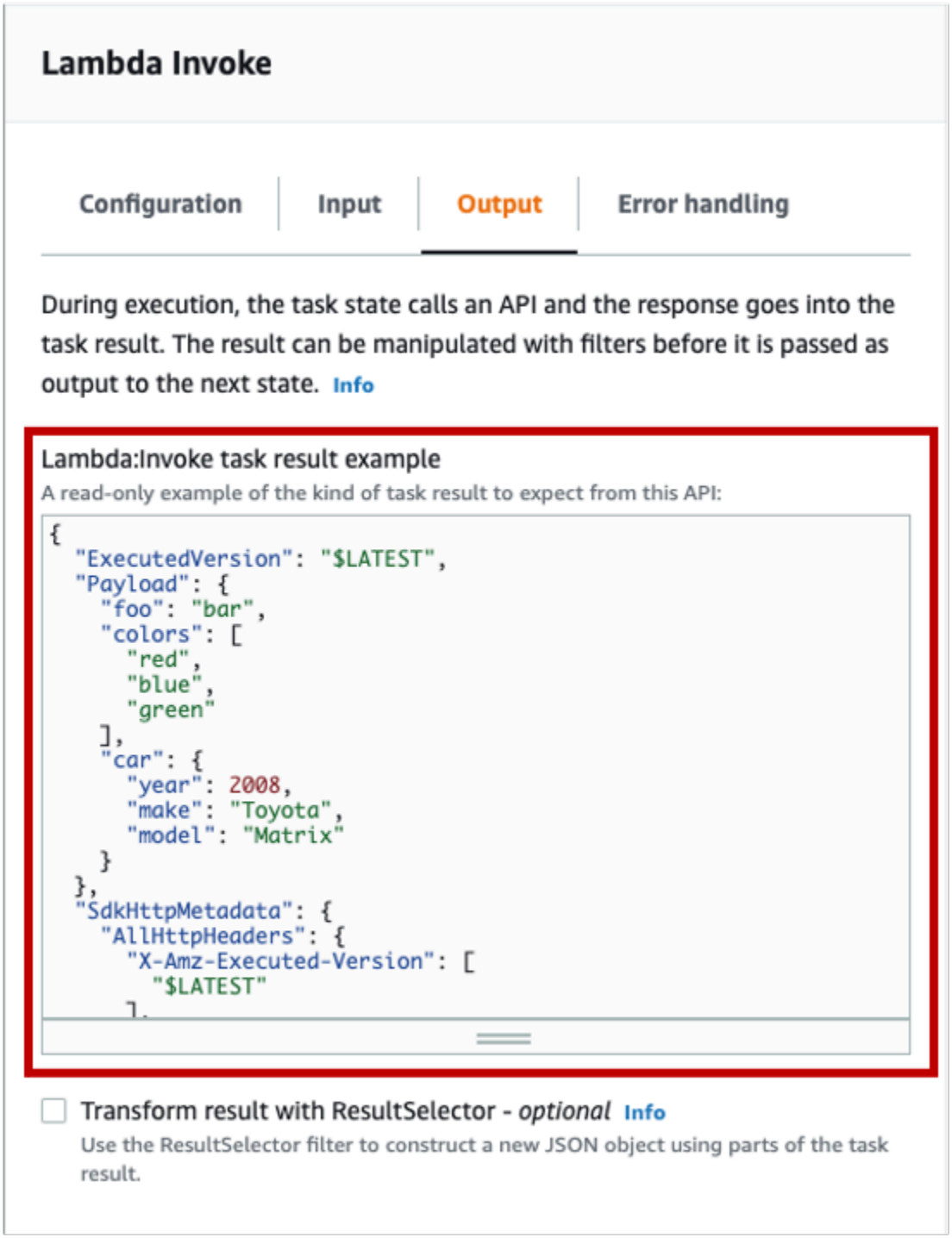
```
{
  "details": "Default example",
  "who": "AWS Step Functions",
  "taskresult": "Hello, AWS Step Functions!"
}
```

使用する OutputPath

OutputPath フィルターを使うと、不要な情報をフィルターして、必要な一部の JSON のみを渡すことができるようになります。OutputPath は $ で始まる文字列で、JSON テキスト内のノードを識別します。

Example OutputPath フィルターの使用例

Lambda Invoke API コールは、Lambda 関数の結果であるペイロードに加えてメタデータを返します。この API コールからのレスポンスの例は、状態の [出力] タブの下に表示されます。



OutputPath を使用して、追加のメタデータをフィルターで除外します。デフォルトでは、ワークフロースタジオで作成された Lambda Invoke OutputPathステートのフィルターの値はです。$.Payloadこのデフォルト値では、追加のメタデータが削除され、Lambda 関数を直接実行するのに等しい出力が返されます。

Lambda 呼び出しタスクの結果の例と [出力] フィルターの `$.Payload` の値は、次の JSON データを出力として渡します。

```
{
 "foo": "bar",
 "colors": [
      "red",
      "blue",
      "green"
 ],
 "car": {
      "year": 2008,
      "make": "Toyota",
      "model": "Matrix"
 }
}
```

> **Note**
>
> `OutputPath` フィルターは、最後に有効になる出力フィルターなので、`ResultSelector` または `ResultPath` のような追加出力フィルターを使用するのであれば、`OutputPath` フィルターの `$.Payload` のデフォルト値を適宜変更する必要があります。

Workflow Studio での実行ロール

すべてのStep Functionsステートマシンには、 AWS のサービス および リソースに対してアクションを実行したり、サードパーティー APIsを呼び出したりするためのアクセス許可をステートマシンに付与する AWS Identity and Access Management （ IAM) ロールが必要です。このロールは、実行ロールと呼ばれます。このロールには、ステートマシンが AWS Lambda 関数を呼び出したり、AWS Batch ジョブを実行したり、Stripe API を呼び出したりすることを許可する IAM ポリシーなど、アクションごとのポリシーが含まれている必要があります。Step Functions では、次の場合に実行ロールを指定する必要があります。

- ステートマシンは、コンソール、 AWS SDKs、または CreateStateMachine API AWS CLI を使用して作成します。
- 状態は、コンソール、 AWS SDKsAWS CLIを使用してテストします。 TestState

Workflow Studio には、ワークフローの実行ロールを簡単に管理できる機能があります。

トピック

- 自動生成されたロールについて
- ロールを自動生成する
- ロール生成に関する問題を解決する
- Workflow Studio で HTTP タスクをテストするためのロール
- Workflow Studio で最適化されたサービス統合をテストするためのロール
- Workflow Studio で AWS SDK サービス統合をテストするためのロール
- Workflow Studio でフローステートをテストするためのロール

自動生成されたロールについて

Step Functions コンソールでステートマシンを作成すると、Workflow Studio は必要な IAM ポリシーを含む実行ロールを自動的に作成できます。Workflow Studio はステートマシンの定義を分析し、ワークフローの実行に必要な最小特権でポリシーを生成します。

Workflow Studio では以下の IAM ポリシーを生成できます。

- サードパーティー API を呼び出す HTTP タスク。
- 、、DynamoDB GetItemまたは の呼び出しなど、最適化された統合 AWS のサービス を使用して他の を呼び出すタスク状態AWS Glue StartJobRun。 Lambda
- ネストされたワークフローを実行するタスクステート。
- 子ワークフロー実行の開始、Amazon S3 バケットのリスト、S3 オブジェクトの読み取りと記述などのポリシーを含む、分散マップ状態。
- X-Ray トレース。Workflow Studio で自動生成されるすべてのロールには、X-Ray にトレースを送信するアクセス許可をステートマシンに付与するポリシーが含まれています。
- ステートマシンでロギングが有効になっている場合の CloudWatch Logs を使用したログ記録。

Workflow Studio は、 AWS SDK 統合 AWS のサービス を使用して他の を呼び出すタスク状態の IAMポリシーを生成できません。

ロールを自動生成する

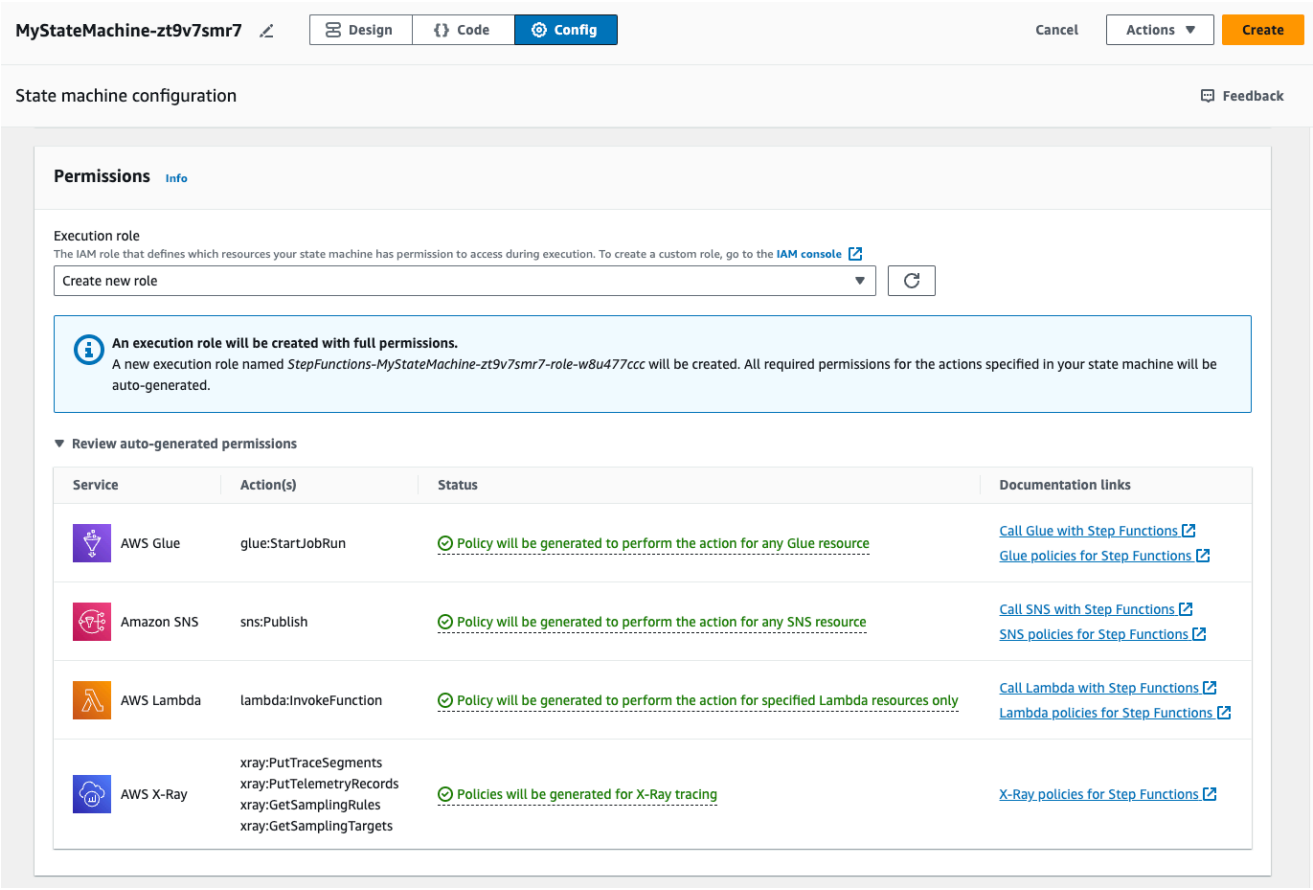
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

既存のステートマシンを更新することもできます。ステートマシンを更新する場合は、ステップ 4 を参照してください。

2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. [設定] タブを選択します。
5. [アクセス許可] セクションまでスクロールして、次の操作を行います。

   a. [実行ロール] では、デフォルトの [新しいロールを作成] を選択したままにしてください。

   Workflow Studio は、ステートマシン定義内の有効なステートごとに、必要なすべての IAM ポリシーを自動的に生成します。[完全なアクセス許可で実行ロールが作成されます] というメッセージを含むバナーが表示されます。



> **Tip**
>
> Workflow Studio がステートマシンに対して自動的に生成するアクセス許可を確認するには、[自動生成されたアクセス許可をレビューする] を選択します。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

Workflow Studio で必要な IAM ポリシーをすべて生成できない場合、[特定のアクションのアクセス許可は自動生成できません] というメッセージを含むバナーが表示されます。IAM ロールは部分的なアクセス許可のみで作成されます。不足しているアクセス許可を追加する方法については、「ロール生成に関する問題を解決する」を参照してください。

b. ステートマシンを作成する場合は [作成] を選択します。問題がなければ、[保存] を選択します。

c. 表示されるダイアログボックスで、[確認] を選択します。

Workflow Studio はステートマシンを保存し、新しい実行ロールを作成します。

ロール生成に関する問題を解決する

以下の場合、Workflow Studio は必要なすべてのアクセス許可を含む実行ロールを自動的に生成できません。

- ステートマシンにエラーがある。Workflow Studio の検証エラーをすべて解決してください。また、保存中に発生したサーバー側のエラーには必ず対処してください。
- ステートマシンには AWS SDK 統合を使用するタスクが含まれています。この場合、Workflow Studio は IAM ポリシーを自動生成できません。Workflow Studio には、[特定のアクションのアクセス許可は自動生成できません] というメッセージを含むバナーが表示されます。IAM ロールは部分的なアクセス許可のみで作成されます。[自動生成アクセス許可の確認] テーブルで、[ステータス] の内容を選択すると、実行ロールにないポリシーの詳細が表示されます。Workflow Studio は

引き続き実行ロールを生成できますが、このロールにはすべてのアクションの IAM ポリシーが含まれるわけではありません。独自のポリシーを作成し、生成後にロールに追加するには、[ドキュメントリンク] の下のリンクを参照してください。これらのリンクは、ステートマシンを保存した後でも使用できます。

Workflow Studio で HTTP タスクをテストするためのロール

HTTP タスクステートをテストするには実行ロールが必要です。十分なアクセス許可を持つロールがない場合は、以下のオプションのいずれかを使用してロールを作成してください。

- Workflow Studio でロールを自動生成する (推奨) — これは安全なオプションです。[ステートをテスト] ダイアログボックスを閉じて、ロールを自動生成する の指示に従います。そのためには、まずステートマシンを作成または更新してから、Workflow Studio に戻ってステートをテストする必要があります。
- 管理者アクセス権を持つロールを使用する – のすべてのサービスとリソースへのフルアクセス権を持つロールを作成するアクセス許可がある場合は AWS、そのロールを使用してワークフロー内の任意のタイプの状態をテストできます。これを行うには、Step Functionsサービスロールを作成し、IAMコンソール https://console.aws.amazon.com/iam/ でAdministratorAccess ポリシーを追加します。

Workflow Studio で最適化されたサービス統合をテストするためのロール

最適化されたサービス統合を呼び出すタスクステートには実行ロールが必要です。十分なアクセス許可を持つロールがない場合は、以下のオプションのいずれかを使用してロールを作成してください。

- Workflow Studio のドキュメントリンクを使用して独自の IAM ポリシーを記述する (推奨) — これは安全なオプションです。[ステートをテスト] ダイアログボックスを閉じて、ロールを自動生成する の指示に従います。そのためには、まずステートマシンを作成または更新してから、Workflow Studio に戻ってステートをテストする必要があります。
- 管理者アクセス権を持つロールを使用する – のすべてのサービスとリソースへのフルアクセス権を持つロールを作成するアクセス許可がある場合は AWS、そのロールを使用してワークフロー内の任意のタイプの状態をテストできます。これを行うには、Step Functionsサービスロールを作成し、IAMコンソール https://console.aws.amazon.com/iam/ でAdministratorAccess ポリシーを追加します。

Workflow Studio で AWS SDK サービス統合をテストするためのロール

AWS SDK 統合を呼び出すタスクステートには実行ロールが必要です。十分なアクセス許可を持つロールがない場合は、以下のオプションのいずれかを使用してロールを作成してください。

- Workflow Studio のドキュメントリンクを使用して独自の IAM ポリシーを記述する (推奨) — これは安全なオプションです。[ステートをテスト] ダイアログボックスを閉じて、ロールを自動生成する の指示に従います。そのためには、まずステートマシンを作成または更新してから、Workflow Studio に戻ってステートをテストする必要があります。以下の操作を実行します。
  1. [ステートをテスト] ダイアログボックスを閉じます。
  2. [設定] タブを選択し、設定モードを表示します。
  3. [アクセス許可] セクションまで下にスクロールします。
  4. Workflow Studio には、[特定のアクションのアクセス許可は自動生成できません] というメッセージを含むバナーが表示されます。IAM ロールは部分的なアクセス許可のみで作成されます。[自動生成されたアクセス許可を確認] を選択します。
  5. [自動生成されたアクセス許可を確認] テーブルには、テストするタスクステートに対応するアクションを示す行が表示されます。カスタムロールに独自の IAM ポリシーを記述するには、[ドキュメントリンク] の下のリンクを参照してください。
- 管理者アクセス権を持つロールを使用する – のすべてのサービスとリソースへのフルアクセス権を持つロールを作成するアクセス許可がある場合は AWS、そのロールを使用してワークフロー内の任意のタイプの状態をテストできます。これを行うには、Step Functionsサービスロールを作成し、IAMコンソール https://console.aws.amazon.com/iam/ でAdministratorAccess ポリシーを追加します。

Workflow Studio でフローステートをテストするためのロール

Workflow Studio でフローステートをテストするには実行ロールが必要です。フローステートは、選択、並行、マッピング、パス、待機、成功、または 失敗 などの実行フローを指示する状態です。TestState API はマップ状態または並列状態では機能しません。次のいずれかのオプションを使用してフロー状態をテストするためのロールを作成します。

- で任意のロールを使用する AWS アカウント （ 推奨) – フロー状態は AWS アクションやリソースを呼び出す必要がないため、特定のIAMポリシーは必要ありません。したがって、 で任意のIAMロールを使用できます AWS アカウント。
  1. [ステートをテスト] ダイアログボックスで、[実行ロール] ドロップダウンリストから任意のロールを選択します。

2. ドロップダウンリストにロールが表示されない場合は、以下の作業を行います。

   a. IAM コンソール https://console.aws.amazon.com/iam/ で [ロール] を選択します。

   b. リストからロールを選択し、ロールの詳細ページからその ARN をコピーします。この ARN は [ステートをテスト] ダイアログボックスに入力する必要があります。

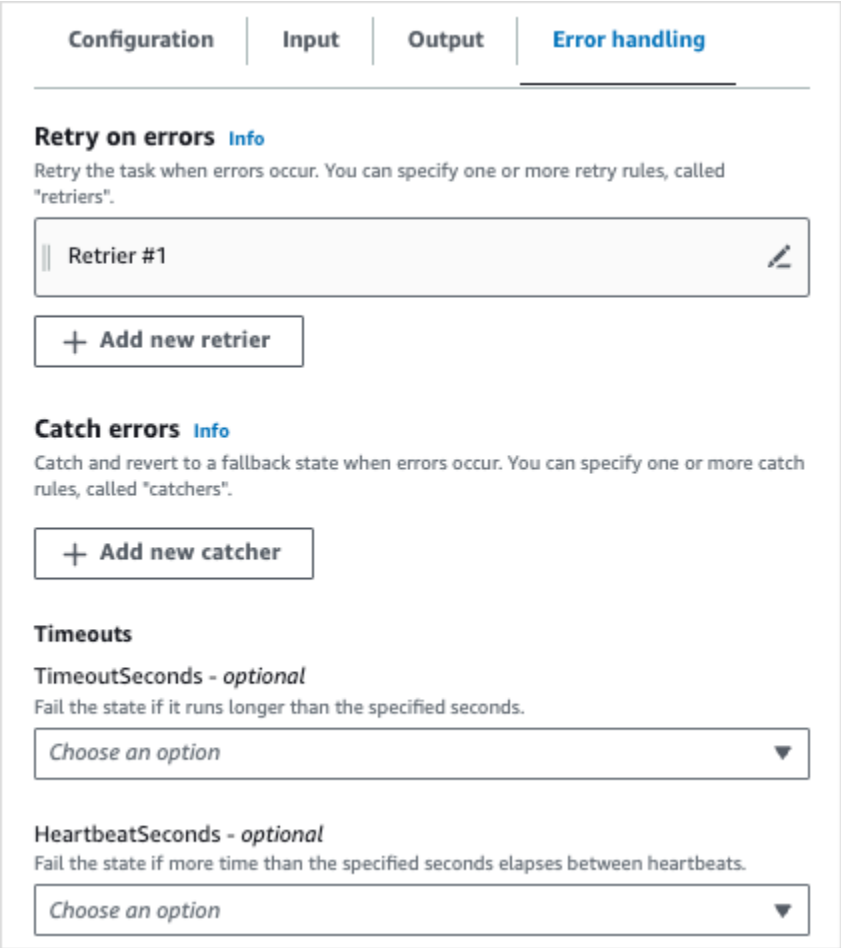
   c. [ステートをテスト] ダイアログボックスで、[実行ロール] ドロップダウンリストから [ロール ARN を入力] を選択します。

   d. ARN を [ロール ARN] に貼り付けます。

- 管理者アクセス権を持つロールを使用する – のすべてのサービスとリソースへのフルアクセス権を持つロールを作成するアクセス許可がある場合は AWS、そのロールを使用してワークフロー内の任意のタイプの状態をテストできます。これを行うには、Step Functionsサービスロールを作成し、IAMコンソール https://console.aws.amazon.com/iam/ でAdministratorAccess ポリシーを追加します。

エラー処理

デフォルトでは、ステートでエラーが報告されると、Step Functions の実行全体が失敗します。アクションといくつかのフロー状態については、Step Functions のエラー処理方法を設定できます。エラー処理を設定した場合でも、一部のエラーによってワークフロー実行が失敗する可能性があります。詳細については、「Step Functions のエラー処理」を参照してください。Workflow Studio で、エラー処理を Inspector パネルの [エラー処理] タブで設定します。



エラーを再試行

1 つ以上のルールをアクション状態と 並行 フロー状態に追加し、エラーが発生したときにタスクを再試行できます。これらのルールは retriers と呼ばれます。retrier を追加するには、[Retrier #1] ボックスで編集アイコンを選択し、そのオプションを設定します。

- (オプション) [コメント] フィールドで、コメントを追加します。ワークフローには影響しませんが、ワークフローに注釈を付けるために使用できます。
- [エラー] フィールドにカーソルを置き、retrier をトリガーするエラーを選択するか、カスタムエラー名を入力します。複数のエラーを選択または追加できます。
- (オプション) [Interval] (間隔) を設定します。これは、Step Functions が最初の再試行を行うまでの秒単位の時間です。追加の再試行は、[Max attempts] (最大試行回数) と [Backoff rate] (バックオフ率) で設定できる間隔の後で行われます。
- (オプション) [Max attempts] (最大試行回数) を設定します。これは、Step Functions の実行が失敗するまでの最大再試行回数です。

- (オプション) [Backoff rate] (バックオフ率) を設定します。これは、試行ごとに再試行間隔がどのくらい増加するかを決定する乗数です。

> **Note**
>
> どんなエラー処理オプションでも、あらゆる状態で利用できるわけではありません。Lambda Invoke には、デフォルトで retrier が 1 つが設定されています。

エラーの捕捉

1 つ以上のルールをアクション状態に追加し、エラーをキャッチするため、並行 と マッピング フロー状態に追加できます。これらのルールは catchers と呼ばれます。キャッチャーを追加するには、[Add new catcher] (新しいキャッチャーの追加) を選択し、そのオプションを設定します。

- (オプション) [コメント] フィールドで、コメントを追加します。ワークフローには影響しませんが、ワークフローに注釈を付けるために使用できます。
- [エラー] フィールドにカーソルを置き、catcher をトリガーするエラーを選択するか、カスタムエラー名を入力します。複数のエラーを選択または追加できます。
- [フォールバック状態] フィールドで、フォールバック状態を選択します。これは、エラーがキャッチされた後にワークフローが次へと移動する状態です。
- (オプション) ResultPathフィールドに、`ResultPath`元の状態入力にエラーを追加するフィルターを追加します。`ResultPath`は有効でなければなりませんJsonPath。これはフォールバック状態に送信されます。

タイムアウト

アクション状態のタイムアウトを設定して、失敗する前に状態を実行できる最大秒数を設定できます。タイムアウトを使用して実行のスタックを回避する タイムアウトを設定するには、実行が失敗するまでの状態の待機秒数を入力します。タイムアウトの詳細については、タスク ステートの「`TimeoutSeconds`」を参照してください。

HeartbeatSeconds

ハートビートまたは、タスクから定期的に送信される通知を設定できます。ハートビート間隔を設定し、設定された間隔で状態がハートビート通知を送信しない場合、タスクは失敗としてマークされま

す。ハートビートを設定するには、0 以外の正の整数の秒数を設定します。詳細については、タスクステートの「HeartBeatSeconds」を参照してください。

チュートリアル: AWS Step Functions Workflow Studio 使用を学ぶ

このチュートリアルでは、 AWS Step Functions用 Workflow Studio の基本的な使用方法を学習します。Workflow Studio の デザインモード で、Pass、Choice、Fail、Wait、Parallel などの複数の状態を含むステートマシンを作成します。ドラッグアンドドロップ機能を使用して、これらの状態を検索、選択、設定します。次に、自動生成されたワークフローの Amazon ステートメント言語 (ASL) 定義が表示されます。また、Workflow Studio の コードモード を使用して、ワークフロー定義を編集します。その後、Workflow Studio を終了してステートマシンを実行し、実行の詳細を確認します。

このチュートリアルでは、ステートマシンを更新し、実行出力の変更を確認する方法についても学習します。最後に、クリーンアップステップを実行し、ステートマシンを削除します。

このチュートリアルを完了すると、Workflow Studio を使用して、[デザイン] モードと [コード] モードの両方を使用してワークフローを作成および構成する方法を習得できます。また、ステートマシンを更新、実行、削除する方法も習得できます。

> **Note**
>
> 開始する前に、このチュートリアルの前提条件を必ず完了してください。

トピック

- ステップ 1: Workflow Studio に移動する
- ステップ 2: ステートマシンを作成する
- ステップ 3: 自動生成された Amazon States Language 定義を確認する
- ステップ 4: コードモードでワークフローの定義を編集する
- ステップ 5: ステートマシンを保存する
- ステップ 6: ステートマシンを実行する
- ステップ 7: ステートマシンを更新する
- ステップ 8: クリーンアップする

ステップ 1: Workflow Studio に移動する

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。

ステップ 2: ステートマシンを作成する

Workflow Studio では、ステートマシンはワークフローをグラフィカルに表示したものです。Workflow Studio を使用して、ワークフローの個々のステップを定義、設定、および検証できます。次のステップでは、Workflow Studio の デザインモード を使用してステートマシンを作成します。

ステートマシンを作成するには

1. Workflow Studio が[デザイン] モードになっていることを確認してください。
2. 左側の [State browser] (状態ブラウザ) で、[フロー] タブを選択します。次に、[パス] 状態を [最初の状態をここにドラッグ] とラベル付けされた空の状態にドラッグします。
3. [フロー] タブから [選択] 状態をドラッグし、[パス] 状態の下にドロップします。
4. [状態名] のデフォルト名を [選択] に置き換えます。このチュートリアルでは、名前に **`IsHelloWorldExample`** を使用します。
5. 別の Pass ステートをドラッグして、IsHelloWorldExampleそのステートの 1 つのブランチにドロップします。次に、Fail ステートをドラッグして、ステートのもう一方のブランチの下にドロップします。IsHelloWorldExample
6. [パス (1)] 状態を選択し、その名前を **`Yes`** に変更します。[失敗] 状態の名前を **`No`** に変更します。
7. boolean IsHelloWorldExample変数を使用してステートの分岐ロジックを指定します。`IsHelloWorldExample`

   `IsHelloWorldExample` が `False` の場合、ワークフローは [No] 状態を入力します。それ以外は、ワークフローは [Yes] 状態で実行フローを継続します。

   分岐ロジックを定義するには、次の手順を実行します。

   a. IsHelloWorldExampleでステートを選択しキャンバス、「Choice Rules」の「Rule #1」ボックスにある編集アイコンを選択して、最初の選択肢ルールを定義します。

b. [条件を追加] を選択します。

c. [ルール #1 の条件] ダイアログボックスで、[変数] に **$.IsHelloWorldExample** を入力します。

d. [演算子] で、[次と等しい] を選択します。

e. [値] で [ブール定数] を選択し、ドロップダウンリストから [true] を選択します。

f. [条件を保存する] を選択します。

g. [次の状態:] ドロップダウンリストで [Yes] が選択されていることを確認します。

h. [新しい選択ルールを追加] を選択し、次に [条件を追加] を選択します。

i. [ルール #2] ボックスで、サブステップ 7.c から 7.f を繰り返して、`IsHelloWorldExample` 変数の値が [false] のときの 2 番目の選択ルールを定義します。ステップ 7.e では、[true] の代わりに [false] を選択します。

j. [ルール #2] ボックスで、[次の状態:] ドロップダウンリストから [No] を選択します。

k. [デフォルトのルール] ボックスで、編集アイコンを選択してデフォルトの選択ルールを定義し、ドロップダウンリストから [Yes] を選択します。

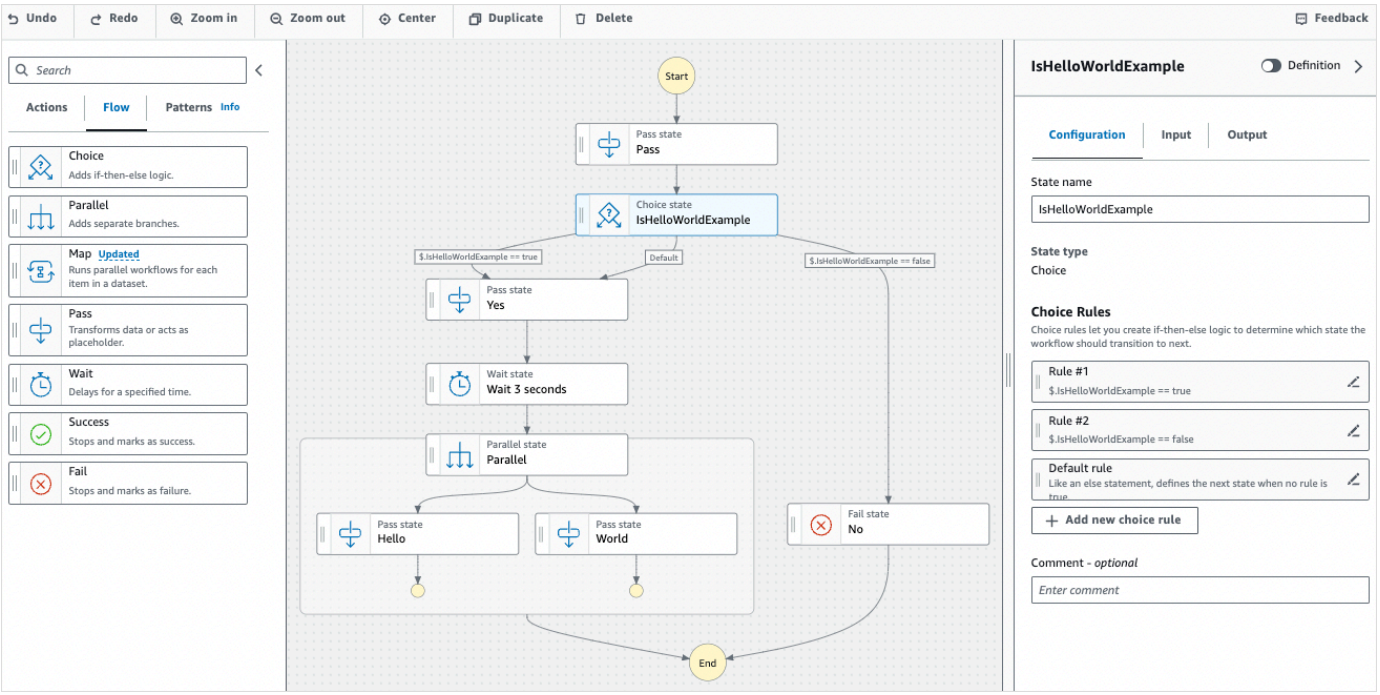
8. [Yes] 状態の後に [待機] 状態を追加し、**Wait 3 sec** という名前を付けます。次に、次の手順を実行して待機時間を 3 秒に設定します。

   a. [オプション] では、デフォルトの [一定の時間間隔を待機] のままにします。

   b. [秒] で [秒を入力] が選択されていることを確認し、ボックスに **3** を入力します。

9. [3 秒間待機] 状態になったら、[並列] ステートを追加します。さらに 2 つの [パス] 状態を 2 つの分岐に追加します。最初の [パス] 状態に **Hello** という名前を付けます。2 つ目の [パス] 状態に **World** という名前を付けます。

   完了したワークフローは次のようになります。



ステップ 3: 自動生成された Amazon States Language 定義を確認する

[フロー] タブから状態をキャンバスにドラッグアンドドロップすると、Workflow Studio はワークフローの Amazon States Language (ASL) の定義を、リアルタイムで自動的に作成します。Inspector パネルで、[定義] トグルボタンを選択してこの定義を表示するか、コードモード に切り替えて、必要に応じてこの定義を編集します。ワークフロー定義の編集については、このチュートリアルのステップ 4 を参照してください。

- (オプション) [Inspector] パネルで [定義] を選択し、ステートマシンのワークフローを表示します。

  次のサンプルコードは、ワークフロー用に自動的に生成された `IsHelloWorldExample` ステートマシンの Amazon States Language の定義を示しています。Workflow Studio で追加した `Choice` 状態は、ステップ 2 で定義した分岐ロジックに基づいて実行フローを決定するために使用されます。

```
{
  "Comment": "A Hello World example of the Amazon States Language using Pass
 states",
  "StartAt": "Pass",
  "States": {
    "Pass": {
```

```
    "Type": "Pass",
    "Next": "IsHelloWorldExample",
    "Comment": "A Pass state passes its input to its output, without performing
work. Pass states are useful when constructing and debugging state machines."
  },
  "IsHelloWorldExample": {
    "Type": "Choice",
    "Comment": "A Choice state adds branching logic to a state machine. Choice
rules can implement 16 different comparison operators, and can be combined using
And, Or, and Not\"",
    "Choices": [
      {
        "Variable": "$.IsHelloWorldExample",
        "BooleanEquals": false,
        "Next": "No"
      },
      {
        "Variable": "$.IsHelloWorldExample",
        "BooleanEquals": true,
        "Next": "Yes"
      }
    ],
    "Default": "Yes"
  },
  "No": {
    "Type": "Fail",
    "Cause": "Not Hello World"
  },
  "Yes": {
    "Type": "Pass",
    "Next": "Wait 3 sec"
  },
  "Wait 3 sec": {
    "Type": "Wait",
    "Seconds": 3,
    "Next": "Parallel"
  },
  "Parallel": {
    "Type": "Parallel",
    "End": true,
    "Branches": [
      {
        "StartAt": "Hello",
        "States": {
```

```
            "Hello": {
              "Type": "Pass",
              "End": true
            }
          }
        },
        {
          "StartAt": "World",
          "States": {
            "World": {
              "Type": "Pass",
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

ステップ 4: コードモードでワークフローの定義を編集する

Workflow Studio の コードモードには、ワークフローの ASL 定義を表示および編集するための統合コードエディタが用意されています。

1. [コード] を選択して、コードモードに切り替えます。
2. [並列] 状態が定義されたら、カーソルを置いて **Enter** を押します。
3. **Ctrl+space** を押すと、[並列] 状態の後に追加できる状態の一覧が表示されます。
4. オプションの一覧から [パス状態] を選択します。コードエディタは、[パス状態] のボイラープレートコードを追加します。
5. この状態を追加すると、ワークフロー定義にエラーが発生します。[並列] 状態の定義では、`"End": true` を **`"Next": "PassState"`** に置き換えてください。
6. 追加した [パス状態] の定義で、次の変更を実行します。

   a. [結果] ノードを削除します。

   b. `"ResultPath": "$.result"`, と `"Next": "NextState"` を削除します。

   c. `"Type": "Pass"`, の後で、**`"End": true`** と入力します。

   d. [パス状態] 定義の後で、**`,`** を追加します。

ワークフローの定義は、次のような定義になっているはずです。

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "IsHelloWorldExample"
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "Yes"
        },
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": false,
          "Next": "No"
        }
      ],
      "Default": "Yes"
    },
    "Yes": {
      "Type": "Pass",
      "Next": "Wait 3 seconds"
    },
    "Wait 3 seconds": {
      "Type": "Wait",
      "Seconds": 3,
      "Next": "Parallel"
    },
    "Parallel": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Hello",
          "States": {
            "Hello": {
              "Type": "Pass",
              "End": true
```

```
            }
          }
        },
        {
          "StartAt": "World",
          "States": {
            "World": {
              "Type": "Pass",
              "End": true
            }
          }
        }
      ],
      "Next": "PassState"
    },
    "PassState": {
      "Type": "Pass",
      "End": true
    },
    "No": {
      "Type": "Fail"
    }
  }
}
```

ステップ 5: ステートマシンを保存する

1. Config more を選択するか、デフォルトのステートマシン名の横にある編集アイコンを選択しますMyStateMachine。[ステートマシンの設定] で、名前を指定します。たとえば、**HelloWorld** と入力します。
2. (オプション) ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。
3. [作成] を選択します。
4. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して、[設定] モードに戻ることもできます。

[設定] モードについての詳細は、「Workflow Studio の設定モード」を参照してください。

ステップ 6: ステートマシンを実行する

ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。

1. ステートマシンページで、HelloWorldステートマシンを選択します。
2. HelloWorldページで [実行開始] を選択します。
3. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

4. [入力] ボックスに、実行の入力値を JSON 形式で入力します。入力に基づいて、`IsHelloWorldExample` 変数は、どのステートマシンフローが実行されるかを決定します。ここでは、次の入力値を使用します。

   ```
   {
      "IsHelloWorldExample": true
   }
   ```

   > **Note**
   >
   > 実行入力の指定はオプションですが、このチュートリアルでは、上記の入力例と同様の実行入力を指定することが必要です。この入力値は、ステートマシンを実行する際に `Choice` 状態で参照されます。

5. [実行のスタート] を選択します。
6. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実

行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

このチュートリアルでは、`"IsHelloWorldExample": true` の入力値を入力すると、次のような出力が表示されます。

```
{
   "IsHelloWorldExample": true
},
{
   "IsHelloWorldExample": true
}
```

ステップ 7: ステートマシンを更新する

ステートマシンを更新すると、更新は結果に整合します。短時間のうちに、新しく開始したすべての実行にステートマシンの更新された定義が反映されます。現在実行中のすべての実行は、前の定義に基づいて完了します。

このステップでは、Workflow Studio の デザインモード モードでステートマシンを更新します。[パス] 状態に [World] という名前の `Result` フィールドを追加します。

1. 実行 ID のタイトルのページで、[Edit state machine] (ステートマシンの編集) を選択します。
2. [デザイン] モードになっていることを確認してください。
3. キャンバスで [World] という名前の [パス] 状態を選択し、次に [出力] を選択します。
4. [結果] ボックスに、**`"World has been updated!"`** と入力します。
5. [保存] を選択します。
6. (オプション) [定義] 領域では、ワークフローの更新された Amazon States Language の定義を表示します。

```
{
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Hello",
          "States": {
            "Hello": {
```

```
            "Type": "Pass",
            "End": true
          }
        }
      },
      {
        "StartAt": "World",
        "States": {
          "World": {
            "Type": "Pass",
            "Result": "World has been updated!",
            "End": true
          }
        }
      }
    ],
    "Next": "PassState"
  }
```

7. [実行] を選択します。
8. 新しいタブで [実行を開始] ダイアログボックスを開き、次の実行入力を指定します。

```
{
   "IsHelloWorldExample": true
}
```

9. [実行のスタート] を選択します。
10. (オプション) [グラフビュー] で [World] ステップを選択し、次に [出力] を選択します。この出力は、「World が更新されました」です。

ステップ 8: クリーンアップする

ステートマシンを削除するには

1. ナビゲーションメニューから、[State machines] (ステートマシン) を選択します。
2. 「ステートマシン」ページで、を選択しHelloWorld、「削除」を選択します。
3. [ステートマシンの削除] ダイアログボックスに、**delete** と入力して削除を確定します。
4. [削除] をクリックします。

正常に削除されると、緑色のステータスバーが画面の上部に表示されます。緑色のステータスバーには、ステートマシンが削除対象としてマークされていると書かれています。進行中のすべての実行が停止すると、ステートマシンは削除されます。

実行ロールを削除するには

1. IAM 用 [Roles] (ロール) ページを開きます。
2. Step Functions が作成した IAM ロールを選択します。たとえば、StepFunctions-HelloWorld-ロール例。
3. [Delete role] (ロールの削除) を選択します。
4. [Yes, delete] (はい、削除します) を選択します。

Step Functions チュートリアル

このセクションのチュートリアルは、 AWS Step Functions を使用した作業のさまざまな側面を理解するのに役立ちます。

これらのチュートリアルを完了するには、 AWS アカウントが必要です。 AWS アカウントをお持ちでない場合は、https://aws.amazon.com/ に移動し、 AWS アカウントの作成を選択します。

トピック

- Lambda を使用する Step Functions ステートマシン状態の作成
- Step Functions ステートマシンを使用してエラー条件を処理する
- インラインマップステートを使用してアクションを反復する
- 分散マップを使用した大規模 CSV データのコピー
- Lambda 関数でデータバッチ全体を処理する
- Lambda 関数で個々のデータ項目を処理する
- Amazon S3 イベント発生時にステートマシンの実行をスタートする
- API Gateway を使用した Step Functions API の作成
- AWS SAM を使用して Step Functions ステートマシンを作成する
- Step Functions を使用してアクティビティステートマシンを作成する
- Lambda を使用してループを反復処理する
- 長時間実行されているワークフロー実行を新しい実行として継続する
- 人間による承諾プロジェクト例をデプロイする
- Step Functions で X-Ray によるトレースを表示する
- AWS SDK サービスインテグレーションを使用して Amazon S3 バケット情報を収集する

Lambda を使用する Step Functions ステートマシン状態の作成

このチュートリアルでは、 AWS Step Functions 関数を呼び出すための単一ステップのワークフローを作成します。 AWS Lambda

> **Note**
>
> Step Functions はステートマシンとタスクに基づいています。Step Functions では、ワークフローをステートマシンと呼びます。ステートマシンは、一連のイベント駆動型ステップで

す。ワークフローの各ステップはステートと呼ばれます。タスクステートは AWS Lambda、AWS などの別のサービスが実行する作業単位を表します。タスクステートは任意の API AWS のサービス または API を呼び出すことができます。
詳細については、以下を参照してください。

- とは何ですか AWS Step Functions?
- AWS 他のサービスに電話してください。

Lambda 関数はサーバーレスで、記述が容易なため、Lambda は `Task` 状態に適しています。 AWS Management Console またはお気に入りのエディタでコードを記述できます。 AWS 関数用のコンピューティング環境の提供と実行の詳細を処理します。

このトピックの内容

- ステップ 1: Lambda 関数を作成する
- ステップ 2: Lambda 関数をテストする
- ステップ 3: ステートマシンを作成する
- ステップ 4: ステートマシンを実行する

ステップ 1: Lambda 関数を作成する

Lambda 関数は、イベントデータを受け取り、グリーティングメッセージを返します。

> **Important**
>
> Lambda AWS AWS 関数がステートマシンと同じアカウントとリージョンにあることを確認してください。

1. Lambda コンソールを開き、[関数を作成] を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。
3. [関数名] に `HelloFunction` と入力します。
4. その他のすべてのオプションはデフォルトのまま選択して、[関数を作成] を選択します。
5. Lambda 関数が作成されたら、ページの右上隅に表示されている関数の Amazon リソースネーム (ARN) をコピーします。ARN をコピーするに

は、
をクリックします。ARN の例を次に示します。

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

6. Lambda 関数の次のコードをページの [コードソース] セクションにコピーします。***HelloFunction***

```
export const handler = async(event, context, callback) => {
    callback(null, "Hello from " + event.who + "!");
};
```

このコードは、関数に渡された event オブジェクトから提供された入力データの who フィールドを使用して、挨拶をアセンブルします。後で新しい実行をスタートするときに、この関数の入力データを追加します。callback メソッドによって、関数からアセンブルされた挨拶が返されます。

7. デプロイを選択します。

ステップ 2: Lambda 関数をテストする

Lambda 関数をテストしてオペレーションを確認します。

1. [テスト] を選択します。
2. イベント名()で、HelloEvent と入力します。
3. Event JSON データを次のものに置き換えます。

```
{
    "who": "AWS Step Functions"
}
```

"who" エントリは Lambda 関数の event.who フィールドに対応し、挨拶を完了させます。ステートマシンを実行する場合にも、同じ入力データを入力します。

4. [保存] を選択し、次に [テスト] を選択します。
5. テスト結果を確認するには、[Execution result] (実行結果) で、[Details] (詳細) を展開します。

ステップ 3: ステートマシンを作成する

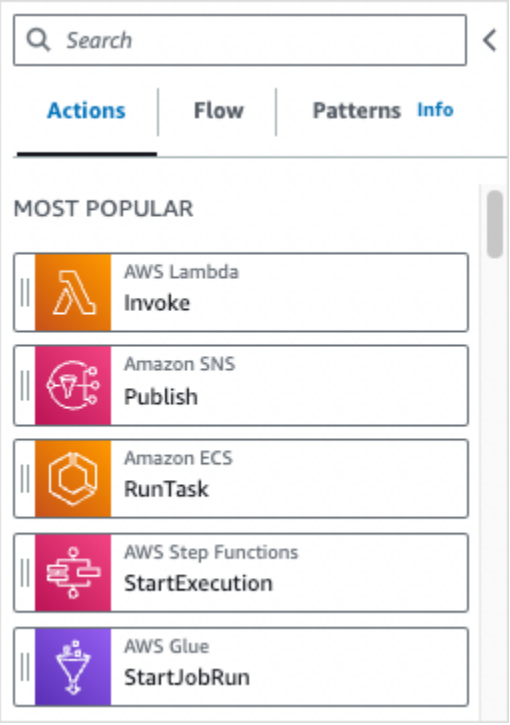
Step Functions コンソールを使用して、ステップ 1 で作成した Lambda 関数を呼び出すステートマシンを作成します。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

   > **Important**
   >
   > ステートマシンが、前に作成した Lambda AWS 関数と同じアカウントとリージョンにあることを確認してください。

2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. 左側の[状態ブラウザ] で、[アクション] タブが選択されていることを確認します。次に、以下の操作を実行します。

   - AWS Lambda 呼び出し API を [最初の状態をここにドラッグ] とラベル付けされた空の状態にドラッグします。



5. 右側の [Inspector] パネルで、Lambda 関数を設定します。

a. [API パラメータ] セクションで、[関数名] ドロップダウンリストで、以前に作成した Lambda 関数を選択します。

b. [ペイロード] ドロップダウンリストでは、デフォルトの選択をそのまま使用します。

6. (オプション) [定義] を選択すると、ステートマシンの Amazon ステートメント言語 (ASL) の定義が表示されます。この定義は、[アクション] タブと [Inspector] パネルの選択によって自動的に生成されます。

7. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を入力します。

   例えば、名前を **`LambdaStateMachine`** と入力します。

   > **Note**
   >
   > ステートマシン、実行、アクティビティタスクの名前は 80 文字以下にする必要があります。 AWS これらの名前はアカウントと地域で一意である必要があり、次のものを含んではいけません。
   >
   > - 空白
   > - ワイルドカード文字 (? *)
   > - 角かっこ (< > { } [ ])
   > - 特殊文字 (: ; , \ | ^ ~ $ # % & ` ")
   > - 制御文字 (\\u0000 - \\u001f または \\u007f - \\u009f)
   >
   > ステートマシンのタイプが [Express] の場合、ステートマシンを複数回実行する際に同じ名前を指定できます。Step Functions では、複数の実行が同じ名前であっても、Express ステートマシンの実行ごとに一意の実行 ARN を生成します。
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

8. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

9. [作成] を選択します。
10. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

    [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 4: ステートマシンを実行する

ステートマシンを作成した後、それを実行できます。

1. ステートマシンページで、を選択します。LambdaStateMachine
2. [実行のスタート] を選択します。

   [実行を開始] ダイアログが表示されます。
3. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

4. [入力] 領域で、サンプルの実行データを次のものに置き換えます。

```
{
    "who" : "AWS Step Functions"
```

```
}
```

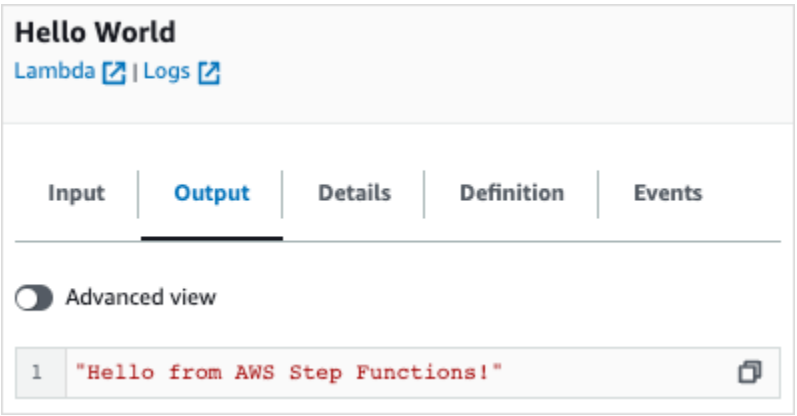
"who" は Lambda 関数が挨拶する相手の名前を取得するために使用するキー名です。

5. [実行のスタート] を選択します。

   ステートマシンの実行が開始され、実行中の実行が表示されている新しいページが表示されます。

6. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。



> **Note**
>
> ステートマシンから Lambda を呼び出す際にペイロードを渡すこともできます。`Parameters` フィールドにペイロードを渡すことによって Lambda を呼び出す方法の詳細と例については、「Step Functions で Lambda を呼び出す」を参照してください。

Step Functions ステートマシンを使用してエラー条件を処理する

このチュートリアルでは、 AWS Step Functions Fallback 状態フィールドを持つステートマシンを作成します。Catch AWS Lambda フィールドは関数を使用して、エラーメッセージタイプに基づいて条件付きロジックで応答します。これは、関数エラー処理と呼ばれるテクニックです。

詳細については、「AWS Lambda デベロッパーガイド」の「Node.js のAWS Lambda 関数エラー」を参照してください。

> **Note**
>
> また、タイムアウト時に再試行するステートマシンや、エラーやタイムアウトが発生したときに、Catch を使用して特定の状態に移行するステートマシンを作成できます。これらのエラー処理方法の例については、Retry の使用例と Catch の使用例を参照してください。

このトピックの内容

- ステップ 1: 失敗する Lambda 関数を作成する
- ステップ 2: Lambda 関数をテストする
- ステップ 3: Catch フィールドを使用するステートマシンを作成する
- ステップ 4: ステートマシンを実行する

ステップ 1: 失敗する Lambda 関数を作成する

Lambda 関数を使用してエラー条件をシミュレートします。

> **Important**
>
> Lambda AWS AWS 関数がステートマシンと同じアカウントとリージョンにあることを確認してください。

1. https://console.aws.amazon.com/lambda/ AWS Lambda でコンソールを開きます。
2. [関数を作成] を選択します。
3. [ブループリントを使う] を選択し、検索ボックスに step-functions を入力してから [カスタムエラーをスローする] ブループリントを選択します。

4. [関数名] に FailFunction と入力します。
5. [ロール] は、デフォルトの選択 ([基本的な Lambda アクセス許可で新しいロールを作成する]) のままにします。
6. 次のコードが [Lambda function cod] (関数コード) ペインに表示されます。

```
exports.handler = async (event, context) => {
    function CustomError(message) {
        this.name = 'CustomError';
        this.message = message;
    }
    CustomError.prototype = new Error();

    throw new CustomError('This is a custom error!');
};
```

   context オブジェクトはエラーメッセージ This is a custom error! を返します。
7. [関数を作成] を選択します。
8. Lambda 関数が作成されたら、ページの右上隅に表示されている関数の Amazon リソースネーム (ARN) をコピーします。ARN をコピーするには、
   をクリックします。ARN の例を次に示します。

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

9. デプロイを選択します。

ステップ 2: Lambda 関数をテストする

Lambda 関数をテストしてオペレーションを確認します。

1. FailFunctionページで [テスト] タブを選択し、[テスト] を選択します。テストイベントを作成する必要はありません。
2. テスト結果 (シミュレートしたエラー) を確認するには、[実行結果] で、[詳細] を展開します。

ステップ 3: Catch フィールドを使用するステートマシンを作成する

Step Functions コンソールにより、タスク 状態で Catch フィールドを使用するステートマシンを作成します。Task 状態に Lambda 関数へのリファレンスを追加します。ステートマシンは Lambda 関数を呼び出しますが、実行中に失敗します。Step Functions は、再試行間のエクスポネンシャルバックオフを使用して、関数を 2 回再試行します。

1. Step Functions コンソールを開き、[Create a state machine] (ステートマシンの作成)] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. [コード] を選択してコードエディタを開きます。コードエディタで、ワークフローの Amazon ステートメント言語 (ASL) 定義を記述して編集します。
5. 次のコードを貼り付けます。ただし、Resource フィールドで以前に作成した Lambda 関数の ARN を置き換えます。

```
{
   "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
      }
   }
}
```

これは、Amazon ステートメント言語を使用したステートマシンの説明です。CreateAccount という名前の単一の Task 状態を定義します。詳細については、「State Machine Structure」を参照してください。

Retry フィールドの構文の詳細については、Retry と Catch を使用するステートマシンの例 を参照してください。

> **Note**
>
> Lambda での未処理のエラーは、エラー出力で Lambda.Unknown として報告されます。これらには、 out-of-memory エラーや関数のタイムアウトが含まれます。Lambda.Unknown、States.ALL、または States.TaskFailed を一致させて、こういったエラーに処理できます。Lambda が最大呼び出し数に達すると、エラーは Lambda.TooManyRequestsException となります。Lambda 関数のエラーの詳細については、「AWS Lambda デベロッパーガイド」の「エラー処理と自動再試行」を参照してください。

6. (オプション) グラフ可視化 では、ワークフローをリアルタイムでグラフィカルに視覚化して表示します。
7. ステートマシンの名前を指定します。そのためには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を入力します。

   このチュートリアルでは、**Catchfailure** と入力します。

8. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

9. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 4: ステートマシンを実行する

ステートマシンを作成した後、それを実行できます。

1. [ステートマシン] ページで [Catchfailure] を選択します。
2. [Catchfailure] ページで、[実行を開始] を選択します。[実行を開始] ダイアログが表示されます。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

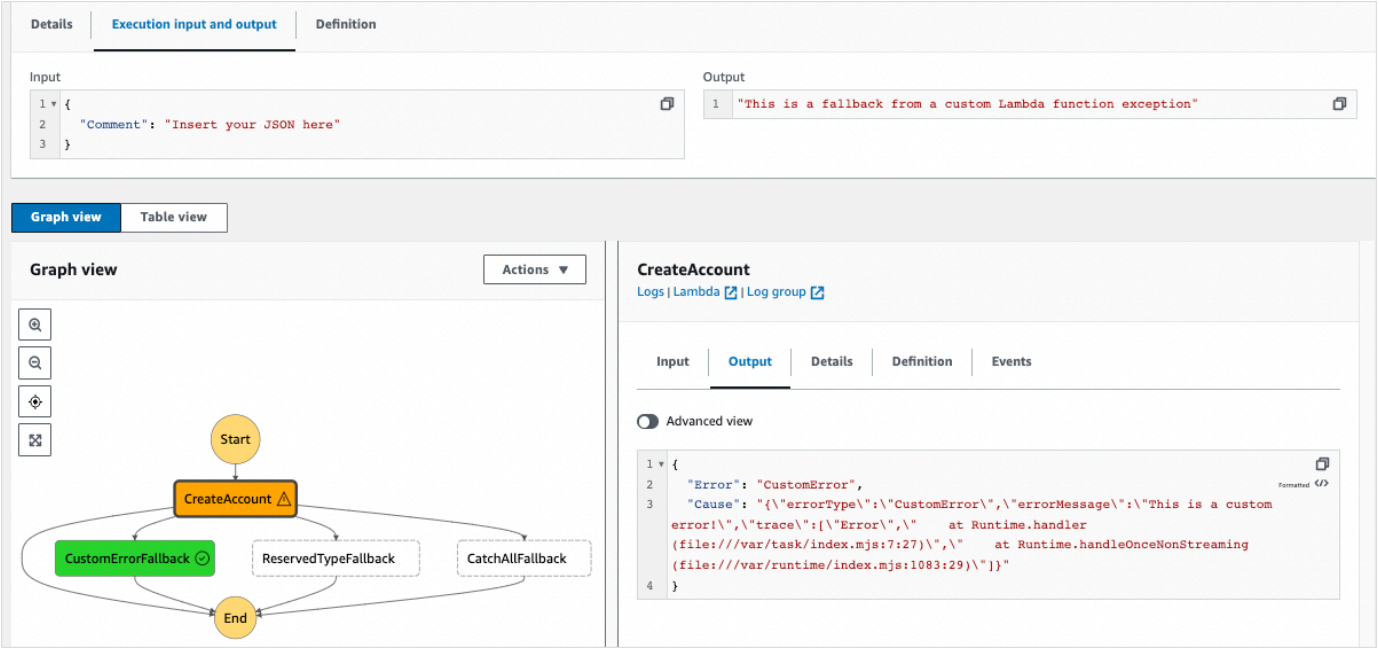
      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。
   3. [実行のスタート] を選択します。

4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

たとえば、カスタムエラーメッセージを表示するには、CreateAccountグラフビューでステップを選択し、次に [出力] タブを選択します。



> **Note**
>
> 状態入力とエラーを保持するには、ResultPath を使用します。「ResultPath を使用して、エラーと入力の両方を に含める Catch」を参照してください。

インラインマップステートを使用してアクションを反復する

このチュートリアルでは、インラインモードで Map 状態を使用する方法を説明します。ワークフローで、インラインマップステートを使用してアクションを繰り返し実行します。インラインモードの詳細については、「インラインモードのマップステート」を参照してください。

このチュートリアルでは、インラインマップステートを使用して、バージョン 4 の汎用一意識別子 (v4 UUID) を繰り返し生成します。まず、Workflow Studio で 2 つの パス ステートと 1 つのインラインマップステートを含むワークフローを作成します。次に、Map 状態の入力 JSON 配列を含め、入力と出力を設定します。Map 状態は、入力配列内の各項目に対して生成された v4 UUID を含む出力配列を返します。

コンテンツ

- ステップ 1: ワークフロープロトタイプを作成する
- ステップ 2: 入力と出力を構成する
- ステップ 3: 自動生成された Amazon States Language 定義を確認してワークフローを保存する
- ステップ 4: ステートマシンを実行する

ステップ 1: ワークフロープロトタイプを作成する

このステップでは、Workflow Studio を使用してワークフロープロトタイプを作成します。Workflow Studio は、Step Functions コンソールで使用可能なビジュアルワークフローデザイナーです。[フロー] タブから必要な状態を選択し、Workflow Studio のドラッグアンドドロップ機能を使用してワークフロープロトタイプを作成します。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. [フロー] タブで、[パス] 状態をドラッグし、[最初の状態をここにドラッグ] とラベル付けされた空の状態にドロップします。
5. [マップ] 状態をドラッグして、[パス] 状態の下にドロップします。[マップ] 状態の名前を **Map demo** に変更します。
6. 2 つ目の [パス] 状態をドラッグして [マップデモ] 状態の中にドロップします。
7. 2 つ目の [パス] 状態の名前を **Generate UUID** に変更します。

ステップ 2: 入力と出力を構成する

このステップでは、ワークフロープロトタイプのすべての状態に対する入力と出力を設定します。まず、最初の [パス] 状態を使用して、ワークフローに固定データを挿入します。[パス] 状態は、このデータを [マップデモ] 状態への入力として渡します。この入力の中で、[マップデモ] 状態が反復処理する必要がある入力配列を含むノードを指定します。次に、[マップデモ] 状態が v4 UUID を生成す

るために反復する必要があるステップを定義します。最後に、反復するごとに返される出力を構成します。

1. ワークフロープロトタイプの最初の [パス] 状態を選択します。[出力] タブで、[結果] に次のものを入力します。

```
{
  "foo": "bar",
  "colors": [
    "red",
    "green",
    "blue",
    "yellow",
    "white"
  ]
}
```

2. [マップデモ] 状態を選択し、[構成]タブで次の操作を行います。

   a. [項目配列へのパスを指定] を選択します。

   b. 次の [リファレンスパス] を指定して、入力配列を含むノードを選択します。

```
$.colors
```

3. [UUID の生成] 状態を選択し、[入力] タブで次の操作を行います。

   a. [Parameters を使用して入力を変換] を選択します。

   b. 次の JSON 入力を入力して、入力配列の各項目の v4 UUID を生成します。`States.UUID` 組み込み関数を使用して UUID を生成します。

```
{
  "uuid.$": "States.UUID()"
}
```

4. [UUID の生成] 状態で [出力] タブを選択し、次の操作を行います。

   a. [出力のフィルタリング] を選択します OutputPath。

   b. 次の [リファレンスパス] を入力して、出力配列を含む JSON ノードを選択します。

```
$.uuid
```

ステップ 3: 自動生成された Amazon States Language 定義を確認してワークフローを保存する

[フロー] パネルから状態をキャンバスにドラッグアンドドロップすると、Workflow Studio はワークフローの Amazon States Language (ASL) の定義を、リアルタイムで自動的に作成します。この定義は、必要に応じて編集できます。

1. (オプション) Inspector パネルの [定義] を選択すると、ワークフローの自動生成された Amazon States Language の定義が表示されます。

   > **Tip**
   >
   > また、Workflow Studio の コードエディタ で ASL の定義を表示できます。コードエディタで、ワークフローの ASL の定義を編集することもできます。

   次の例は、ワークフロー用に自動的に生成された Amazon States Language の定義を示しています。

```
{
    "Comment": "Using Map state in Inline mode",
    "StartAt": "Pass",
    "States": {
      "Pass": {
        "Type": "Pass",
        "Next": "Map demo",
        "Result": {
          "foo": "bar",
          "colors": [
            "red",
            "green",
            "blue",
            "yellow",
            "white"
          ]
        }
      },
      "Map demo": {
        "Type": "Map",
        "ItemsPath": "$.colors",
        "ItemProcessor": {
```

```
          "ProcessorConfig": {
            "Mode": "INLINE"
          },
          "StartAt": "Generate UUID",
          "States": {
            "Generate UUID": {
              "Type": "Pass",
              "End": true,
              "Parameters": {
                "uuid.$": "States.UUID()"
              },
              "OutputPath": "$.uuid"
            }
          }
        },
        "End": true
      }
    }
  }
```

2. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択しますMyStateMachine。次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**InlineMapDemo**」と入力します。

3. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

4. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 4: ステートマシンを実行する

ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。

1. InlineMapDemoページで [実行開始] を選択します。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。
   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

   実行の入力と出力を表示するには side-by-side、「実行の入力と出力」を選択します。[出力] に、Map 状態から返された出力配列が表示されます。出力配列の例を次に示します。

```
[
  "a85cbc7b-4e65-4ac2-97af-80ed504adc1d",
  "b05bca11-d481-414e-aa9a-88285ec6590d",
  "f42d59f7-bd32-480f-b270-caddb518ce2a",
  "15f18616-517d-4b69-b7c3-bf22222d2efd",
  "690bcfee-6d58-408c-a6b4-1995ccafdbd2"
]
```

分散マップを使用した大規模 CSV データのコピー

このチュートリアルでは、分散モードで Map 状態を使用開始する方法を説明します。[分散] に設定された Map ステートは、分散マップ状態と呼ばれます。ワークフローで分散マップ状態を使用し、大規模な Amazon S3 データソースを反復処理します。Map 状態は、それぞれの反復を子ワークフロー実行として実行するため、高い同時実行性が得られます。分散モードについての詳細は、「分散モードでのマップステート」を参照してください。

このチュートリアルでは、分散マップ状態を使用して、Amazon S3 バケット内の CSV ファイルを反復処理します。次に、その内容と子ワークフロー実行の ARN を別の Amazon S3 バケットに返します。最初に、Workflow Studio でワークフロープロトタイプを作成します。次に、Map 状態の処理モード を [分散] に設定し、CSV ファイルをデータセットとして指定し、その場所を Map 状態に提供します。また、[Express] として分散マップ状態を開始する子ワークフロー実行のワークフロータイプを指定します。

これらの設定に加えて、このチュートリアルで使用するワークフローの例では、子ワークフローの同時実行の最大数や Map の結果をエクスポートする場所など、その他の設定も指定します。

コンテンツ

- 前提条件
- ステップ 1: ワークフロープロトタイプを作成する
- ステップ 2: マップステートの必須フィールドを設定する
- ステップ 3: 追加オプションを設定する
- ステップ 4: Lambda 関数を設定する
- ステップ 5: ワークフロープロトタイプを更新する
- ステップ 6: 自動生成された Amazon States Language 定義を確認してワークフローを保存する
- ステップ 7: ステートマシンを実行する

前提条件

- CSV ファイルを Amazon S3 バケットにアップロードする CSV ファイル内にヘッダー行を定義する必要があります。CSV ファイルに適用されるサイズ制限とヘッダー行の指定方法については、「Amazon S3 バケット内の CSV ファイル」を参照してください。
- 別の Amazon S3 バケットを作成し、そのバケット内に Map 状態の結果を エクスポートするフォルダを作成します。

> **Important**
>
> Amazon S3 AWS アカウント バケットがステートマシンと同じ下にあることを確認してください。 AWS リージョン

ステップ 1: ワークフロープロトタイプを作成する

このステップでは、Workflow Studio を使用してワークフロープロトタイプを作成します。Workflow Studio は、Step Functions コンソールで使用可能なビジュアルワークフローデザイナーです。必要な状態と API アクションを [フロー] タブと [アクション] タブのそれぞれから選択します。Workflow Studio のドラッグアンドドロップ機能を使用して、ワークフロープロトタイプを作成します。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. [フロー] タブで、[マップ] 状態をドラッグし、[最初の状態をここにドラッグ] とラベル付けされた空の状態にドロップします。
5. [設定] タブの [状態名] に **`Process data`** と入力します。
6. [アクション] タブから [AWS Lambda 呼び出し] API アクションをドラッグし、[データの処理] 状態の中にドロップします。
7. AWS Lambda 呼び出し 状態の名前を **`Process CSV data`** に変更します。

ステップ 2: マップステートの必須フィールドを設定する

このステップでは、分散マップ状態に必要な次のフィールドを設定します。

- ItemReader— `Map` ステートが入力を読み取ることができるデータセットとその場所を指定します。
- ItemProcessor - 次の値を指定します。
  - `ProcessorConfig` - `Mode` と `ExecutionType` を、それぞれ `DISTRIBUTED` と `EXPRESS` に設定します。これにより、`Map` ステートの処理モードと、分散マップ状態 が開始する子ワークフロー実行のワークフロータイプが設定されます。
  - `StartAt` - マップワークフローの最初のステート。

- `States` - マップワークフローを定義します。マップワークフローは、子ワークフローを実行するたびに繰り返す一連のステップです。
- ResultWriter— Step Functions が分散マップの状態結果を書き込む Amazon S3 の場所を指定します。

> **Important**
>
> Map Run の結果をエクスポートするために使用する Amazon S3 バケットが AWS アカウント 、 AWS リージョン ステートマシンと同じ場所にあることを確認してください。それ以外の場合は、`States.ResultWriterFailed` エラーが発生してステートマシンの実行が失敗します。

必須フィールドを設定するには:

1. [データの処理] 状態を選択し、[設定] タブで次の操作を行います。
   a. [処理モード] は、[分散] を選択します。
   b. [項目ソース] は、[Amazon S3] を選択し、[S3 項目ソース] ドロップダウンリストから [S3 の CSV ファイル] を選択します。
   c. CSV ファイルの Amazon S3 の場所を指定するには、次の手順を実行します。
      i. [S3 オブジェクト] は、ドロップダウンリストから [バケットとキーを入力] を選択します。
      ii. [バケット] には、CSV ファイルを含む Amazon S3 バケットの名前を入力します。例えば **`sourceBucket`** です。
      iii. [キー] には、CSV ファイルを保存した Amazon S3 オブジェクトの名前を入力します。また、このフィールドには、CSV ファイルの名前を指定する必要があります。例えば **`csvDataset/ratings.csv`** です。
   d. CSV ファイルの場合は、列ヘッダーの場所も指定する必要があります。これを指定するには、[追加設定] を選択し、CSV ファイルの最初の行がヘッダーの場合は、[CSV ヘッダーの場所] をデフォルトの [最初の行] のままにします。それ以外の場合は、[指定済み] を選択してステートマシン定義内のヘッダーを指定します。詳細については、「`ReaderConfig`」を参照してください。
   e. [子実行タイプ] には、[Express] を選択します。

2. [エクスポートの場所] で、マップの実行結果を特定の Amazon S3 の場所にエクスポートするには、[マップステートの出力を Amazon S3 にエクスポートする] を選択します。
3. 以下の操作を実行します。
   a. [S3 バケット] は、ドロップダウンリストから [バケット名とプレフィックスを入力する] を選択します。
   b. [バケット] には、結果をエクスポートする Amazon S3 バケットの名前を入力します。例えば **mapOutputs** です。
   c. [プレフィックス] には、結果を保存するフォルダ名を入力します。例えば **resultData** です。

ステップ 3: 追加オプションを設定する

分散マップ状態に必要な設定に加えて、他のオプションも指定できます。これには、同時に実行する子ワークフローの最大数や、Map 状態の結果をエクスポートする場所などが含まれます。

1. データの処理状態の場所を指定します。次に、[項目ソース] で [追加設定] を選択します。
2. 以下の操作を実行します。
   a. [Modify items with ItemSelector] を選択して、子ワークフローを実行するたびにカスタム JSON 入力を指定します。
   b. 次の JSON 入力を入力します。

```
{
  "index.$": "$$.Map.Item.Index",
  "value.$": "$$.Map.Item.Value"
}
```

   カスタム入力の作成方法については、「ItemSelector」を参照してください。
3. [ランタイムの設定] の [同時実行数の制限] で、分散マップ状態で開始できる子ワークフローの同時実行数を指定します。たとえば、**100** と入力します。
4. ステップ 4: Lambda 関数を設定する で説明されているように、ブラウザで新しいウィンドウまたはタブを開き、このワークフローで使用する Lambda 関数の設定を完了します。

ステップ 4: Lambda 関数を設定する

> **Important**
>
> Lambda AWS リージョン 関数がステートマシンと同じ下にあることを確認してください。

1. Lambda コンソールを開き、[関数を作成] を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。
3. [Basic information] (ベーシックな情報) セクションで、Lambda 関数を構成:
   a. [関数名] に **distributedMapLambda** と入力します。
   b. [ランタイム] で [Node.js 16.x] を選択します。
   c. デフォルトの選択をすべて維持して、[関数を作成] を選択します。
   d. Lambda 関数を作成したら、ページの右上隅に表示されている関数の Amazon リソースネーム (ARN) をコピーします。これをワークフロープロトタイプに入力する必要があります。ARN をコピーするには、
   をクリックします。ARN の例を次に示します。

   ```
   arn:aws:lambda:us-east-2:123456789012:function:distributedMapLambda
   ```

4. Lambda 関数の次のコードをコピーして、ページの [コードソース] セクションに貼り付けます。distributedMapLambda

   ```
   exports.handler = async function(event, context) {
       console.log("Received Input:\n", event);

       return {
           'statusCode' : 200,
           'inputReceived' : event //returns the input that it received
       }
   };
   ```

5. デプロイを選択します。関数がデプロイされたら、[テスト] を選択して Lambda 関数の出力を確認します。

ステップ 5: ワークフロープロトタイプを更新する

Step Functions コンソールで、ワークフローを更新して Lambda 関数の ARN を追加します。

1. ワークフロープロトタイプを作成したタブまたはウィンドウに戻ります。
2. [CSV データの処理] ステップを選択し、[設定] タブで次の操作を行います。

   a. [統合タイプ] は、[最適化] を選択します。
   b. [関数名] には、Lambda 関数の名前を入力します。表示されるドロップダウンリストから関数を選択するか、[関数名を入力] を選択して Lambda 関数の ARN を入力します。

ステップ 6: 自動生成された Amazon States Language 定義を確認してワークフローを保存する

[アクション] タブおよび [フロー] タブから状態をキャンバスにドラッグアンドドロップすると、Workflow Studio はワークフローの Amazon States Language の定義を、リアルタイムで自動的に作成します。この定義は、必要に応じて編集できます。

1. (オプション) Inspector パネルで [定義] を選択して、ステートマシンの定義を表示します。

   > **Tip**
   >
   > また、Workflow Studio の コードエディタ で ASL の定義を表示できます。コードエディタで、ワークフローの ASL の定義を編集することもできます。

   次のサンプルコードは、ワークフロー用に自動的に生成された Amazon States Language の定義を示しています。

```
{
  "Comment": "Using Map state in Distributed mode",
  "StartAt": "Process data",
  "States": {
    "Process data": {
      "Type": "Map",
      "MaxConcurrency": 100,
      "ItemReader": {
        "ReaderConfig": {
          "InputType": "CSV",
```

```
          "CSVHeaderLocation": "FIRST_ROW"
        },
        "Resource": "arn:aws:states:::s3:getObject",
        "Parameters": {
          "Bucket": "sourceBucket",
          "Key": "csvDataset/ratings.csv"
        }
      },
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "EXPRESS"
        },
        "StartAt": "Process CSV data",
        "States": {
          "Process CSV data": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
              "Payload.$": "$",
              "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:distributedMapLambda"
            },
            "End": true
          }
        }
      },
      "Label": "Processdata",
      "End": true,
      "ResultWriter": {
        "Resource": "arn:aws:states:::s3:putObject",
        "Parameters": {
          "Bucket": "mapOutputs",
          "Prefix": "resultData"
        }
      },
      "ItemSelector": {
        "index.$": "$$.Map.Item.Index",
        "value.$": "$$.Map.Item.Value"
      }
    }
  }
```

```
}
```

2. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**DistributedMapDemo**」と入力します。

3. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

4. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 7: ステートマシンを実行する

実行とは、タスク実行のためにワークフローを実行するステートマシンのインスタンスのことです。

1. DistributedMapDemoページで [実行開始] を選択します。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しませ

ん。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。
3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

例えば、Map 状態を選択してから [マップ実行] を選択すると、[マップ実行の詳細] ページが開きます。このページでは、分散マップ状態と、その状態が開始した子ワークフロー実行に関するすべての実行の詳細を表示できます。このページの詳細については、「マップ実行の確認」を参照してください。

Lambda 関数でデータバッチ全体を処理する

このチュートリアルでは、分散マップ状態の ItemBatcher フィールドを使用して、Lambda 関数内の項目のバッチ全体を処理します。各バッチには、最大 3 つの項目が含まれます。分散マップ状態では、4 つの子ワークフロー実行が開始されます。それぞれの実行では 3 つの項目が処理され、1 つの実行では 1 つの項目が処理されます。子ワークフローを実行するたびに、バッチに存在する個々の項目を反復処理する Lambda 関数を呼び出します。

整数の配列を乗算するステートマシンを作成します。入力として指定した整数配列が `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` で、乗算係数が 7 であるとします。この場合、これらの整数に係数 7 を掛けた結果の配列は `[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]` になります。

トピック

- ステップ 1: ステートマシンを作成する
- ステップ 2 : Lambda 関数を作成する
- ステップ 3: ステートマシンを実行する

ステップ 1: ステートマシンを作成する

このステップでは、ステップ 2 で作成する Lambda 関数に、データのバッチ全体を渡すステートマシンのワークフロープロトタイプを作成します。

- 次の定義を使用して、Step Functions コンソールを使用してステートマシンを作成します。ステートマシンの作成については、チュートリアル「分散マップ状態の使用を開始する」の「ステップ 1: ワークフロープロトタイプを作成する」を参照してください。

  このステートマシンでは、入力として 10 個の整数の配列を入力として受け取り、この配列を 3 のバッチで Lambda 関数に渡す分散マップ状態を定義します。Lambda 関数は、バッチに存在する個々の項目を反復処理し、`multiplied` という名前の出力配列を返します。出力配列には、入力配列に渡された項目の乗算結果が含まれています。

  > **Important**
  >
  > 次のコードの Lambda 関数の Amazon リソースネーム (ARN) は、ステップ 2 で作成する関数の ARN に置き換えてください。

```
{
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "Map",
      "Result": {
        "MyMultiplicationFactor": 7,
        "MyItems": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      }
    },
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "Lambda Invoke",
        "States": {
          "Lambda Invoke": {
```

```
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
              "Payload.$": "$",
              "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:functionName"
            },
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException",
                  "Lambda.TooManyRequestsException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "End": true
          }
        }
      },
      "End": true,
      "Label": "Map",
      "MaxConcurrency": 1000,
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
        }
      },
      "ItemsPath": "$.MyItems"
    }
  }
}
```

ステップ 2 : Lambda 関数を作成する

このステップでは、バッチで渡されるすべての項目を処理する Lambda 関数を作成します。

> **Important**
>
> Lambda 関数がステートマシン AWS リージョン と同じ にあることを確認します。

Lambda 関数を作成するには

1. Lambda コンソールを使用して、**ProcessEntireBatch** という名前の Python 3.9 Lambda 関数を作成します。Lambda 関数の作成の詳細については、「分散マップ状態の使用開始」チュートリアルの「ステップ 4: Lambda 関数を設定する」を参照してください。 ???
2. Lambda 関数用の次のコードをコピーし、Lambda 関数の [コードソース] セクションに貼り付けます。

```
import json

def lambda_handler(event, context):
    multiplication_factor = event['BatchInput']['MyMultiplicationFactor']
    items = event['Items']

    results = [multiplication_factor * item for item in items]

    return {
        'statusCode': 200,
        'multiplied': results
    }
```

3. Lambda 関数を作成したら、ページの右上隅に表示されている関数の ARN をコピーします。ARN をコピーするには、 をクリックします。ARN の例を次に示します。ここで、*function-name* は Lambda 関数の名前 (この場合は ProcessEntireBatch) です。

```
arn:aws:lambda:us-east-1:123456789012:function:function-name
```

   ステップ 1 で作成したステートマシンに関数 ARN を指定する必要があります。
4. [デプロイ] を選択して、変更をデプロイします。

ステップ 3: ステートマシンを実行する

ステートマシンを実行すると、分散マップ状態では、4 つの子ワークフロー実行が開始されます。それぞれの実行では 3 つの項目が処理され、1 つの実行では 1 つの項目が処理されます。

次の例は、子ワークフロー実行の 1 つによって ProcessEntireBatch 関数に渡されるデータを示しています。

```
{
  "BatchInput": {
    "MyMultiplicationFactor": 7
  },
  "Items": [1, 2, 3]
}
```

この入力の場合、次の例では Lambda 関数によって返される multiplied という名前の付いた出力配列を示しています。

```
{
  "statusCode": 200,
  "multiplied": [7, 14, 21]
}
```

ステートマシンは、4 つの子ワークフロー実行に multiplied という名前の付いた 4 つの配列を含む、次の出力を返します。これらの配列には、個々の入力項目の乗算結果が含まれます。

```
[
  {
    "statusCode": 200,
    "multiplied": [7, 14, 21]
  },
  {
    "statusCode": 200,
    "multiplied": [28, 35, 42]
  },
  {
    "statusCode": 200,
    "multiplied": [49, 56, 63]
  },
  {
    "statusCode": 200,
    "multiplied": [70]
```

```
  }
]
```

返されたすべての配列項目を 1 つの出力配列にまとめるには、ResultSelector フィールドを使用できます。分散マップ状態内でこのフィールドを定義すると、すべての multiplied 配列を検索し、その配列内のすべての項目が抽出され、それらの項目を 1 つの出力配列に結合できます。

ResultSelector フィールドを使用するには、次の例に示すようにステートマシン定義を更新します。

```
{
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
      ...
      ...
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied[*]"
      }
    }
  }
}
```

更新されたステートマシンは、次の例に示すように統合された出力配列を返します。

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

Lambda 関数で個々のデータ項目を処理する

このチュートリアルでは、分散マップ状態の ItemBatcher フィールドを使用して、Lambda 関数を使用してバッチに存在する個々のアイテムを反復処理します。分散マップ状態は、4 つの子ワークフローの実行を開始します。これらの子ワークフローはそれぞれインラインマップステートを実行します。インラインマップステートでは、反復のたびに Lambda 関数を呼び出し、バッチから 1 つの項目を関数に渡します。次に、Lambda 関数は項目を処理し、結果を返します。

整数の配列を乗算するステートマシンを作成します。入力として指定した整数配列が [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] で、乗算係数が 7 であるとします。この場合、これらの整数に係数 7 を掛けた結果の配列は [7, 14, 21, 28, 35, 42, 49, 56, 63, 70] になります。

トピック

- ステップ 1: ステートマシンを作成する
- ステップ 2 : Lambda 関数を作成する
- ステップ 3: ステートマシンを実行する

ステップ 1: ステートマシンを作成する

このステップでは、ステップ 2 で作成する Lambda 関数のそれぞれの呼び出しに、項目のバッチから 1 つの項目を渡すステートマシンのワークフロープロトタイプを作成します。

- 次の定義を使用して、Step Functions コンソールを使用してステートマシンを作成します。ステートマシンの作成については、チュートリアル「分散マップ状態の使用を開始する」の「ステップ 1: ワークフロープロトタイプを作成する」を参照してください。

  このステートマシンでは、入力として 10 個の整数の配列を入力として受け取り、これらの配列項目をバッチで子ワークフロー実行に渡す分散マップ状態を定義します。子ワークフローを実行するたびに、入力として 3 つの項目のバッチが受け取り、インラインマップステートを実行します。インラインマップステートでは、反復のたびに Lambda 関数を呼び出し、バッチから 1 つの項目を関数に渡します。次に、この関数は項目に 7 の係数を掛けて、その結果を返します。

  子ワークフロー実行のそれぞれの出力は、渡された各項目の乗算結果を含む JSON 配列です。

  > **Important**
  >
  > 次のコードの Lambda 関数の Amazon リソースネーム (ARN) は、ステップ 2 で作成する関数の ARN に置き換えてください。

```
{
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
```

```
      "Next": "Map",
      "Result": {
        "MyMultiplicationFactor": 7,
        "MyItems": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      }
    },
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "InnerMap",
        "States": {
          "InnerMap": {
            "Type": "Map",
            "ItemProcessor": {
              "ProcessorConfig": {
                "Mode": "INLINE"
              },
              "StartAt": "Lambda Invoke",
              "States": {
                "Lambda Invoke": {
                  "Type": "Task",
                  "Resource": "arn:aws:states:::lambda:invoke",
                  "OutputPath": "$.Payload",
                  "Parameters": {
                    "Payload.$": "$",
                    "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:functionName"
                  },
                  "Retry": [
                    {
                      "ErrorEquals": [
                        "Lambda.ServiceException",
                        "Lambda.AWSLambdaException",
                        "Lambda.SdkClientException",
                        "Lambda.TooManyRequestsException"
                      ],
                      "IntervalSeconds": 2,
                      "MaxAttempts": 6,
                      "BackoffRate": 2
                    }
```

```
                ],
                "End": true
              }
            }
          },
          "End": true,
          "ItemsPath": "$.Items",
          "ItemSelector": {
            "MyMultiplicationFactor.$": "$.BatchInput.MyMultiplicationFactor",
            "MyItem.$": "$$.Map.Item.Value"
          }
        }
      }
    },
    "End": true,
    "Label": "Map",
    "MaxConcurrency": 1000,
    "ItemsPath": "$.MyItems",
    "ItemBatcher": {
      "MaxItemsPerBatch": 3,
      "BatchInput": {
        "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
      }
    }
  }
}
```

ステップ 2 : Lambda 関数を作成する

このステップでは、バッチから渡される各項目を処理する Lambda 関数を作成します。

> **Important**
>
> Lambda 関数がステートマシン AWS リージョン と同じ にあることを確認します。

Lambda 関数を作成するには

1. Lambda コンソールを使用して、**ProcessSingleItem** という名前の Python 3.9 Lambda 関数を作成します。Lambda 関数の作成の詳細については、「分散マップ状態の使用開始」チュートリアルの「ステップ 4: Lambda 関数を設定する」を参照してください。 ???
2. Lambda 関数用の次のコードをコピーし、Lambda 関数の [コードソース] セクションに貼り付けます。

```
import json

def lambda_handler(event, context):

    multiplication_factor = event['MyMultiplicationFactor']
    item = event['MyItem']

    result = multiplication_factor * item

    return {
        'statusCode': 200,
        'multiplied': result
    }
```

3. Lambda 関数を作成したら、ページの右上隅に表示されている関数の ARN をコピーします。ARN をコピーするには、

をクリックします。ARN の例を次に示します。ここで、*function-name* は Lambda 関数の名前 (この場合は ProcessSingleItem) です。

```
arn:aws:lambda:us-east-1:123456789012:function:function-name
```

   ステップ 1 で作成したステートマシンに関数 ARN を指定する必要があります。
4. [デプロイ] を選択して、変更をデプロイします。

ステップ 3: ステートマシンを実行する

ステートマシンを実行すると、分散マップ状態では、4 つの子ワークフロー実行が開始されます。それぞれの実行では 3 つの項目が処理され、1 つの実行では 1 つの項目が処理されます。

次の例は、子ワークフロー実行内の ProcessSingleItem 関数呼び出しの 1 つに渡されるデータを示しています。

```
{
  "MyMultiplicationFactor": 7,
  "MyItem": 1
}
```

この入力の場合、次の例では Lambda 関数によって返される出力を示しています。

```
{
  "statusCode": 200,
  "multiplied": 7
}
```

次の例は、子ワークフローを実行の 1 つに対する出力 JSON 配列を示しています。

```
[
  {
    "statusCode": 200,
    "multiplied": 7
  },
  {
    "statusCode": 200,
    "multiplied": 14
  },
  {
    "statusCode": 200,
    "multiplied": 21
  }
]
```

ステートマシンは、4 つの子ワークフロー実行のための 4 つの配列を含む、次の出力を返します。これらの配列には、個々の入力項目の乗算結果が含まれます。

最後に、ステートマシンの出力は、4 つの子ワークフロー実行に返された乗算結果をすべて組み合わせた、multiplied という名前の配列です。

```
[
  [
    {
      "statusCode": 200,
```

```
      "multiplied": 7
    },
    {
      "statusCode": 200,
      "multiplied": 14
    },
    {
      "statusCode": 200,
      "multiplied": 21
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 28
    },
    {
      "statusCode": 200,
      "multiplied": 35
    },
    {
      "statusCode": 200,
      "multiplied": 42
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 49
    },
    {
      "statusCode": 200,
      "multiplied": 56
    },
    {
      "statusCode": 200,
      "multiplied": 63
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 70
    }
```

```
  ]
]
```

子ワークフローの実行によって返されたすべての乗算結果を 1 つの出力配列にまとめるには、ResultSelector フィールドを使用できます。分散マップ状態内でこのフィールドを定義すると、すべての結果を検索し、それらを multiplied という名前の 1 つの出力配列に結合できます。

ResultSelector フィールドを使用するには、次の例に示すようにステートマシン定義を更新します。

```
{
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
      ...
      ...
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
        }
      },
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied"
      }
    }
  }
}
```

更新されたステートマシンは、次の例に示すように統合された出力配列を返します。

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

Amazon S3 イベント発生時にステートマシンの実行をスタートする

Amazon EventBridge ルールに対応する AWS Step Functions ステートマシンを実行できます。

このチュートリアルでは、Amazon EventBridge ルールのターゲットとして、ステートマシンを設定する方法を説明します。このルールは、Amazon Simple Storage Service (Amazon S3) バケットにファイルが追加されたときに、ステートマシンの実行を開始します。

実際のアプリケーションでは、バケットに追加したファイルに対してオペレーションを時刻するステートマシンを起動できます。イメージやビデオファイルに対するサムネイルの作成や Amazon Rekognition 分析の実行などです。

このチュートリアルでは、Amazon S3 バケットにファイルをアップロードすることにより、`Helloworld` ステートマシンの実行を開始します。次に、その実行の入力例を確認して、EventBridge に配信される Amazon S3 イベント通知からの入力に含まれる情報を確認します。

トピック

- 前提条件: ステートマシンを作成する
- ステップ 1: Amazon S3 バケットを作成する
- ステップ 2: EventBridge で Amazon S3 イベント通知を有効にする
- ステップ 3: Amazon EventBridge ルールを作成する
- ステップ 4 : ルールをテストする
- 実行入力の例

前提条件: ステートマシンを作成する

Amazon EventBridge ターゲットとしてステートマシンを構成する前に、ステートマシンを作成する必要があります。

- 基本的なステートマシンを作成する場合は、「Lambda 関数を使用するステートマシンの作成」のチュートリアルを使用してください。
- 既に `Helloworld` ステートマシンがある場合は、次のステップに進みます。

ステップ 1: Amazon S3 バケットを作成する

`Helloworld` ステートマシンを作成したら、次に Amazon S3 バケットを作成する必要があります。このチュートリアルのステップ 3 で、バケットにファイルがアップロードされると、EventBridge によってステートマシンの実行をトリガーするようにルールを設定します。

1. Amazon S3 コンソールに移動し、[バケットを作成] を選択してファイルを保存するバケットを作成し、Amazon S3 イベントルールをトリガーします。
2. [Bucket name] (バケット名) を入力します (例: *username*`-sfn-tutorial`)。

   > **Note**
   >
   > バケット名は必ず、Amazon S3 のすべての AWS リージョン内の既存バケット名の中で一意になるようにします。この名前を一意にするには、自分の*#######*を使用します。すべてのリソースは同じ AWS リージョン内に作成する必要があります。

3. ページのデフォルトの選択をすべて維持して、[バケットを作成] を選択します。

ステップ 2: EventBridge で Amazon S3 イベント通知を有効にする

Amazon S3 バケットを作成したら、S3 バケットで特定のイベント (ファイルのアップロードなど) が発生したときにイベントを EventBridge に送信するように設定します。

1. [Amazon S3 console] (Amazon S3 のコンソール) に移動します。
2. [Buckets (バケット)] リストで、イベントを有効にするバケットの名前を選択します。
3. [プロパティ] を選択します。
4. ページを下にスクロールして [イベント通知] セクションを表示し、[Amazon EventBridge] サブセクションで [編集] を選択します。
5. [このバケット内のすべてのイベント用の Amazon EventBridge に通知を送信する] の下にある [On] を選択します。
6. [Save changes] (変更の保存) をクリックします。

   > **Note**
   >
   > EventBridge を有効にすると、変更が適用されるまで約 5 分かかります。

ステップ 3: Amazon EventBridge ルールを作成する

ステートマシンの作成後に Amazon S3 バケットの追跡を作成したら、EventBridge ルールを作成します。

> **Note**
>
> Amazon S3 バケットと同じ AWS リージョンで EventBridge ルールを設定する必要があります。

ルールを作成するには

1. Amazon EventBridge コンソールに移動して、[Create rule] (ルールの作成) を選択します。

   > **Tip**
   >
   > または、EventBridge コンソールのナビゲーションペインで、[バス] の [ルール] を選択し、次に [ルールの作成] を選択します。

2. ルールの [名前] (例: *S3Step Functions*) を入力し、(オプション) ルールの [説明] を入力します。
3. [イベントバス] と [ルールタイプ] は、デフォルト設定のままにします。
4. [次へ] をクリックします。これにより、[イベントパターンを構築] ページが開きます。
5. [イベントパターン] セクションまでスクロールして、次の操作を行います。

   a. [イベントソース] で、デフォルトの [AWS イベントまたは EventBridge パートナーイベント] を選択したままにします。

   b. [AWS サービス] で、[Simple Storage Service (S3)] を選択します。

   c. [イベントタイプ] には、[Amazon S3 イベント通知] を選択します。

   d. [特定のイベント] を選択し、次に [オブジェクトの作成] を選択します。

   e. [名前別の特定のバケット] を選択し、ステップ 1 で作成したファイルを保存するバケット名 (*username*-sfn-tutorial) を入力します。

   f. [次へ] をクリックします。これにより、[ターゲットを選択] ページが開きます。

ターゲットを作成するには

1. [ターゲット 1] では、デフォルトの [AWS サービス] を選択したままにします。
2. [ターゲットを選択] ドロップダウンリストで、[Step Functions ステートマシン] を選択します。
3. [ステートマシン] リストで、以前に作成したステートマシン (例: `Helloworld`) を選択します。
4. ページのデフォルトの選択をすべて維持して、[次へ] を選択します。これにより、[タグを設定] ページが開きます。
5. [次へ] をもう一度選択します。これにより、[レビューして作成] ページが開きます。
6. ルールの詳細を確認し、[ルールの作成] を選択します。

   ルールが作成され、[Rules] (ルール) ページが表示されてすべての Amazon EventBridge ルールがリストされます。

ステップ 4 : ルールをテストする

すべて準備ができたので、Amazon S3 バケットへのファイルの追加をテストして、ステートマシンが実行された結果の入力を確認します。

1. Amazon S3 バケットにファイルを追加します。

   Amazon S3 コンソールに移動して、ファイルを保存するために作成したバケット (*username*-`sfn-tutorial`) を選択し、[アップロード] を選択します。
2. ファイル (例: *`test.png`*) を追加し、次に [アップロード] を選択します。

   これによりステートマシンの実行が起動され、AWS CloudTrail からの情報が入力として渡されます。
3. ステートマシンの実行をチェックします。

   Step Functions コンソールに移動して、Amazon EventBridge ルールで使用されているステートマシン (`Helloworld`) を選択します。
4. そのステートマシンの最新の実行を選択して、[実行の入力] セクションを展開します。

   この入力には、バケット名やオブジェクト名などの情報が含まれています。実際のユースケースでは、この入力を使用してステートマシンがそのオブジェクトに対してアクションを実行できます。

実行入力の例

次の例では、ステートマシン実行への一般的な入力を示しています。

```
{
    "version": "0",
    "id": "6c540ad4-0671-9974-6511-756fbd7771c3",
    "detail-type": "Object Created",
    "source": "aws.s3",
    "account": "123456789012",
    "time": "2023-06-23T23:45:48Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:s3:::username-sfn-tutorial"
    ],
    "detail": {
      "version": "0",
      "bucket": {
        "name": "username-sfn-tutorial"
      },
      "object": {
        "key": "test.png",
        "size": 800704,
        "etag": "f31d8546bb67845b4d3048cde533b937",
        "sequencer": "00621049BA9A8C712B"
      },
      "request-id": "79104EXAMPLEB723",
      "requester": "123456789012",
      "source-ip-address": "200.0.100.11",
      "reason": "PutObject"
    }
  }
```

API Gateway を使用した Step Functions API の作成

Amazon API Gateway を使用して、API Gateway AWS Step Functions APIsのメソッドに API を関連付けることができます。HTTPS リクエストが API メソッドに送信されると、API Gateway は Step Functions API アクションを呼び出します。

このチュートリアルでは、1 つのリソースと POST メソッドを使用して StartExecution API アクションとやり取りする API を作成する方法を示します。 AWS Identity and Access Management (IAM) コンソールを使用して、API Gateway のロールを作成します。次に、API Gateway コン

ソールを使用して API Gateway API を作成し、リソースとメソッドを作成して、メソッドを `StartExecution` API アクションにマッピングします。最後に、API をデプロイしてテストします。

> **Note**
>
> `StartExecution` を呼び出して Amazon API Gateway は Step Functions 実行をスタートできますが、結果を取得するには `DescribeExecution` を呼び出す必要があります。

トピック

- ステップ 1: API Gateway 用にIAM ロールを作成する
- ステップ 2: API Gateway で API を作成する
- ステップ 3: API Gateway API のテストとデプロイ

ステップ 1: API Gateway 用にIAM ロールを作成する

API Gateway API を作成する前に、API Gateway 許可を与えて Step Functions API アクションを呼び出す必要があります。

API Gateway の許可を設定するには

1. IAM コンソールにサインインして、[Roles] (ロール)、[Create role] (ロールの作成) の順に選択します。
2. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. [信頼できるエンティティタイプ] は、デフォルトの AWS のサービス の選択のままにします。
   b. [ユースケース] には、ドロップダウンリストから、[API Gateway] を選択します。
3. [API Gateway] を選択し、[次へ] をクリックします。
4. [アクセス許可を追加] ページで [次へ] を選択します。
5. (オプション) [名前、確認、および作成] ページで、ロール名などの詳細を入力します。たとえば、**`APIGatewayToStepFunctions`** と入力します。
6. [ロールの作成] を選択します。

   ロールのリストで、IAM ロールが表示されます。

7. 次の例に示すように、お客様のロールの名前を選択し、[Role ARN] (ロール ARN) をメモします。

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

IAM ロールにポリシーをアタッチするには

1. [Roles] (ロール) ページで、ロール (APIGatewayToStepFunctions) を検索して選択します。
2. [アクセス許可] タブで、[アクセス許可の追加] を選択してから、[ポリシーのアタッチ] を選択します。
3. [ポリシーのアタッチ] ページで AWSStepFunctionsFullAccess を検索し、ポリシーを選択して、次に [アクセス許可の追加] を選択します。

ステップ 2: API Gateway で API を作成する

IAM ロールを作成した後、カスタムの API Gateway API を作成できます。

API を作成するには

1. Amazon API Gateway コンソールを開き、[API を作成] を選択します。
2. [API タイプを選択] ページの [REST API] ペインで、[ビルド] を選択します。
3. REST API の作成ページで、新しい API を選択し、***StartExecutionAPI*** 名 に API を入力します。
4. [API エンドポイントタイプ] は [リージョン] のままにし、[API の作成] を選択します。

リソースを作成するには

1. ***StartExecutionAPI*** のリソースページで、リソースの作成を選択します。
2. [リソースの作成] ページで、[リソース名] に **execution** と入力し、[リソースを作成] を選択します。

POST メソッドを作成するには

1. [/execution] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、POST を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. AWS リージョン については、リストからリージョンを選択します。

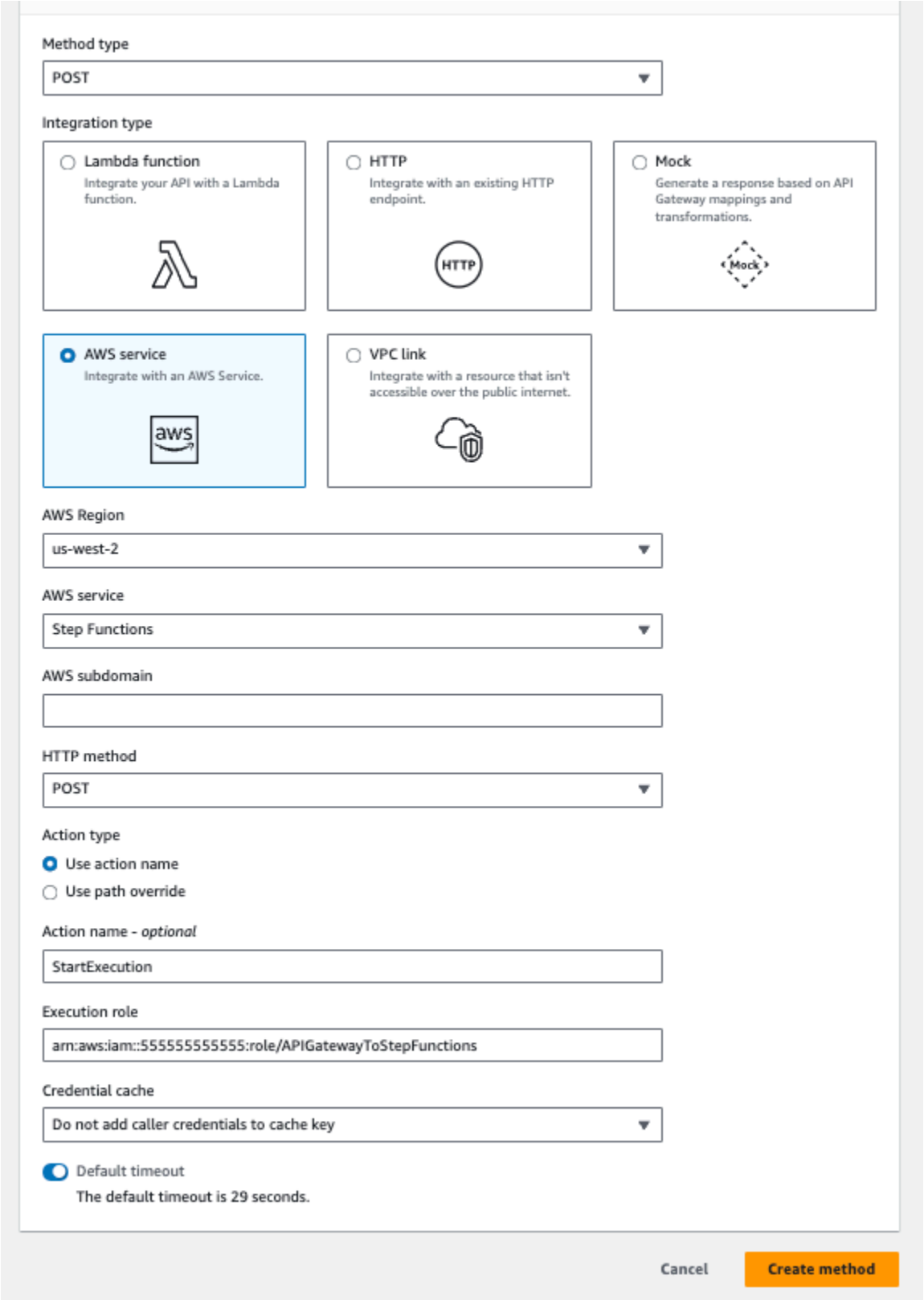
> **Note**
>
> 現在 Step Functions をサポートしているリージョンについては、[Supported Regions] (サポートされているリージョン) を参照してください。

5. AWS のサービス の場合、リストから [Step Functions] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] でリストから [POST] を選択します。

> **Note**
>
> すべての Step Functions API アクションは HTTP POST メソッドを使用します。

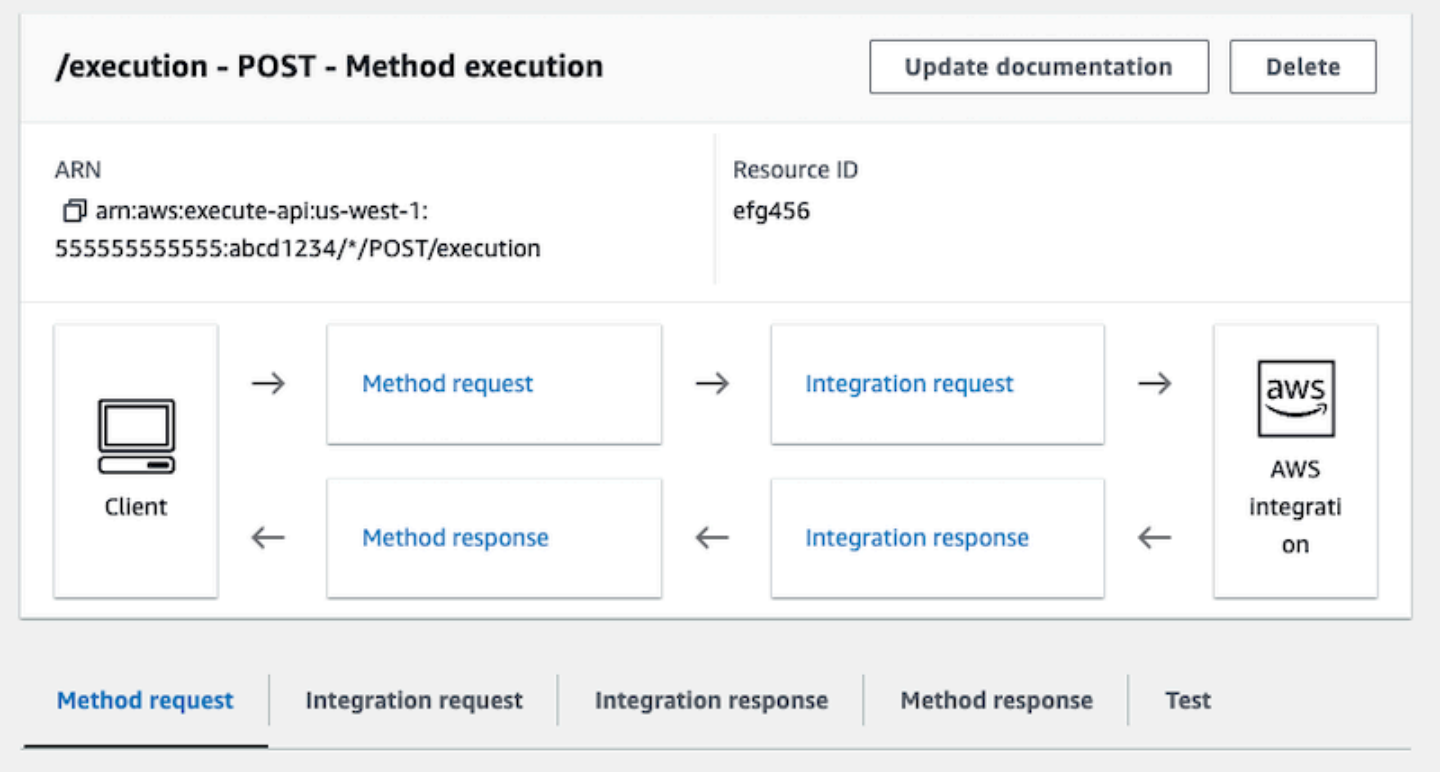
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**StartExecution**」と入力します。
10. 次の例に示すように、[実行ロール] の場合は、前に作成した IAM ロールのロール ARN を入力します。

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


Method type
POST
Integration type
Lambda function
Integrate your API with a Lambda function.
HTTP
Integrate with an existing HTTP endpoint.
HTTP
Mock
Generate a response based on API Gateway mappings and transformations.
Mock
AWS service
Integrate with an AWS Service.
aws
VPC link
Integrate with a resource that isn't accessible over the public internet.
AWS Region
us-west-2
AWS service
Step Functions
AWS subdomain
HTTP method
POST
Action type
Use action name
Use path override
Action name - optional
StartExecution
Execution role
arn:aws:iam::555555555555:role/APIGatewayToStepFunctions
Credential cache
Do not add caller credentials to cache key
Default timeout
The default timeout is 29 seconds.
Cancel
Create method

11. [認証情報キャッシュ] と [デフォルトタイムアウト] はデフォルトのままにし、[保存] を選択します。

API ゲートウェイおよび Step Functions 間の視覚的なマッピングは [/execution - POST - メソッドの実行] ページに表示されます。



ステップ 3: API Gateway API のテストとデプロイ

API を作成したら、テストしてデプロイします。

API Gateway と Step Functions 間の通信をテストするには

1. [/execution - POST - メソッドの実行] ページで、[テスト] を選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [/execution - POST - メソッドのテスト] タブで、次のリクエストパラメータを、既存のステートマシン (または Lambda 関数を使用する新しいステートマシンを作成) の ARN を使用して [リクエスト本文] セクションにコピーし、[テスト] を選択します。

```
{
   "input": "{}",
   "name": "MyExecution",
```

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

詳細については、[AWS Step Functions API Reference] (API リファレンス) の`StartExecution`リクエストの構文を参照してください。

> **Note**
>
> ステートマシンの ARN を API Gateway 呼び出しの本文に含めない場合は、次の例に示すように、[統合リクエスト] タブでマッピングテンプレートを設定できます。
>
> ```
> {
>     "input": "$util.escapeJavaScript($input.json('$'))",
>     "stateMachineArn": "$util.escapeJavaScript($stageVariables.arn)"
> }
> ```
>
> このアプローチにより、開発ステージ (例えば、`dev`、`test`、`prod`) に基づいて異なるステートマシンの ARN を指定できます。マッピングテンプレートのステージ変数指定の詳細については、「API Gateway デベロッパーガイド」の「`$stageVariables`」を参照してください。

3. 実行がスタートされ、実行 ARN とその工ポック日が [レスポンス本文] の下に表示されます。

```
{
   "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution",
   "startDate": 1486768956.878
}
```

> **Note**
>
> AWS Step Functions コンソールでステートマシンを選択することで、実行を確認できます。

API をデプロイするには

1. ***StartExecutionAPI*** のリソースページで、API をデプロイを選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **alpha** と入力します。
4. (オプション) [説明] に説明を入力します。
5. デプロイを選択します。

デプロイをテストするには

1. ***StartExecutionAPI*** のステージページで、アルファ 、/、/execution 、POST を展開し、POST メソッドを選択します。
2. [メソッドの上書き] で、コピーアイコンを選択して API の呼び出し URL をコピーします。完全な URL は、次の例のようになります。

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

3. コマンドラインから、ステートマシンの ARN を使用して `curl` コマンドを実行し、次の例に示すようにデプロイの URL を呼び出します。

```
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
```

次の例に示すように、実行 ARN とそのエポック日付が返されます。

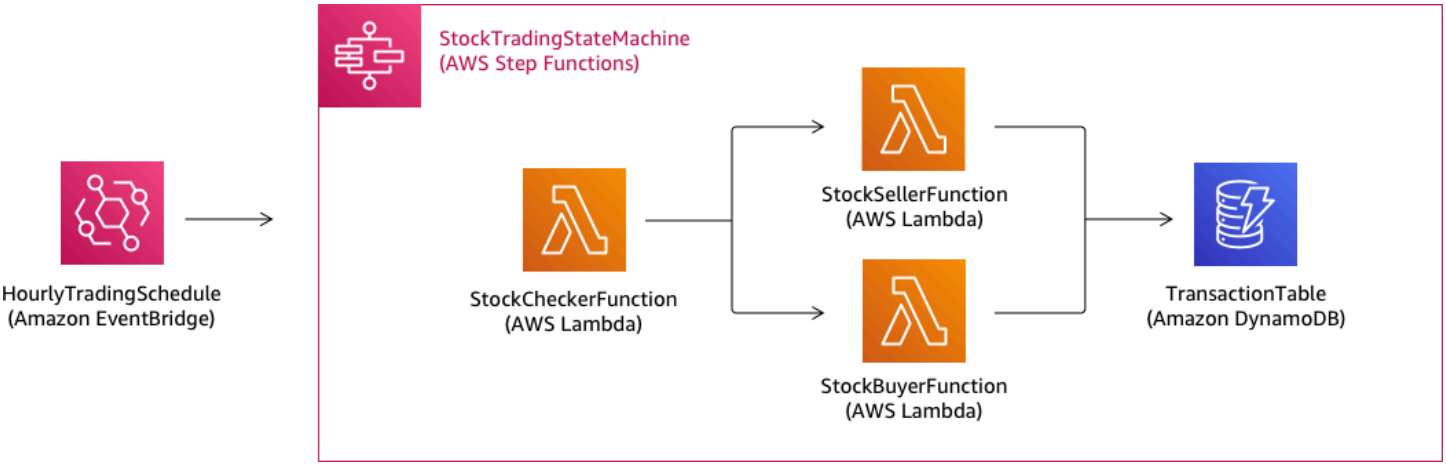
```
{"executionArn":"arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution","startDate":1.486772644911E9}
```

> **Note**
>
> 「認証トークンが見つかりません」というエラーが表示された場合は、呼び出し URL が /execution で終わっていることを確認してください。

AWS SAM を使用して Step Functions ステートマシンを作成する

このガイドでは、AWS Step Functions ステートマシンを含むサンプル AWS SAM アプリケーションをダウンロード、ビルド、デプロイします。このアプリケーションは、事前定義されたスケジュールで実行されるモック株式取引ワークフローを作成します (スケジュールは手数料の発生を避けるためにデフォルトで無効になっています)。

以下の図は、このアプリケーションのコンポーネントを示しています。



以下は、サンプルアプリケーションを作成するために実行するコマンドのプレビューです。各コマンドの詳細については、このページの後半のセクションを参照してください。

```
# Step 1 - Download a sample application. For this tutorial you
#   will follow the prompts to select an AWS Quick Start Template
#   called 'Multi-step workflow'
sam init

# Step 2 - Build your application
cd project-directory
sam build

# Step 3 - Deploy your application
sam deploy --guided
```

前提条件

このガイドは、ご使用の OS で AWS SAM CLI のインストールのステップを完了していることを前提としています。また、以下を完了していることを前提としています。

1. AWS アカウントを作成します。

2. IAM 許可を設定する
3. Homebrew のインストール。注: Homebrew は、Linux と macOS の唯一の前提条件です。
4. AWS SAM CLI のインストール。注: バージョン 0.52.0 以降を使用していることを確認してください。コマンド `sam --version` を実行すると、使用しているバージョンをチェックできます。

ステップ 1: サンプル AWS SAM アプリケーションをダウンロードする

実行するコマンド:

```
sam init
```

画面の指示に従って、次の項目を選択します。

1. [Template:] (テンプレート:) AWS クイックスタートテンプレート
2. [Language:] (言語:) Python、Ruby、NodeJS、Go、Java、または .NET
3. [Project name:] (プロジェクト名:) (任意の名前 - デフォルトは `sam-app`)
4. クイックスタートアプリケーション: マルチステップワークフロー

AWS SAM が実行する事柄:

このコマンドにより、プロジェクト名プロンプトとして指定した名前のディレクトリが作成されます (デフォルトは `sam-app`)。ディレクトリの特定の内容は、選択した言語によって異なります。

Python ランタイムの 1 つを選択したときのディレクトリの内容は次のとおりです。

```
### README.md
### functions
#   ### __init__.py
#   ### stock_buyer
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### stock_checker
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### stock_seller
#       ### __init__.py
#       ### app.py
```

```
#       ### requirements.txt
### statemachine
#   ### stock_trader.asl.json
### template.yaml
### tests
    ### unit
        ### __init__.py
        ### test_buyer.py
        ### test_checker.py
        ### test_seller.py
```

表示できる 2 つの特に興味深いファイルがあります:

- AWS: アプリケーションの AWS SAM リソースを定義する `template.yaml` テンプレートが含まれます。
- `statemachine/stockTrader.asl.json`: アプリケーションのステートマシン定義が含まれます。この定義は、Amazon ステートメント言語 で記述されています。

`template.yaml` ファイル内に次のエントリが表示されます。このエントリは、ステートマシン定義ファイルを指しています。

```
    Properties:
      DefinitionUri: statemachine/stock_trader.asl.json
```

ステートマシン定義を AWS SAM テンプレートに埋め込むのではなく、別ファイルとして保存しておくと便利です。例えば、ステートマシン定義の変更を追跡する場合は、定義をテンプレートに含めない方が簡単です。Workflow Studio を使用してステートマシン定義の作成と管理を行い、その定義をテンプレートにマージせずにコンソールから Amazon States Language 仕様ファイルに直接エクスポートできます。

サンプルアプリケーションの詳細については、プロジェクトディレクトリの `README.md` ファイルを参照してください。

ステップ 2: アプリケーションを構築する

実行するコマンド:

まず、プロジェクトディレクトリ (サンプルアプリケーションの `template.yaml` ファイルが置かれているディレクトリ。デフォルトは `sam-app` です) に変更してから、次のコマンドを実行します。

```
sam build
```

出力例:

```
Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Invoke Function: sam local invoke
[*] Deploy: sam deploy --guided
```

AWS SAM が実行する事柄:

AWS SAM CLI には、依存関係を構築するための Lambda ランタイムの抽象化が多数含まれており、すべてのビルドアーティファクトをステージングフォルダにコピーして、すべてをパッケージ化してデプロイできるようにします。`sam build` コマンドは、アプリケーションが持つすべての依存関係を構築し、`.aws-sam/build` のフォルダにビルドアーティファクトをコピーします。

ステップ 3: アプリケーションを AWS クラウドにデプロイする

実行するコマンド:

```
sam deploy --guided
```

画面に表示されるプロンプトに従ってください。`Enter` で応答するだけで、インタラクティブな形式で提供されるデフォルトのオプションを受け入れることができます。

AWS SAM が実行する事柄:

このコマンドは、アプリケーションを AWS クラウドにデプロイします。`sam build` コマンドで構築したデプロイアーティファクトを取得し、パッケージ化して、AWS SAM CLI で作成された Amazon S3 バケットにアップロードして AWS CloudFormation を使用してアプリケーションをデプロイします。デプロイコマンドの出力では、AWS CloudFormation スタックに加えられた変更を確認できます。

Step Functions ステートマシンの例が、次のこれらのステップに従って、正常にデプロイされたことを確認できます。

1. AWS Management Console にサインインし、https://console.aws.amazon.com/states/ で、Step Functions コンソールを開きます。
2. 左側のナビゲーションで、[State machines] (ステートマシン) を選択します。
3. リストから新しいステートマシンを見つけて選択します。これは、StockTradingStateMachine-*<unique-hash>* という名前になります。
4. [Definition] (定義) タブを選択します。

これで、ステートマシンのビジュアル表現が表示されます。ビジュアル表現が、プロジェクトディレクトリの `statemachine/stockTrader.asl.json` ファイルにあるステートマシン定義と一致していることを確認できます。

トラブルシューティング

SAM CLI エラー: "no such option: --guided"

`sam deploy` の実行時に、以下のエラーが表示されます。

```
Error: no such option: --guided
```

これは、`--guided` パラメータをサポートしていない古いバージョンの AWS SAM CLI を使用していることを意味します。これを修正するには、AWS SAM CLI のバージョンを 0.33.0 以降に更新するか、`--guided` コマンドから `sam deploy` パラメータを削除します。

SAM CLI エラー: 「管理対象リソースを作成できませんでした: 認証情報を見つけることができません」

`sam deploy` の実行時に、以下のエラーが表示されます。

```
Error: Failed to create managed resources: Unable to locate credentials
```

これは、AWS 認証情報をセットアップして、AWS SAM CLI が AWS のサービスの呼び出せるようにしていないことを意味します。これを修正するには、AWS 認証情報をセットアップする必要があ

ります。詳細については、AWS Serverless Application Model デベロッパーガイドのコンソールを使用した AWS の設定を参照してください。

クリーンアップ

このチュートリアルを実行して作成した AWS リソースが不要になった場合は、デプロイした AWS CloudFormation スタックを削除することでそれらを削除できます。

AWS Management Console を使用してこのチュートリアルで作成した AWS CloudFormation スタックを削除するには、以下のステップを実行します。

1. AWS Management Console にサインインし、AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. 左のナビゲーションペインで [Stacks] (スタック) をクリックします。
3. スタックのリストで、[sam-app] (または作成したスタックの名前) を選択します。
4. [Delete] (削除) を選択します。

完了すると、スタックのステータスが DELETE_COMPLETE に変わります。

または、次の AWS CLI コマンドを実行して AWS CloudFormation スタックを削除することもできます。

```
aws cloudformation delete-stack --stack-name sam-app --region region
```

削除されたスタックを確認する

どちらの方法でも、AWS CloudFormation スタックが削除されていることを確認するには、https://console.aws.amazon.com/cloudformation にアクセスして、左側のナビゲーションペインで [Stacks] (スタック) を選択し、検索テキストボックスの右側のドロップダウンで [Deleted] (削除済み) を選択します。削除されたスタックのリストに、スタック名 [sam-app] (または作成したスタックの名前) が表示されます。

Step Functions を使用してアクティビティステートマシンを作成する

このチュートリアルでは、Java と AWS Step Functionsを使用してアクティビティベースのステートマシンを作成する方法について説明します。アクティビティにより、ステートマシンの別の場所で実

行されるワーカーコードを制御できます。アクティビティ の概要については、Step Functions 仕組み を参照してください。

このチュートリアルを完了するには、以下が必要です。

- SDK for Java。このチュートリアルのアクティビティ例は、を使用してと通信する AWS SDK for Java Java AWSアプリケーションです。
- AWS AWS 環境内または標準設定ファイル内の認証情報。詳細については、『AWS SDK for Java 開発者ガイド』の「 AWS 認証情報の設定」を参照してください。

トピック

- ステップ 1: アクティビティを作成する
- ステップ 2: ステートマシンを作成する
- ステップ 3: ワーカーを実装する
- ステップ 4: ステートマシンを実行する
- ステップ 5: ワーカーを実行して停止する

ステップ 1: アクティビティを作成する

Step Functions に、作成したい[worker] (ワーカー) (プログラム) の [activity] (アクティビティ) を認識させる必要があります。Step Functions は、アクティビティのアイデンティティを確立する Amazon リソースネーム (ARN) で応答します。このアイデンティティを使用して、ステートマシンとワーカー間で渡される情報を調整します。

> **Important**
>
> AWS アクティビティタスクがステートマシンと同じアカウントにあることを確認してください。

1. Step Functions コンソール内、左のナビゲーションペインで、[Activities] (アクティビティ) を選択します。
2. [Create activity] (アクティビティの作成) を選択します。
3. アクティビティの [名前] (*get-greeting* など) を入力し、[アクティビティの作成] を選択します。

4. 次の例に示すように、アクティビティタスクが作成されたら、その ARN を記録します。

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

ステップ 2: ステートマシンを作成する

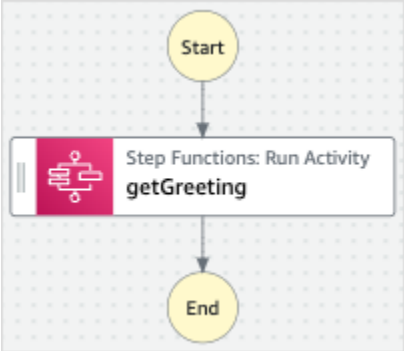
アクティビティがいつ呼び出されるかと、ワーカーがいつプライマリ作業を実行して、その結果を収集し、結果を戻すべきかを判断するステートマシンを作成します。ステートマシンを作成するには、Workflow Studio の コードエディタ を使用します。

1. Step Functions コンソール内、左のナビゲーションペインで、[State machines] (ステートマシン) を選択します。
2. [ステートマシン] ページで、[ステートマシンの作成] を選択します。
3. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
4. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
5. このチュートリアルでは、コードエディタでステートマシンの Amazon ステートメント言語 (ASL) 定義を記述します。これを行うには、[コード] を選択します。
6. 既存のボイラープレートコードを削除して、次のコードを貼り付けます。このコード内の ARN の例を、`Resource` フィールドで以前に作成したアクティビティタスクの ARN に置き換えることを留意してください。

```
{
  "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
    }
  }
}
```

これは、Amazon ステートメント言語 (ASL) を使用したステートマシンの説明です。`getGreeting` という名前の単一の `Task` 状態を定義します。詳細については、「State Machine Structure」を参照してください。

7. グラフ可視化 で、追加した ASL 定義のワークフローグラフが次のグラフのようになっていることを確認します。



8. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択しますMyStateMachine。次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**`ActivityStateMachine`**」と入力します。

9. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

   ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

10. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

    [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパ

ルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 3: ワーカーを実装する

ワーカーを作成します。ワーカーは、次の処理を行うプログラムです。

- `GetActivityTask` API アクションを使用したアクティビティの Step Functions のポーリング。
- コードを使用したアクティビティの作業の実行 (次のコードの `getGreeting( )` メソッドなど)。
- `SendTaskSuccess`、`SendTaskFailure`、`SendTaskHeartbeat` API アクションを使用して結果を返す。

> **Note**
>
> アクティビティワーカーの詳細な例については、Ruby のサンプルアクティビティワーカーを参照してください この例により、アクティビティワーカーのリファレンスとして使用できるベストプラクティスに基づく実装が提供されます。このコードには、ポーラーおよびアクティビティワーカー用に構成可能な数のスレッドを含む、コンシューマー/プロデューサーパターンが実装されています。

ワーカーを実装するには

1. `GreeterActivities.java` という名前のファイルを作成します。
2. 次のコードを追加します。

```
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**
>
> この例の EnvironmentVariableCredentialsProvider クラスでは、AWS_ACCESS_KEY_ID (または AWS_ACCESS_KEY) および AWS_SECRET_KEY (または AWS_SECRET_ACCESS_KEY) 環境変数が設定されていることを前提としています。必要な認証情報をファクトリに提供する方法の詳細については、『AWS SDK for Java 開発者ガイド』の「AWS SDK for Java API リファレンス」と「AWS 開発用の認証情報とリージョンの設定」のを参照してくださいAWSCredentialsProvider。
> デフォルトでは、 AWS SDK はどの操作でもサーバーからデータを受信するまで最大 50 秒待機します。GetActivityTask オペレーションは、次に使用可能なタスクまで最大 60 秒待機するロングポーリングオペレーションです。SocketTimeoutException SocketTimeout エラーを受信しないようにするには、70 秒に設定します。

3. GetActivityTaskRequest().withActivityArn() コンストラクタのパラメータリストで、ACTIVITY_ARN 値を、以前に作成したアクティビティタスクの ARN に置き換えます。

ステップ 4: ステートマシンを実行する

ステートマシンの実行をスタートすると、ワーカーがアクティビティの Step Functions をポーリングして、その作業を実行し (指定した入力を使用)、その結果を返します。

1. ***ActivityStateMachine***ページで [実行開始] を選択します。

   [実行を開始] ダイアログが表示されます。

2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

b. [入力] ボックスに、次の JSON 入力を入力してワークフローを実行します。

```
{
  "who": "AWS Step Functions"
}
```

c. [実行のスタート] を選択します。

d. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステップ 5: ワーカーを実行して停止する

ワーカーにアクティビティのステートマシンをポーリングさせるには、ワーカーを実行する必要があります。

1. コマンドラインで、`GreeterActivities.java` を作成したディレクトリに移動します。
2. AWS SDK を使用するには、`libthird-party`およびディレクトリのフルパスをビルドファイルと Java の依存関係に追加します。CLASSPATH詳細については、[AWS SDK for Java Developer Guide] (デベロッパーガイド) の SDK をダウンロードして抽出するを参照してください。
3. ファイルをコンパイルします。

```
$ javac GreeterActivities.java
```

4. ファイルを実行します。

```
$ java GreeterActivities
```

5. Step Functions コンソールで、[実行の詳細] ページに移動します。
6. 実行が完了したら、実行結果を確認します。
7. ワーカーを停止します。

Lambda を使用してループを反復処理する

このチュートリアルでは、ステートマシンと AWS Lambda 関数を使用してループを一定回数反復する設計パターンを実装します。

必要に応じて、この設計パターンを随時使用し、ステートマシンのループ数を追跡できます。この実装を使用すると、大規模なタスクや時間がかかる実行を小さいチャンクに分割したり、特定のイベント数の後で実行を終了させたりできます。同様の実装を使用して、長時間実行される実行を定期的に終了して再開し、、 AWS Step Functions、 AWS Lambdaまたは他のサービスのサービスクォータを超えないようにすることができます。 AWS

開始する前に Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルを参照して、Lambda と Step Functions の使用方法に慣れておいてください。

ステップ 1: Lambda 関数を作成してカウントを反復する

Lambda 関数を使用することで、ステートマシンでのループの反復回数を追跡できます。次の Lambda 関数は、`count`、`index`、`step` の入力値を受け取ります。次に、更新された `index` と、`continue` というブール値を持つこれらの値を返します。`index` が `count` 未満の場合、Lambda 関数は `continue` を `true` に設定します。

その後、ステートマシンは、`continue` が `true` の場合に `Choice` 状態を実装し、`false` の場合に終了します。

Lambda 関数を作成するには

1. [Lambda console] (Lambda コンソール) にサインインし、[Create function] (関数の作成) を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。
3. [ベーシック情報] セクションで、以下のように Lambda 関数を設定します。

a. [Function name] (関数名) に `Iterator` と入力します。

b. [ランタイム] で、[Node.js] を選択します。

c. [デフォルトの実行ロールの変更] で、[基本的な Lambda アクセス許可で新しいロールを作成] を選択します。

d. [関数を作成] を選択します。

4. Lambda 関数の次のコードをコードソースにコピーします。

```
export const handler = function (event, context, callback) {
  let index = event.iterator.index
  let step = event.iterator.step
  let count = event.iterator.count

  index = index + step

  callback(null, {
    index,
    step,
    count,
    continue: index < count
  })
}
```

このコードは、`count`、`index`、`step` の入力値を受け入れています。`index` を `step` の値増分し、それらの値とブール値 `continue` を返します。`index` が `count` 未満の場合、`continue` の値は `true` です。

5. デプロイを選択します。

ステップ 2: Lambda 関数をテストする

数値を使用して Lambda 関数を実行し、関数のオペレーションを確認します。Lambda 関数には、反復を模倣した入力値を指定できます。

Lambda 関数をテストします。

1. [テスト] を選択します。

2. [テストイベントの設定] ダイアログボックスで、[イベント名] ボックスに `TestIterator` を入力します。

3. データ例を以下に置き換えます。

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

これらの値は、反復でステートマシンから返される値を表します。Lambda 関数はインデックスをインクリメントし、truecontinueインデックスがそれより小さくなると戻ります。countこのテストでは、インデックスは既に 5 に増分されています。テストはにインクリメントされ、に設定されますindex。6 continue true

4. [作成] を選択します。
5. [テスト] を選択して Lambda 関数をテストします。

   テストの結果は [実行結果] タブに表示されます。
6. 実行結果を表示するには、[出力] タブを選択します。

```
{
  "index": 6,
  "step": 1,
  "count": 10,
  "continue": true
}
```

> **Note**
>
> index9に設定して再度テストすると、indexはにインクリメントされ10continue、になります。false

ステップ 3: ステートマシンを作成する

> **Lambda コンソールを終了する前に...**
>
> Lambda 関数 ARN をコピーします。これをメモに貼り付けます。これは次のステップで必要になります。

次に、以下のステートを持つステートマシンを作成します。

- `ConfigureCount`— `count`、`index`、のデフォルト値を設定しますstep。
- `Iterator`— 以前に作成した Lambda 関数を参照し、で設定した値を渡します。`ConfigureCount`
- `IsCountReached`— 関数から返された値に基づいて、ループを継続するか、`Done`ステートに進む選択ステート。`Iterator`
- `ExampleWork`— 実行する必要のある作業のスタブ。この例では、`Pass`ワークフローには状態がありますが、実際のソリューションではを使用するのが普通です`Task`。
- `Done`— ワークフローの終了状態。

コンソールでステートマシンを作成するには:

1. Step Functions コンソールを開き、[Create a state machine] (ステートマシンの作成) を選択します。

   > **Important**
   >
   > ステートマシンは Lambda AWS 関数と同じアカウントとリージョンにある必要があります。

2. 空白のテンプレートを選択します。
3. コードペインに、ステートマシンを定義する次の JSON を貼り付けます。

   Amazon ステートメント言語の詳細については、[State Machine Structure] (ステートマシンの構造) を参照してください。

```
{
    "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

        }
```

```
    }
}
```

4. `Iterator Resource`フィールドを、前に作成した `Iterator` Lambda 関数の ARN に置き換えます。
5. Config を選択し、ステートマシンの名前 (など) を入力します*`IterateCount`*。

> **Note**
>
> ステートマシン、実行、およびアクティビティタスク名は 80 文字以下にする必要があります。 AWS これらの名前はアカウントと地域で一意である必要があり、次のものを含んではいけません。
>
> - 空白
> - ワイルドカード文字 (? *)
> - 角かっこ (< > { } [ ])
> - 特殊文字 (: ; , \ | ^ ~ $ # % & ` ")
> - 制御文字 (\\u0000 - \\u001f または \\u007f - \\u009f)
>
> ステートマシンのタイプが [Express] の場合、ステートマシンを複数回実行する際に同じ名前を指定できます。Step Functions では、複数の実行が同じ名前であっても、Express ステートマシンの実行ごとに一意の実行 ARN を生成します。
> Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

6. 「タイプ」には、「標準」のデフォルト値をそのまま使用します。[権限] で [新しいロールを作成] を選択します。
7. [作成] を選択し、ロールの作成を確認します。

ステップ 4: 新しい実行をスタートする

ステートマシンを作成した後、実行をスタートできます。

1. IterateCountページで [実行開始] を選択します。

2. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

3. [実行のスタート] を選択します。

   ステートマシンの新しい実行がスタートされ、進行中の実行が表示されます。
   ステートマシングラフビュー。Iterator の状態は青色で表示され、進行中であることを示します。

   実行は、Lambda 関数を使用してカウントを追跡しながら段階的に増分されます。反復ごとに、ステートマシンの `ExampleWork` 状態で参照されている作業例が実行されます。

   カウントがステートマシンの `ConfigureCount` 状態で設定した数に到達すると、実行が反復を完了して終了します。
   ステートマシングラフビュー。Iterator 状態と Done 状態は緑色で表示され、どちらも成功していることを示します。

長時間実行されているワークフロー実行を新しい実行として継続する

AWS Step Functions は一定の期間とステップ数のあるワークフローを実行するように設計されています。実行の期間は最大 1 年間で、そのイベント数は最大 25,000 です (クォータ を参照してください)。

実行時間が長い場合、実行イベント履歴のハードクォータの 25,000 エントリに達しないように、ステートマシンの `Task` 状態から直接新しいワークフロー実行を開始することを推奨します。これにより、ワークフローをより小さなステートマシンに分割し、進行中の作業を新しい実行で継続できます。これらのワークフロー実行を開始するには、`Task` 状態から `StartExecution` API アクションを呼び出し、必要なパラメータを渡します。

または、Lambda 関数を使用してステートマシンの新しい実行を開始することにより、進行中の作業を複数のワークフロー実行に分割するパターンを実装できます。

このチュートリアルでは、Service Quotas を超えることなくワークフローの実行を継続するための両方の方法を紹介します。

トピック

- Step Functions API アクションを使用して新しい実行を継続する (推奨)
- Lambda 関数を使用して新しい実行を継続する

Step Functions API アクションを使用して新しい実行を継続する (推奨)

Step Functions は、独自の API を統合サービスとして呼び出して、ワークフロー実行を開始できます。実行時間がない場合の Service Quotas の超過を避けるために、この方法を使用することを推奨します。

ステップ 1: 長時間実行するステートマシンを作成する

長時間実行するステートマシンを作成して、別のステートマシンの Task 状態から開始できるようにします。このチュートリアルでは、Lambda 関数を使用するステートマシンを使用します。

> **Note**
>
> このステートマシンの名前と Amazon リソースネームは、後で使用できるようにテキストファイルにコピーしておいてください。

ステップ 2: Step Functions API アクションを呼び出すステートマシンを作成する

Task 状態からワークフロー実行を開始する

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. 「アクション」タブから StartExecutionAPI アクションをドラッグし、「最初の状態をここにドラッグ」というラベルの付いた空のステートにドロップします。
5. StartExecution状態を選択し、の [設定] デザインモード タブで次の操作を行います。

a. 状態の名前を **`Start nested execution`** に変更します。

b. [統合タイプ] には、ドロップダウンリストから [AWS SDK - 新規] を選択します。

c. [API パラメータ] で、次の操作を行います。

i. `StateMachineArn` は、サンプルの Amazon リソースネームをステートマシンの ARN に置き換えます。例えば、Lambda を使用するステートマシン の ARN を入力します。

ii. `Input` ノードで、次のように、既存のプレースホルダーテキストを置き換えます。

```
"Comment": "Starting workflow execution using a Step Functions API action"
```

iii. [API パラメータ] の入力は、次のようになっていることを確認してください。

```
{
  "StateMachineArn": "arn:aws:states:us-
east-2:123456789012:stateMachine:LambdaStateMachine",
  "Input": {
    "Comment": "Starting workflow execution using a Step Functions API
 action",
    "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
  }
```

6. (オプション) Inspector パネルの [定義] を選択すると、ワークフローの自動生成された Amazon ステートメント言語 (ASL) の定義が表示されます。

> **Tip**
>
> また、Workflow Studio の コードエディタ で ASL の定義を表示できます。コードエディタで、ワークフローの ASL の定義を編集することもできます。

7. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択しますMyStateMachine。次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**`ParentStateMachine`**」と入力します。

8. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

9. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 3: IAM ポリシーを変更する

ステートマシンに Lambda 関数を使用するステートマシンの実行を開始するアクセス許可があることを確認するには、ステートマシンの IAM ロールにインラインポリシーをアタッチする必要があります。詳細については、「IAM ユーザーガイド」の「インラインポリシーの埋め込み」を参照してください。

1. ParentStateMachineページで IAM ロール ARN を選択し、ステートマシンの IAM Roles ページに移動します。
2. の IAM ロールに適切な権限を割り当てて、ParentStateMachine別のステートマシンの実行を開始できるようにします。これらのアクセス許可を割り当てるには、次の手順を実行します。

   a. IAM [ロール] ページで、[アクセス許可を追加] を選択し、次に [インラインポリシーを作成] を選択します。

   b. [ポリシーの作成] ページで、[JSON] タブを選択します。

   c. 既存のテキストを次のポリシーに置き換えます。

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

```
                "states:StartExecution"
            ],
            "Resource": [
                "arn:aws:states:us-
east-2:123456789012:stateMachine:LambdaStateMachine"
            ]
        }
    ]
}
```

d. [ポリシーの確認] を選択します。

e. ポリシーの名前を指定し、[ポリシーの作成] を選択します。

ステップ 4: ステートマシンを実行する

ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。

1. ParentStateMachineページで [実行開始] を選択します。

   [実行を開始] ダイアログが表示されます。

2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   b. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   c. [実行のスタート] を選択します。

   d. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

3. LambdaStateMachineページを開いて、によってトリガーされた新しい実行を確認します。ParentStateMachine

Lambda 関数を使用して新しい実行を継続する

Lambda 関数を使用して、現在の実行を終了させる前に新しい実行を開始するステートマシンを作成できます。新しい実行で進行中の作業を継続するためにこのアプローチを使うことにより、大きなジョブを小さなワークフローに分割するステートマシンや、無期限に実行するステートマシンを使用できます。

このチュートリアルは、Lambda を使用してループを反復処理する チュートリアルで説明した、ワークフローを変更する外部 Lambda 関数を使用する概念に基づいています。同じ Lambda 関数 (`Iterator`) を使用して、特定の回数ループを繰り返します。また、別の Lambda 関数を作成し、新しいワークフローの実行をスタートしたり、新しい実行をスタートするたびにカウントを減少させることができます。入力の実行回数を設定することにより、このステートマシンは終了し、指定された回数だけ実行を再開します。

作成するステートマシンは、次の状態を実装します。

都道府県	目的
`ConfigureCount`	`Pass` 状態は、作業の反復を進めるステップ `Iterator` Lambda 関数で使用される、`count`、`index`、および `step` 値を設定します。
`Iterator`	`Task` 状態は、`Iterator` Lambda 関数を参照します。
`IsCountReached`	`Choice` 関数からのブール値を使用する `Iterator` 状態は、ステートマシンがサンプルの作業を続行するか、`ShouldRestart` 状態に移行するかを決定します。
`ExampleWork`	`Pass` 状態は、実際の実装で作業を実行する `Task` 状態を表します。

都道府県	目的
ShouldRestart	Choice 状態は、ある実行を終了して別の実行を開始するか、単に終了するかを決定するために executionCount 値を使用します。
Restart	新しいステートマシンの実行をスタートする Lambda 関数を使用した Task状態。また、Iterator 関数と同様に、この関数はカウントを減らします。Restart 状態は、減少したカウント値を新しい実行の入力に渡します。

前提条件

開始する前に Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルを参照して、Lambda と Step Functions の使用方法に慣れておいてください。

トピック

- ステップ 1: Lambda 関数を作成してカウントを反復する
- ステップ 2: 新しい Step Functions の実行を開始する Lambda 関数を再開する
- ステップ 3: ステートマシンを作成する
- ステップ 4: IAM ポリシーを変更する
- ステップ 5: ステートマシンを実行する

ステップ 1: Lambda 関数を作成してカウントを反復する

> **Note**
>
> Lambda を使用してループを反復処理する チュートリアルを完了している場合は、このステップをスキップして、その Lambda 関数を使用することができます。

このセクションと、Lambda を使用してループを反復処理する チュートリアルでは、Lambda 関数を使用して、ステートマシンのループの反復回数のカウントを追跡する方法などを示します。

次の Lambda 関数は、count、index、step の入力値を受け取ります。これらの値を、更新された index と、continue というブール値を使用して返します。index が count 未満の場合、Lambda 関数は continue を true に設定します。

その後、ステートマシンは、continue が true の場合に、アプリケーションロジックを実行する Choice 状態を実装し、continue が false の場合に、ShouldRestart に移行します。

反復処理 Lambda 関数を作成する

1. Lambda コンソールを開き、[関数の作成] を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。
3. [ベーシック情報] セクションで、以下のように Lambda 関数を設定します。

   a. [Function name] (関数名) に Iterator と入力します。

   b. [ランタイム] で [Node.js 16.x] を選択します。

   c. ページのデフォルトの選択をすべて維持して、[関数を作成] を選択します。

      Lambda 関数が作成されたら、ページの右上隅に表示されている Amazon リソースネーム (ARN) を記録します。例:

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

4. Lambda 関数の以下のコードを、Lambda コンソールの ***[######]*** ページの [コードソース] セクションにコピーします。

```
exports.handler = function iterator (event, context, callback) {
  let index = event.iterator.index;
  let step = event.iterator.step;
  let count = event.iterator.count;

  index = index + step;

  callback(null, {
    index,
    step,
    count,
    continue: index < count
  })
}
```

このコードは、count、index、step の入力値を受け入れています。index を step の値増分し、それらの値とブール値 continue を返します。index が count 未満の場合、continue の値は true です。

5. [デプロイ] を選択して、コードをデプロイします。

反復処理 Lambda 関数をテストする

Iterate 関数の動作を確認するには、関数を数値で実行します。Lambda 関数の反復を表す入力値を指定し、入力値別の出力を確認できます。

Lambda 関数をテストします。

1. [Configure test event] (テストイベントの設定) ダイアログボックスで、[Create new test event] (新規テストイベントを作成) を選択し、[Event name] (イベント名) に TestIterator と入力します。
2. データ例を以下に置き換えます。

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

これらの値は、反復でステートマシンから返される値を表します。Lambda 関数はインデックスを増分し、continue を true として返します。インデックスが count 以上になると、continue を false として返します。このテストでは、インデックスは既に 5 に増分されています。結果では、index が 6 に増分され、continue が true に設定されます。

3. [作成] を選択します。
4. Lambda コンソールの ***Iterator*** ページで、TestIteratorが表示されていることを確認し、[テスト] を選択します。

テストの結果がページの上部に表示されます。[Details] (詳細) を選択し、結果を確認します。

```
{
  "index": 6,
```

```
  "step": 1,
  "count": 10,
  "continue": true
}
```

> **Note**
>
> このテストで `index` を 9 に設定した場合、`index` が `10` に増分され、`continue` が `false` になります。

ステップ 2: 新しい Step Functions の実行を開始する Lambda 関数を再開する

1. Lambda コンソールを開き、[関数の作成] を選択します。
2. [関数の作成] ページで、[一から作成] を選択します。
3. [ベーシック情報] セクションで、以下のように Lambda 関数を設定します。
   a. [Function name] (関数名) に `Restart` と入力します。
   b. [ランタイム] で [Node.js 16.x] を選択します。
4. ページのデフォルトの選択をすべて維持して、[関数を作成] を選択します。

   Lambda 関数が作成されたら、ページの右上隅に表示されている Amazon リソースネーム (ARN) を記録します。例:

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

5. Lambda 関数の以下のコードを、Lambda コンソールの ***[######]*** ページの [コードソース] (設定) セクションにコピーします。

   次のコードは、実行回数を減らし、減少した値を含め、ステートマシンの新しい実行をスタートします。

   ```
   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);
  });

}
```

6. [デプロイ] を選択して、コードをデプロイします。

ステップ 3: ステートマシンを作成する

これで 2 つの Lambda 関数を作成したので、ステートマシンを作成します。このステートマシンで、ShouldRestart および Restart 状態は、複数の実行で作業を分割する方法を示します。

Example ShouldRestart 選択状態

次の出力例に、ShouldRestartChoice 状態が示されています。この状態は、実行を再開すべきかどうかを決定します。

```
"ShouldRestart": {
"Type": "Choice",
"Choices": [
  {
    "Variable": "$.restart.executionCount",
    "NumericGreaterThan": 1,
    "Next": "Restart"
  }
],
```

この $.restart.executionCount 値は、最初の実行の入力に含まれています。これは、Restart 関数が呼び出されるたびに 1 ずつ減らされ、それ以降の実行ごとに、入力に配置されます。

Example Restart タスク状態

次の出力例に、RestartTask 状態が示されています。この状態では、前のステップで作成した Lambda 関数を使用して、実行を再開し、カウントを減らして、スタートする残りの実行数を追跡します。

```
"Restart": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Restart",
  "Next": "Done"
},
```

ステートマシンを作成するには

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

   > **Important**
   > ステートマシンが、ステップ 1 とステップ 2 で以前に作成した Lambda AWS 関数と同じアカウントとリージョンにあることを確認してください。

2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. このチュートリアルでは、コードエディタ でステートマシンの Amazon ステートメント言語 (ASL) 定義を記述します。これを行うには、[コード] を選択します。
5. 既存のボイラープレートコードを削除して、次のコードを貼り付けます。このコードの ARN を、作成した Lambda 関数の ARN に置き換えることに注意してください。

```
{
    "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
        }
    }
}
```

6. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**ContinueAsNew**」と入力します。

7. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

   ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

8. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

   > **Note**
   >
   > Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

9. このステートマシンの Amazon リソースネーム (ARN) をテキストファイルに保存します。新しい Step Functions の実行を開始するには、Lambda 関数にアクセス許可を与えると同時に ARN を指定する必要があります。

ステップ 4: IAM ポリシーを変更する

Lambda 関数に新しい Step Functions 実行を開始する権限を確実に持つようにするには、`Restart` Lambda 関数で使用する IAM ロールにインラインポリシーをアタッチします。詳細については、「IAM ユーザーガイド」の「インラインポリシーの埋め込み」を参照してください。

> **Note**
>
> 前の例の `Resource` 行を更新して、`ContinueAsNew` ステートマシンの ARN を参照することができます。これは、特定のステートマシンの実行のみをスタートできるようにポリシーを制限します。

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": [
             "states:StartExecution"
         ],
         "Resource": "arn:aws:states:us-east-2:123456789012stateMachine:ContinueAsNew"
     }
 ]
}
```

ステップ 5: ステートマシンを実行する

実行をスタートするには、ステートマシンの ARN と、新しい実行をスタートする回数 `executionCount` を含む入力を指定します。

1. ContinueAsNewページで [実行開始] を選択します。

   [実行を開始] ダイアログが表示されます。

2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

b. [入力] ボックスに、次の JSON 入力を入力してワークフローを実行します。

```
{
  "restart": {
    "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:ContinueAsNew",
    "executionCount": 4
  }
}
```

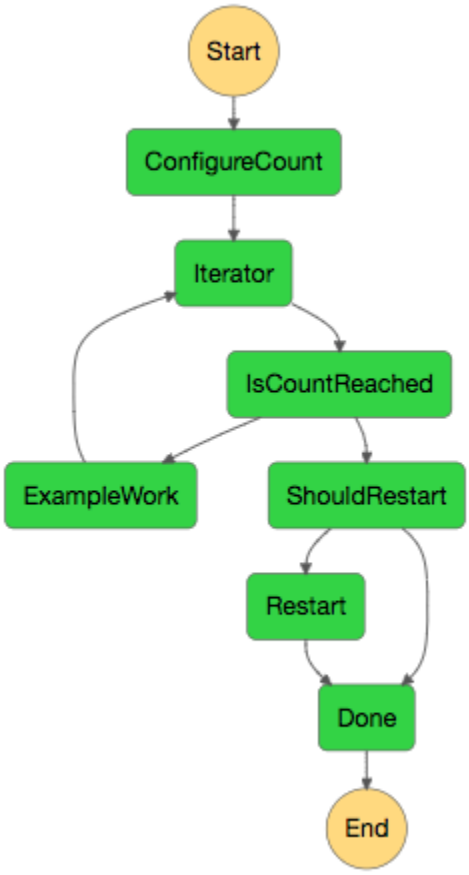
c. `ContinueAsNew` ステートマシンの ARN を使用して、`StateMachineArn` フィールドを更新します。

d. [実行のスタート] を選択します。

e. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

   [グラフビュー] に、4 回のうちの最初の実行が表示されます。完了する前に、`Restart` 状態がそのまま渡され、新しい実行をスタートします。



この実行が完了すると、実行中の次の実行を確認できます。ContinueAsNew上部のリンクを選択すると、実行リストが表示されます。最近終了した実行と、`Restart` Lambda 関数が開始した進行中の実行の両方が表示されるはずです。

すべての実行が完了すると、リストに 4 つの成功した実行が表示されます。最初の実行がスタートされると、選択した名前が表示され、その後の実行には生成された名前が付けられます。

人間による承諾プロジェクト例をデプロイする

このチュートリアルでは、 AWS Step Functions がタスク中に実行を停止し、ユーザーが E メールに返答するまで待機することができる人間による承諾プロジェクトをデプロイする方法を説明します。ユーザーによってタスクの進行が承認されてから、ワークフローは次の状態に進みます。

AWS CloudFormation このチュートリアルに含まれるスタックをデプロイすると、以下を含む必要なリソースがすべて作成されます。

- Amazon API Gateway リソース
- と関数 AWS Lambda
- AWS Step Functions 任意のステートマシン
- Amazon Simple Notification Service eメールトピック
- AWS Identity and Access Management 関連する役割と権限

> **Note**
>
> AWS CloudFormation スタックを作成するときには、アクセスできる有効な E メールアドレスを指定する必要があります。

詳細については、『AWS CloudFormation ユーザーガイド』の「CloudFormation AWS::StepFunctions::StateMachineテンプレートとリソースの使用」を参照してください。

トピック

- ステップ 1: AWS CloudFormation テンプレートを作成する
- ステップ 2: スタックを作成する
- ステップ 3: Amazon SNS サブスクリプションを承認する
- ステップ 4: ステートマシンを実行する
- AWS CloudFormation テンプレートソースコード

ステップ 1: AWS CloudFormation テンプレートを作成する

1. AWS CloudFormation テンプレートソースコード セクションからサンプルコードをコピーします。

2. AWS CloudFormation テンプレートのソースをローカルマシン上のファイルに貼り付けます。

   この例では、ファイルは `human-approval.yaml` という名前です。

ステップ 2: スタックを作成する

1. AWS CloudFormation コンソール にログインします。
2. [スタックの作成] を選択し、[新しいリソースを使用 (標準)] を選択します。
3. [Create stack] (スタックの作成) ページで、次の手順を実行します。

   a. [前提条件 - テンプレートの準備] セクションで、[テンプレートの準備完了] が選択されていることを確認します。

   b. [テンプレートを指定] セクションで [テンプレートファイルをアップロード] を選択し、次に [ファイルを選択] を選択し、先ほど作成した、テンプレートソースコードを含む `human-approval.yaml` ファイルをアップロードします。

4. [次へ] を選択します。
5. [スタックの詳細を指定] ページで、以下を実行します。

   a. [スタック名] に、スタックの名前を入力します。

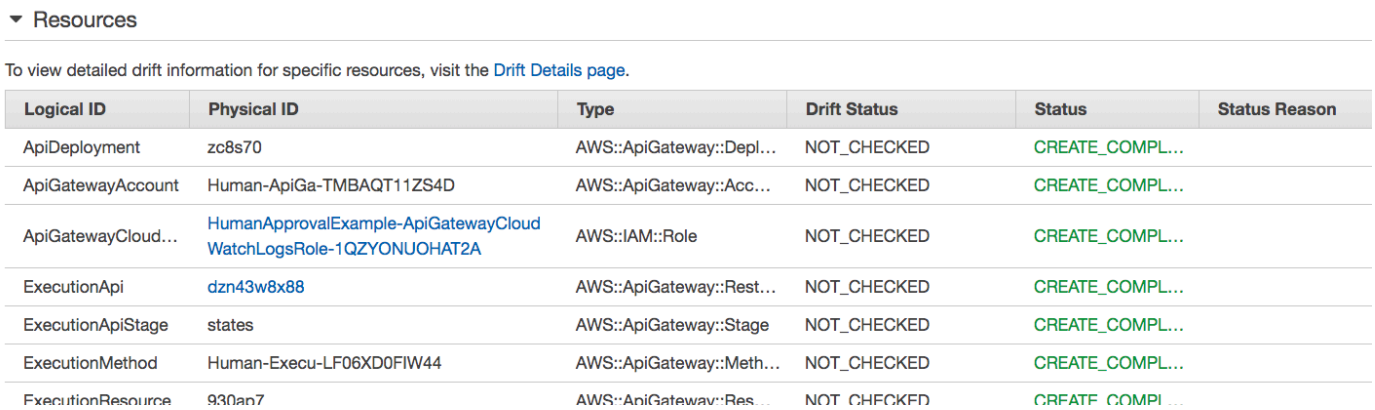
b. [パラメータ] に、有効な E メールアドレスを入力します。この E メールアドレスを使用して、Amazon SNS トピックをサブスクライブします。

6. [次へ] を選択し、もう一度 [次へ] を選択します。

7. 「レビュー」ページで、「IAM AWS CloudFormation リソースが作成される可能性があることを認めます」を選択し、「作成」を選択します。

   AWS CloudFormation スタックの作成が開始され、CREATE_IN_PROGRESS ステータスが表示されます。プロセスが完了すると、CREATE_COMPLETE AWS CloudFormation ステータスが表示されます。

8. (省略可能) スタックのリソースを表示するには、スタックを選択して [Resources] (リソース) タブを選択します。



ステップ 3: Amazon SNS サブスクリプションを承認する

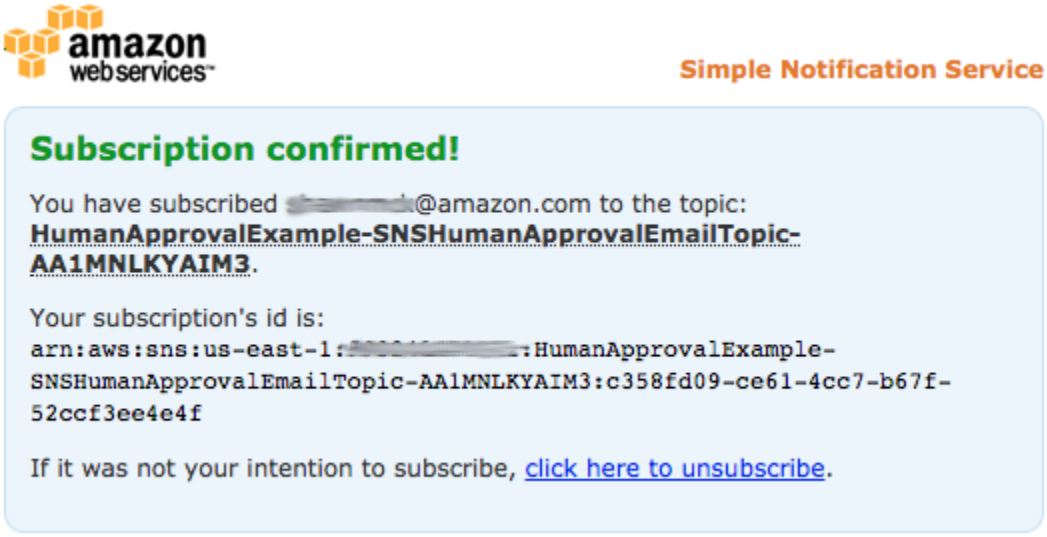
Amazon SNS トピックが作成されると、サブスクリプションの確認を求める E メールが届きます。

1. スタックの作成時に指定した E メールアカウントを開きます。 AWS CloudFormation

2. no-reply@sns.amazonaws.com からの AWS 通知を開く - サブスクリプションの確認メッセージを開きます

   E メールは Amazon SNS トピックの Amazon リソースネームと確認用のリンクを表示します。

3. [confirm subscription] (サブスクリプションの確認) リンクを選択します。



ステップ 4: ステートマシンを実行する

1. HumanApprovalLambdaStateMachineページで [実行開始] を選択します。

   [実行を開始] ダイアログが表示されます。

2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   b. [入力] ボックスに、次の JSON 入力を入力してワークフローを実行します。

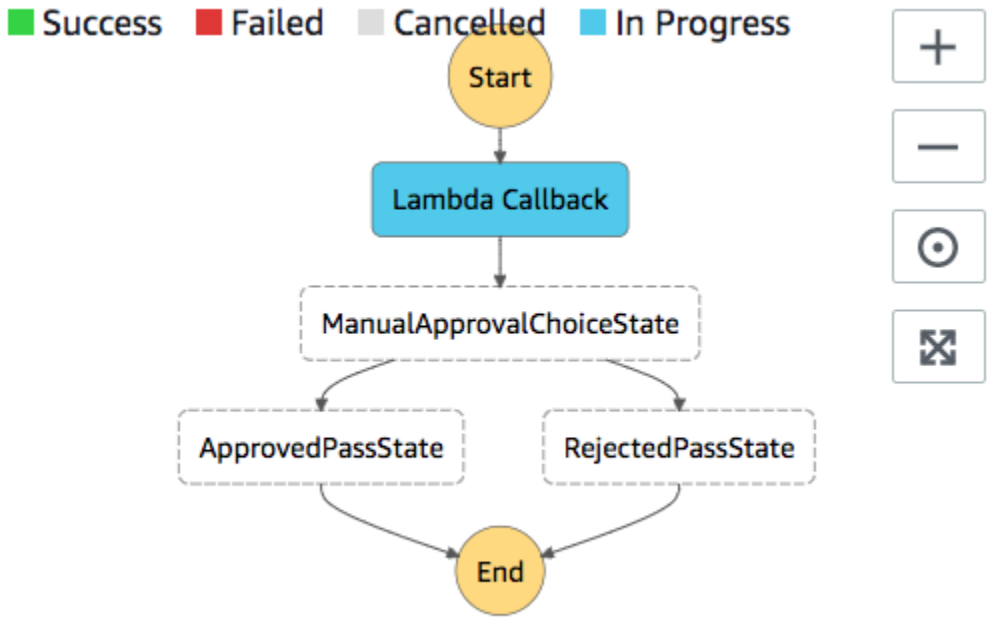
   ```
   {
       "Comment": "Testing the human approval tutorial."
   }
   ```

   c. [実行のスタート] を選択します。

ApprovalTestステートマシンの実行が開始され、Lambda コールバックタスクで一時停止します。

d. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

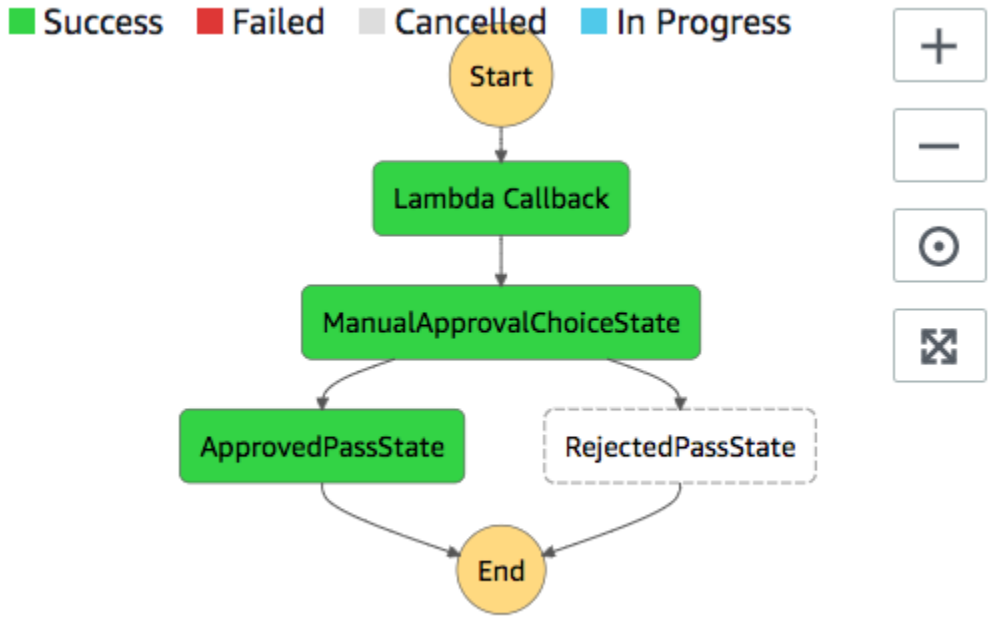


3. 先ほど Amazon SNS トピックで使用した E メールアカウントで、件名が「承認が必要」のメッセージを開きます。 AWS Step Functions

このメッセージには、[Approve] (承認) 用と [Reject] (拒否) 用に別々の URL が含まれています。

4. [Approve] (承認) URL を選択します。

選択に基づいてワークフローが続行されます。



AWS CloudFormation テンプレートソースコード

AWS CloudFormation このテンプレートを使用して、人間による承認プロセスのワークフローの例をデプロイしてください。

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "AWS Step Functions Human based task example. It sends an email with an
 HTTP URL for approval."
Parameters:
  Email:
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    ConstraintDescription: Must be a valid email address.
Resources:
  # Begin API Gateway Resources
  ExecutionApi:
    Type: "AWS::ApiGateway::RestApi"
    Properties:
      Name: "Human approval endpoint"
      Description: "HTTP Endpoint backed by API Gateway and Lambda"
      FailOnWarnings: true

  ExecutionResource:
```

```
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref ExecutionApi
      ParentId: !GetAtt "ExecutionApi.RootResourceId"
      PathPart: execution

  ExecutionMethod:
    Type: "AWS::ApiGateway::Method"
    Properties:
      AuthorizationType: NONE
      HttpMethod: GET
      Integration:
        Type: AWS
        IntegrationHttpMethod: POST
        Uri: !Sub "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${LambdaApprovalFunction.Arn}/invocations"
        IntegrationResponses:
          - StatusCode: 302
            ResponseParameters:
              method.response.header.Location:
 "integration.response.body.headers.Location"
        RequestTemplates:
          application/json: |
            {
              "body" : $input.json('$'),
              "headers": {
                #foreach($header in $input.params().header.keySet())
                "$header":
 "$util.escapeJavaScript($input.params().header.get($header))"
 #if($foreach.hasNext),#end

                #end
              },
              "method": "$context.httpMethod",
              "params": {
                #foreach($param in $input.params().path.keySet())
                "$param": "$util.escapeJavaScript($input.params().path.get($param))"
 #if($foreach.hasNext),#end

                #end
              },
              "query": {
                #foreach($queryParam in $input.params().querystring.keySet())
```

```
                "$queryParam":
"$util.escapeJavaScript($input.params().querystring.get($queryParam))"
#if($foreach.hasNext),#end

                #end
              }
            }
      ResourceId: !Ref ExecutionResource
      RestApiId: !Ref ExecutionApi
      MethodResponses:
        - StatusCode: 302
          ResponseParameters:
            method.response.header.Location: true

  ApiGatewayAccount:
    Type: 'AWS::ApiGateway::Account'
    Properties:
      CloudWatchRoleArn: !GetAtt "ApiGatewayCloudWatchLogsRole.Arn"

  ApiGatewayCloudWatchLogsRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - apigateway.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      Policies:
        - PolicyName: ApiGatewayLogsPolicy
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - "logs:*"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"

  ExecutionApiStage:
    DependsOn:
      - ApiGatewayAccount
```

```
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      DeploymentId: !Ref ApiDeployment
      MethodSettings:
        - DataTraceEnabled: true
          HttpMethod: '*'
          LoggingLevel: INFO
          ResourcePath: /*
      RestApiId: !Ref ExecutionApi
      StageName: states

  ApiDeployment:
    Type: "AWS::ApiGateway::Deployment"
    DependsOn:
      - ExecutionMethod
    Properties:
      RestApiId: !Ref ExecutionApi
      StageName: DummyStage
  # End API Gateway Resources

  # Begin
  # Lambda that will be invoked by API Gateway
  LambdaApprovalFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Code:
        ZipFile:
          Fn::Sub: |
            const { SFN: StepFunctions } = require("@aws-sdk/client-sfn");
            var redirectToStepFunctions = function(lambdaArn, statemachineName,
executionName, callback) {
              const lambdaArnTokens = lambdaArn.split(":");
              const partition = lambdaArnTokens[1];
              const region = lambdaArnTokens[3];
              const accountId = lambdaArnTokens[4];

              console.log("partition=" + partition);
              console.log("region=" + region);
              console.log("accountId=" + accountId);

              const executionArn = "arn:" + partition + ":states:" + region + ":" +
accountId + ":execution:" + statemachineName + ":" + executionName;
              console.log("executionArn=" + executionArn);
```

```
              const url = "https://console.aws.amazon.com/states/home?region=" + region
+ "#/executions/details/" + executionArn;
              callback(null, {
                  statusCode: 302,
                  headers: {
                    Location: url
                  }
              });
            };

            exports.handler = (event, context, callback) => {
              console.log('Event= ' + JSON.stringify(event));
              const action = event.query.action;
              const taskToken = event.query.taskToken;
              const statemachineName = event.query.sm;
              const executionName = event.query.ex;

              const stepfunctions = new StepFunctions();

              var message = "";

              if (action === "approve") {
                message = { "Status": "Approved! Task approved by ${Email}" };
              } else if (action === "reject") {
                message = { "Status": "Rejected! Task rejected by ${Email}" };
              } else {
                console.error("Unrecognized action. Expected: approve, reject.");
                callback({"Status": "Failed to process the request. Unrecognized
Action."});
              }

              stepfunctions.sendTaskSuccess({
                output: JSON.stringify(message),
                taskToken: event.query.taskToken
              })
              .then(function(data) {
                redirectToStepFunctions(context.invokedFunctionArn, statemachineName,
executionName, callback);
              }).catch(function(err) {
                console.error(err, err.stack);
                callback(err);
              });
            }
      Description: Lambda function that callback to AWS Step Functions
```

```
      FunctionName: LambdaApprovalFunction
      Handler: index.handler
      Role: !GetAtt "LambdaApiGatewayIAMRole.Arn"
      Runtime: nodejs18.x

  LambdaApiGatewayInvoke:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:InvokeFunction"
      FunctionName: !GetAtt "LambdaApprovalFunction.Arn"
      Principal: "apigateway.amazonaws.com"
      SourceArn: !Sub "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:
${ExecutionApi}/*"

  LambdaApiGatewayIAMRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Action:
              - "sts:AssumeRole"
            Effect: "Allow"
            Principal:
              Service:
                - "lambda.amazonaws.com"
      Policies:
        - PolicyName: CloudWatchLogsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "logs:*"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"
        - PolicyName: StepFunctionsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "states:SendTaskFailure"
                  - "states:SendTaskSuccess"
                Resource: "*"
  # End Lambda that will be invoked by API Gateway
```

```
  # Begin state machine that publishes to Lambda and sends an email with the link for
 approval
  HumanApprovalLambdaStateMachine:
    Type: AWS::StepFunctions::StateMachine
    Properties:
      RoleArn: !GetAtt LambdaStateMachineExecutionRole.Arn
      DefinitionString:
        Fn::Sub: |
          {
              "StartAt": "Lambda Callback",
              "TimeoutSeconds": 3600,
              "States": {
                  "Lambda Callback": {
                      "Type": "Task",
                      "Resource": "arn:
${AWS::Partition}:states:::lambda:invoke.waitForTaskToken",
                      "Parameters": {
                        "FunctionName": "${LambdaHumanApprovalSendEmailFunction.Arn}",
                        "Payload": {
                          "ExecutionContext.$": "$$",
                          "APIGatewayEndpoint": "https://${ExecutionApi}.execute-api.
${AWS::Region}.amazonaws.com/states"
                        }
                      },
                      "Next": "ManualApprovalChoiceState"
                  },
                  "ManualApprovalChoiceState": {
                    "Type": "Choice",
                    "Choices": [
                      {
                        "Variable": "$.Status",
                        "StringEquals": "Approved! Task approved by ${Email}",
                        "Next": "ApprovedPassState"
                      },
                      {
                        "Variable": "$.Status",
                        "StringEquals": "Rejected! Task rejected by ${Email}",
                        "Next": "RejectedPassState"
                      }
                    ]
                  },
                  "ApprovedPassState": {
                    "Type": "Pass",
                    "End": true
```

```
                },
                "RejectedPassState": {
                  "Type": "Pass",
                  "End": true
                }
            }
        }

  SNSHumanApprovalEmailTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        -
          Endpoint: !Sub ${Email}
          Protocol: email

  LambdaHumanApprovalSendEmailFunction:
    Type: "AWS::Lambda::Function"
    Properties:
      Handler: "index.lambda_handler"
      Role: !GetAtt LambdaSendEmailExecutionRole.Arn
      Runtime: "nodejs18.x"
      Timeout: "25"
      Code:
        ZipFile:
          Fn::Sub: |
            console.log('Loading function');
            const { SNS } = require("@aws-sdk/client-sns");
            exports.lambda_handler = (event, context, callback) => {
                console.log('event= ' + JSON.stringify(event));
                console.log('context= ' + JSON.stringify(context));

                const executionContext = event.ExecutionContext;
                console.log('executionContext= ' + executionContext);

                const executionName = executionContext.Execution.Name;
                console.log('executionName= ' + executionName);

                const statemachineName = executionContext.StateMachine.Name;
                console.log('statemachineName= ' + statemachineName);

                const taskToken = executionContext.Task.Token;
                console.log('taskToken= ' + taskToken);
```

```
                const apigwEndpint = event.APIGatewayEndpoint;
                console.log('apigwEndpint = ' + apigwEndpint)

                const approveEndpoint = apigwEndpint + "/execution?
action=approve&ex=" + executionName + "&sm=" + statemachineName + "&taskToken=" +
 encodeURIComponent(taskToken);
                console.log('approveEndpoint= ' + approveEndpoint);

                const rejectEndpoint = apigwEndpint + "/execution?
action=reject&ex=" + executionName + "&sm=" + statemachineName + "&taskToken=" +
 encodeURIComponent(taskToken);
                console.log('rejectEndpoint= ' + rejectEndpoint);

                const emailSnsTopic = "${SNSHumanApprovalEmailTopic}";
                console.log('emailSnsTopic= ' + emailSnsTopic);

                var emailMessage = 'Welcome! \n\n';
                emailMessage += 'This is an email requiring an approval for a step
 functions execution. \n\n'
                emailMessage += 'Please check the following information and click
 "Approve" link if you want to approve. \n\n'
                emailMessage += 'Execution Name -> ' + executionName + '\n\n'
                emailMessage += 'Approve ' + approveEndpoint + '\n\n'
                emailMessage += 'Reject ' + rejectEndpoint + '\n\n'
                emailMessage += 'Thanks for using Step functions!'

                const sns = new SNS();
                var params = {
                  Message: emailMessage,
                  Subject: "Required approval from AWS Step Functions",
                  TopicArn: emailSnsTopic
                };

                sns.publish(params)
                  .then(function(data) {
                    console.log("MessageID is " + data.MessageId);
                    callback(null);
                  }).catch(
                    function(err) {
                    console.error(err, err.stack);
                    callback(err);
                  });
            }
```

```
  LambdaStateMachineExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: states.amazonaws.com
            Action: "sts:AssumeRole"
      Policies:
        - PolicyName: InvokeCallbackLambda
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "lambda:InvokeFunction"
                Resource:
                  - !Sub "${LambdaHumanApprovalSendEmailFunction.Arn}"

  LambdaSendEmailExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: "sts:AssumeRole"
      Policies:
        - PolicyName: CloudWatchLogsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                  - "logs:PutLogEvents"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"
        - PolicyName: SNSSendEmailPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
```

```
                Action:
                  - "SNS:Publish"
                Resource:
                  - !Sub "${SNSHumanApprovalEmailTopic}"

# End state machine that publishes to Lambda and sends an email with the link for
 approval
Outputs:
  ApiGatewayInvokeURL:
    Value: !Sub "https://${ExecutionApi}.execute-api.${AWS::Region}.amazonaws.com/
states"
  StateMachineHumanApprovalArn:
    Value: !Ref HumanApprovalLambdaStateMachine
```

Step Functions で X-Ray によるトレースを表示する

このチュートリアルでは、X-Ray を使用してステートマシンの実行時に発生するエラーを追跡する方法について説明します。AWS X-Ray を使用して、ステートマシンのコンポーネントの視覚化、パフォーマンスのボトルネックの特定、およびエラーの原因となったリクエストのトラブルシューティングを行うことができます。このチュートリアルでは、ランダムにエラーを生成する Lambda 関数をいくつか作成し、X-Ray を使用してトレースおよび分析します。

Lambda を使用する Step Functions ステートマシン状態の作成 チュートリアルでは、 Lambda関数を呼び出すステートマシンを作成する手順について説明します。このチュートリアルを完了済みである場合は、[Step 2] (ステップ 2) に進み、前に作成した AWS Identity and Access Management (IAM) ロールを使用してください。

トピック

- ステップ 1: Lambda 用に IAM ロールを作成する
- ステップ 2: Lambda 関数を作成する
- ステップ 3: さらに 2 つの Lambda 関数を作成する
- ステップ 4: ステートマシンを作成する
- ステップ 5: ステートマシンを実行する

ステップ 1: Lambda 用に IAM ロールを作成する

AWS Lambda 両方とも、 AWS Step Functions コードを実行したり、 AWS リソース (Amazon S3 バケットに保存されているデータなど) にアクセスしたりできます。セキュリティを維持するために、Lambda および Step Functions にこれらのリソースへのアクセスを付与する必要があります。

Lambda では、Step Functions がステートマシンを作成するときに IAM ロールを割り当てる必要があるのと同じように、Lambda 関数を作成するときに AWS Identity and Access Management (IAM) ロールを割り当てる必要があります。

IAM コンソールを使用して、サービスにリンクされたロールを作成できます。

ロールを作成するには (コンソール)

1. AWS Management Console にサインインし、https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで [Roles] (ロール) を選択します。次に、[Create role] (ロールの作成) を選択します。
3. [AWS のサービス] ロールタイプを選択してから、Lambda を選択します。
4. ユースケースに [Lambda] を選択します。ユースケースは、サービスに必要な信頼ポリシーを含めるように定義されています。次に、[Next: Permissions] (次へ: 許可) を選択します。
5. ロールにアタッチするアクセス許可ポリシーを 1 つ以上選択します (例: `AWSLambdaBasicExecutionRole`)。AWS Lambda アクセス許可モデルを参照してください。

   ロールに許可するアクセス許可を割り当てるポリシーの横にあるボックスを選択し、[Next: Review] (次へ: レビュー) をクリックします。
6. [Role name] (ロール名) に入力します。
7. (オプション) [Role description] (ロールの説明) で、サービスにリンクされた新しいロールの説明を編集します。
8. ロールを確認し、[Create role] (ロールの作成) を選択します。

ステップ 2: Lambda 関数を作成する

Lambda 関数はエラーまたはタイムアウトをランダムにスローし、X-Ray で表示するサンプルデータを生成します。

> **Important**
>
> Lambda AWS AWS 関数がステートマシンと同じアカウントとリージョンにあることを確認してください。

1. Lambda コンソールを開き、[関数を作成] を選択します。
2. [Create function] (関数の作成) セクションで、[Author from scratch] (一から作成) を選択します。
3. [Basic information] (ベーシックな情報) セクションで、Lambda 関数を構成:

   a. [関数名] に `TestFunction1` と入力します。

   b. [ランタイム] で [Node.js 18.x] を選択します。

   c. [Role] (ロール) で、[Choose an existing role] (既存のロールを選択) を選択します。

   d. [既存のロール] で、前に作成した Lambda ロールを選択します。

   > **Note**
   >
   > 作成した IAM ロールがリストに表示されない場合は、そのロールが Lambda に伝達されるまであと数分かかる場合があります。

   e. [関数の作成] を選択します。

   Lambda 関数が作成されたら、ページの右上隅に表示されているその Amazon リソースネーム (ARN) を記録します。例:

   ```
   arn:aws:lambda:us-east-1:123456789012:function:TestFunction1
   ```

4. Lambda 関数の次のコードを ***TestFunction1*** ページの [関数コード] セクションにコピーします。

```
function getRandomSeconds(max) {
    return Math.floor(Math.random() * Math.floor(max)) * 1000;
}
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}
export const handler = async (event) => {
    if(getRandomSeconds(4) === 0) {
```

```
        throw new Error("Something went wrong!");
    }
    let wait_time = getRandomSeconds(5);
    await sleep(wait_time);
    return { 'response': true }
};
```

このコードは、時間切れの障害をランダムに作成します。これは、X-Ray トレースを使用して表示および分析できるステートマシンでエラーの例を生成するために使用されます。

5. [保存] を選択します。

ステップ 3: さらに 2 つの Lambda 関数を作成する

さらに 2 つの Lambda 関数を作成します。

1. ステップ 2 を繰り返して、さらに 2 つの Lambda 関数を作成します。次の関数については、[Function name] (関数名) に `TestFunction2` を入力します。最後の関数については、[Function name] (関数名) に `TestFunction3` を入力します。
2. Lambda コンソールで、`TestFunction1`、`TestFunction2`、`TestFunction3` の 3 つの Lambda 関数があることをチェックします

ステップ 4: ステートマシンを作成する

このステップでは、Step Functions コンソールを使用して、3 つの `Task` 状態を持つステートマシンを作成します。各 `Task` 状態 は 3 つの Lambda 関数のうちの 1 つをリファレンスします。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

> **Important**
>
> ステートマシンが、ステップ 2 とステップ 3 で作成した Lambda AWS 関数と同じアカウントとリージョンにあることを確認してください。

2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. このチュートリアルでは、コードエディタ でステートマシンの Amazon ステートメント言語 (ASL) 定義を記述します。これを行うには、[コード] を選択します。

5. 既存のボイラープレートコードを削除して、次のコードを貼り付けます。タスク状態定義で、この例の ARN を、作成した Lambda 関数の ARN に置き換えることに注意してください。

```
{
  "StartAt": "CallTestFunction1",
  "States": {
    "CallTestFunction1": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function1",
      "Catch": [
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
      "Next": "CallTestFunction2"
    },
    "CallTestFunction2": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function2",
      "Catch": [
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
      "Next": "CallTestFunction3"
    },
    "CallTestFunction3": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function3",
      "TimeoutSeconds": 5,
      "Catch": [
        {
          "ErrorEquals": [
            "States.Timeout"
          ],
          "Next": "AfterTimeout"
        },
```

```
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
      "Next": "Succeed"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "AfterTimeout": {
      "Type": "Fail"
    },
    "AfterTaskFailed": {
      "Type": "Fail"
    }
  }
}
```

これは、Amazon ステートメント言語を使用したステートマシンの説明です。Task、CallTestFunction1 および CallTestFunction2 という名前の 3 つの CallTestFunction3 状態を定義します。それぞれが 3 つの Lambda 関数のうちの 1 つを呼び出します。詳細については、「State Machine Structure」を参照してください。

6. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**TraceFunctions**」と入力します。

7. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   このチュートリアルでは、[追加設定] で [X-Ray トレースを有効にする] を選択します。[ステートマシンの設定] のその他のデフォルト設定をすべてそのまま使用します。

   ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

8. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

[ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 5: ステートマシンを実行する

ステートマシンの実行は、ワークフローを実行してタスクを実施するインスタンスです。

1. ***TraceFunctions***ページで [実行開始] を選択します。

   [新しい実行] ページが表示されます。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

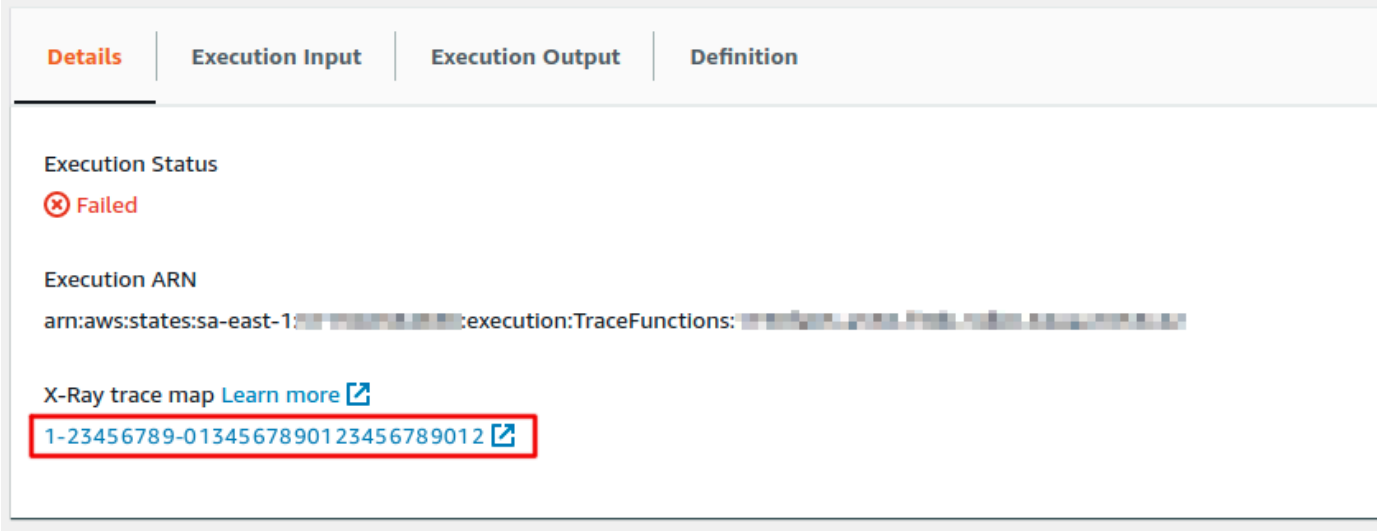
   b. [実行のスタート] を選択します。

   c. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

複数の (少なくとも 3 つ) の実行を行います。

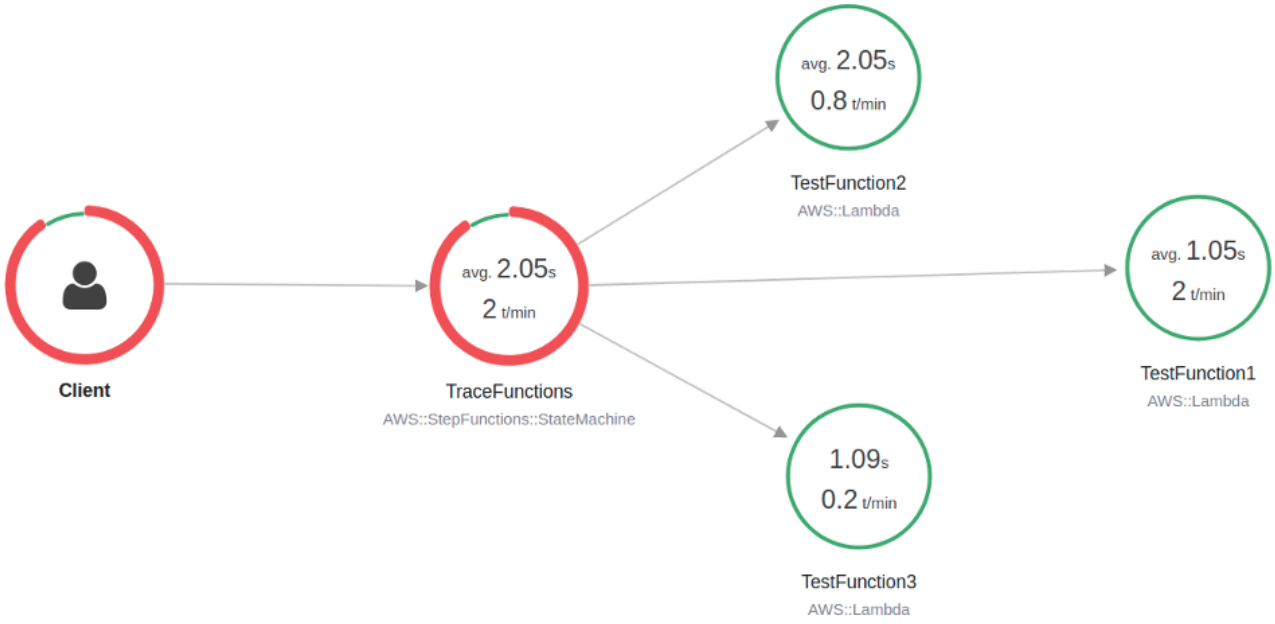
3. 実行が終了した後、[X-Ray trace map] (X-Ray トレースマップ) リンクに従います。実行中であっても、トレースを表示できますが、X-Ray トレースマップを表示する前に、実行結果を確認したくなるかもしれません。



4. サービスマップを表示して、エラーの発生場所、高レイテンシーの接続、失敗したリクエストのトレースを識別します。この例では、各関数が受信しているトラフィック量を確認できます。TestFunction2 の呼び出し回数は TestFunction3 よりも多く、TestFunction1 の呼び出し回数は TestFunction2 の 2 倍を超えました。

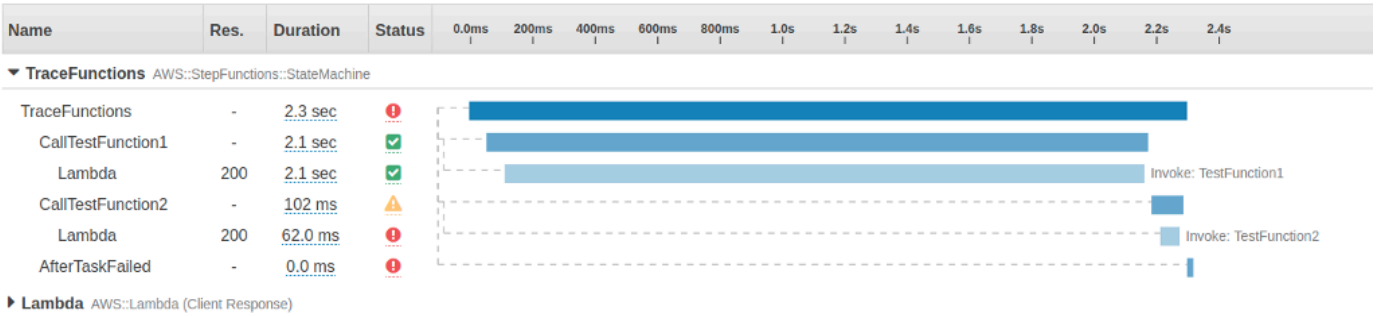
   このサービスマップは、各ノードの状態をエラーと障害に対する正常な呼び出しの比率に基づいて色分けしたものです。

   - 緑は、正常な呼び出し
   - 赤は、サーバー障害 (500 系のエラー)
   - 黄は、クライアントエラー (400 系のエラー)
   - 紫は、スロットリングエラー (429 リクエストが多すぎる)



そのノードのリクエスト、または 2 つのノード間のエッジを表示するサービスノードを選択して、その接続でやりとりされたリクエストを表示できます。

5. X-Ray トレースマップを表示して、実行ごとに何が起こったかを確認します。タイムラインビューには、セグメントとサブセグメントの階層を表示します。リスト内の最初のエントリは、1 回のリクエストに対してサービスによって記録されたすべてのデータを表すセグメントです。セグメントの下にサブセグメントがあります。この例では、Lambda 関数によって記録されたサブセグメントを表示しています。



X-Ray トレースを理解し、Step Functions を使って X-Ray を使用する方法の詳細については、AWS X-Ray および Step Functions を参照

AWS SDK サービスインテグレーションを使用して Amazon S3 バケット情報を収集する

このチュートリアルでは、Amazon Simple Storage Service による AWS SDK 統合の実行方法を学習します。このチュートリアルで作成するステートマシンは、Amazon S3 バケットに関する情報を収集し、現在のリージョンの各バケットのバージョン情報とともにバケットを一覧表示します。

トピック

- ステップ 1: ステートマシンを作成する
- ステップ 2: 必要な IAM ロールを追加する
- ステップ 3: Standard ステートステートマシンを実行する
- ステップ 4: Express ステートマシンを実行する

ステップ 1: ステートマシンを作成する

Step Functions コンソールを使用して、現在のアカウントとリージョンのすべての Amazon S3 バケットを一覧表示する Task 状態を含むステートマシンを作成します。次に、HeadBucket API を呼び出す Task 状態をもう 1 つ追加して、返されたバケットが現在のリージョンでアクセスできるかどうかを確認します。バケットにアクセスできない場合、HeadBucket API コールは S3.S3Exception エラーを返します。この例外を捕捉する Catch ブロックと、フォールバック状態として Pass 状態を含めます。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [テンプレートを選択] ダイアログボックスで [空白] を選択します。
3. [選択] を選びます。これにより、デザインモード で Workflow Studio が開きます。
4. このチュートリアルでは、コードエディタ でステートマシンの Amazon ステートメント言語 (ASL) 定義を記述します。これを行うには、[コード] を選択します。
5. 既存のボイラープレートコードを削除して、次のステートマシンの定義を貼り付けます。

```
{
  "Comment": "A description of my state machine",
  "StartAt": "ListBuckets",
  "States": {
    "ListBuckets": {
      "Type": "Task",
      "Parameters": {},
```

```
      "Resource": "arn:aws:states:::aws-sdk:s3:listBuckets",
      "Next": "Map"
    },
    "Map": {
      "Type": "Map",
      "ItemsPath": "$.Buckets",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "INLINE"
        },
        "StartAt": "HeadBucket",
        "States": {
          "HeadBucket": {
            "Type": "Task",
            "ResultPath": null,
            "Parameters": {
              "Bucket.$": "$.Name"
            },
            "Resource": "arn:aws:states:::aws-sdk:s3:headBucket",
            "Catch": [
              {
                "ErrorEquals": [
                  "S3.S3Exception"
                ],
                "ResultPath": null,
                "Next": "Pass"
              }
            ],
            "Next": "GetBucketVersioning"
          },
          "GetBucketVersioning": {
            "Type": "Task",
            "End": true,
            "Parameters": {
              "Bucket.$": "$.Name"
            },
            "ResultPath": "$.BucketVersioningInfo",
            "Resource": "arn:aws:states:::aws-sdk:s3:getBucketVersioning"
          },
          "Pass": {
            "Type": "Pass",
            "End": true,
            "Result": {
              "Status": "Unknown"
```

```
            },
            "ResultPath": "$.BucketVersioningInfo"
          }
        }
      },
      "End": true
    }
  }
}
```

6. ステートマシンの名前を指定します。これを行うには、デフォルトのステートマシン名の横にある編集アイコンを選択します。MyStateMachine次に、[ステートマシンの設定] の [ステートマシン名] ボックスに名前を指定します。

   このチュートリアルでは、名前として「**Gather-S3-Bucket-Info-Standard**」と入力します。

7. (オプション) [ステートマシンの設定] で、ステートマシンのタイプや実行ロールなど、他のワークフロー設定を指定します。

   [ステートマシンの設定] のデフォルト設定をすべてそのまま使用します。

   ステートマシンに適切なアクセス許可を持つ IAM ロールを以前に作成していて、そのロールを使用する場合は、[アクセス許可] で [既存のロールを選択] を選択し、一覧からロールを選択します。または、[ロールの ARN を入力] を選択し、その IAM ロールの ARN を指定します。

8. [ロールの作成を確認] ダイアログボックスで、[確認] を選択して続行します。

   [ロールの設定を表示] を選択して [ステートマシンの設定] に戻ることもできます。

> **Note**
>
> Step Functions が作成した IAM ロールを削除すると、Step Functions を後で再作成することはできません。同様に、ロールを変更すると (例えば、IAM ポリシーのプリンシパルから Step Functions を削除するなど)、後で Step Functions でそれを元の設定に復元することはできません。

ステップ 2 では、不足しているアクセス許可をステートマシンのロールに追加します。

ステップ 2: 必要な IAM ロールを追加する

現在のリージョンにある Amazon S3 バケットに関する情報を収集するには、Amazon S3 バケットにアクセスするために必要なアクセス許可をステートマシンに提供する必要があります。

1. ステートマシンのページで [IAM ロール ARN] を選択し、ステートマシンのロールの [ロール] ページを開きます。
2. [アクセス許可を追加]、[インラインポリシーを作成] の順に選択します。
3. [JSON] タブを選択して、次のアクセス許可を JSON エディタに貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*"
        }
    ]
}
```

4. [ポリシーの確認] を選択します。
5. [ポリシーの確認] でポリシーの [名前] に「**s3-bucket-permissions**」と入力します。
6. [ポリシーの作成] を選択します。

ステップ 3: Standard ステートステートマシンを実行する

1. [Gather-S3-Bucket-Info-Standard] ページで、[実行を開始] を選択します。
2. [実行を開始] ダイアログボックスで、以下の操作を行います。
   a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

b. [実行のスタート] を選択します。

c. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステップ 4: Express ステートマシンを実行する

1. ステップ 1 で提供されたステートマシン定義を使用して Express ステートマシンを作成します。ステップ 2 で説明したように、必要な IAM ロールのアクセス許可も必ず含めてください。

> **Tip**
>
> 前に作成した Standard マシンと区別するため、Express ステートマシンには **`Gather-S3-Bucket-Info-Express`** という名前を付けます。

2. [Gather-S3-Bucket-Info-Standard] ページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

a. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しませ

ん。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

b. [実行のスタート] を選択します。

c. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

デベロッパーツール

次のリソースには、サーバーレスワークフローの構築とステートマシンの操作に関する追加情報が含まれています。

- AWS CDK

- AWS Toolkit for VS Code

以下のトピックには、ステートマシンの作成、テスト、およびデバッグ方法を説明する情報が含まれています。

トピック

- 開発オプション
- AWS Step Functions および AWS SAM
- Application Composer の Workflow Studio を使用する
- AWS CloudFormationを使用して Step Functions 用の Lambda ステートマシンを作成する
- AWS CDK を使用してStep Functions 用 Lambda ステートマシンを作成する
- を使用した同期 Express ステートマシンを使用した API Gateway REST API の作成 AWS CDK
- AWS Step Functions Python 用データサイエンス SDK
- Terraform を使用する、ステートマシンのデプロイ

開発オプション

AWS Step Functions ステートマシンは、テストや開発にコンソール、SDK、ローカルバージョンの Step Functions を使用するなど、いくつかの方法で実装できます。

トピック

- Step Functions コンソール
- AWS SDK
- 標準ワークフローと Express ワークフロー
- HTTPS サービス API

- [Development environments] (開発環境)
- エンドポイント
- AWS CLI
- Step Functions Local
- AWS Toolkit for Visual Studio Code
- AWS Serverless Application Model とStep Functions
- Terraform と Step Functions
- 定義形式のサポート

Step Functions コンソール

ステートマシンは Step Functions コンソールを使用して定義できます。を使用してタスク用のコードを提供することで、 AWS Lambda ローカルの開発環境を使わずにクラウドで複雑なステートマシンを作成できます。作成が終わったら、Step Functions コンソールで Amazon States Language を使用してステートマシンを定義できます。

Lambda ステートマシンの作成チュートリアルでは、この方法を使用してシンプルなステートマシンを作成し、それを実行して結果を確認します。

データフローシミュレータ

Step Functions コンソールでワークフローを設計、実装、およびデバッグできます。JsonPath の入出力データ処理を使用して、ワークフロー内のデータの流れを制御することもできます。Step Functions コンソールのデータフローシミュレータを使用して、ステートからステートへの情報の流れを学び、複数のデータをフィルタリングして操作する方法を理解します。このツールを使用すると、Step Functions が 使用して、InputPath、Parameters、ResultSelector、OutputPath、ResultPathなどのデータを処理する各フィールドをシミュレーションします。

詳細については、「データフローシミュレータ」を参照してください。

AWS SDK

Step Functions は Java、.NET、Ruby、PHP、Python (Boto 3)、Go JavaScript、C++ の AWS SDK でサポートされています。これらの SDK は、複数のプログラミング言語で Step Functions HTTPS API アクションを使用する便利な方法を提供します。

これらの SDK ライブラリで公開されている API アクションを使用して、ステートマシン、アクティビティ、ステートマシンスターターを開発できます。また、これらのライブラリを使用して可視性オペレーションにアクセスし、独自の Step Functions のモニタリングおよびレポートツールを開発できます。

Step Functions AWS を他のサービスで使用するには、Amazon Web Services 最新の AWS SDK とツールのリファレンスドキュメントを参照してください。

> **Note**
>
> Step Functions は HTTPS エンドポイントのみをサポートしています。

標準ワークフローと Express ワークフロー

新しいステートマシンを作成するときは、Type に標準または Express のいずれかを選択する必要があります。どちらの場合も、Amazon ステートメント言語を使用してステートマシンを定義します。ステートマシンの実行の動作は、選択したタイプによって異なります。ステートマシンが作成された後は、選択したタイプを変更することはできません。

詳細については、CloudWatch Logs を使用した口グ記録 を参照してください。

HTTPS サービス API

Step Functions は HTTPS リクエスト経由でアクセス可能なサービスオペレーションを提供します。これらのオペレーションを使用して Step Functions と直接通信したり、HTTPS を介して Step Functions と通信できる任意の言語で独自のライブラリを開発したりできます。

サービス API アクションを使用するステートマシン、ワーカー、ステートマシンスターターを開発できます。また、API アクションを介して可視性オペレーションにアクセスし、独自のモニタリングおよびレポートツールを開発できます。

API アクションの詳細については、AWS Step Functions API リファレンスを参照してください。

[Development environments] (開発環境)

使用する予定のプログラミング言語に適した開発環境を設定する必要があります。

たとえば、Step Functions を Java を使用して開発するには、各開発ワークステーションに AWS SDK for Java、などの Java 開発環境をインストールする必要があります。Java 開発者が Eclipse

IDE を使用する場合は、 AWS Toolkit for Eclipseもインストールする必要があります。この Eclipse プラグインによって AWSの開発に役立つ機能が追加されます。

プログラミング言語にランタイム環境が必要な場合は、これらのプロセスを実行する各コンピュータに環境を設定する必要があります。

エンドポイント

レイテンシーを短縮し、要件を満たす場所にデータを保存するために、Step Functions AWS は異なるリージョンのエンドポイントを提供しています。

Step Functions の各エンドポイントは完全に独立しています。ステートマシンまたはアクティビティは、それが作成されたリージョン内にのみ存在します。あるリージョンで作成されたステートマシンおよびアクティビティは、別のリージョンで作成されたものとデータや属性を共有しません。例えば、2 つの異なるリージョンに `STATES-Flows-1` というステートマシンを登録できます。一方のリージョンの `STATES-Flows-1` ステートマシンは、もう一方のリージョンの `STATES-Flow-1` ステートマシンとはデータや属性を共有しません。

Step Functions エンドポイントのリストについては、「AWS 全般のリファレンス」の「AWS Step Functions のリージョンとエンドポイント」を参照してください。

AWS CLI

AWS Command Line Interface (AWS CLI) から多くのStep Functions 機能にアクセスできます。AWS CLI は、Step Functions コンソールを使用したり、場合によっては Step Functions API アクションを使用してプログラミングしたりする代わりに使用できます。例えば、 AWS CLI を使用してステートマシンを作成し、その後、既存のステートマシンを一覧表示できます。

AWS CLI で Step Functions コマンドを使用して、実行、アクティビティのポーリング、タスクハートビートの記録などを開始および管理できます。Step Functions コマンドがすべて記載されたリスト、使用可能な引数、および、使用方法を示す例については、「AWS CLI コマンドリファレンス」を参照してください。

AWS CLI コマンドは Amazon ステート言語に厳密に従うため、を使用してStep AWS CLI Functions API アクションについて学習できます。また、既存の API の知識を使用して、コマンドラインからコードのプロトタイプを作成したり Step Functions アクションを実行したりできます。

Step Functions Local

テストと開発のため、ローカルマシンで Step Functions をインストールおよび実行できます。Step Functions Local を使って、任意のマシンで実行をスタートできます。

Step Functions のローカルバージョンは、 AWS Lambda AWS ローカルで実行中とローカル実行中の両方で関数を呼び出すことができます。AWS サポートされている他のサービスを調整することもできます。詳細については、「ステートマシンのローカルテスト」を参照してください。

> **Note**
>
> Step Functions Local はダミーアカウントを使用して動作します。

AWS Toolkit for Visual Studio Code

VS コードを使用して、リモートステートマシンとやり取りし、ローカルでステートマシンを開発できます。ステートマシンの作成や更新、既存のステートマシンの一覧表示、ステートマシンの実行およびダウンロードを行うことができます。VS Code を使用して、テンプレートから新しいステートマシンを作成したり、ステートマシンを可視化したり、コードスニペット、コード補完、コード検証を行ったりすることもできます。

詳細については、『 AWS Toolkit for Visual Studio Code ユーザーガイド』を参照してください。

AWS Serverless Application Model とStep Functions

Step Functions はと統合されているため AWS Serverless Application Model、ワークフローを Lambda 関数、API、イベントと統合して、サーバーレスアプリケーションを作成できます。

AWS Toolkit for Visual Studio Code 統合エクスペリエンスの一部として AWS SAM CLI をと組み合わせて使用することもできます。

詳細については、「AWS Step Functions 」および「 AWS SAM」を参照してください。

Terraform と Step Functions

Terraform by HashiCorp は、コードとしてのインフラストラクチャ (IaC) を使用してアプリケーションを構築するためのフレームワークです。Terraform では、ステートマシンを作成したり、インフラストラクチャデプロイのプレビューや再利用可能なテンプレートの作成などの機能を使用したりできます。Terraform テンプレートを使用すると、コードが小さなチャンクに分割されるため、コードを保守して再利用しやすくなります。

詳細については、「Terraform を使用する、ステートマシンのデプロイ」を参照してください。

定義形式のサポート

Step Functions には、さまざまな形式でステートマシン定義を提供できるさまざまなツールが用意されています。ステートマシンの詳細を指定する Amazon ステートメント言語 (ASL) 定義は、文字列で、または JSON または YAML を使用してシリアル化されたオブジェクトとして提供できます。

> **Note**
>
> YAML は単一行のコメントを許可します。テンプレートのステートマシン定義部分で提供される YAML コメントは、作成されたリソースの定義には転送されません。代わりに、ステートマシン定義内の `Comment` プロパティを使用できます。詳細については、ステートマシン構造 のページを参照してください。

次の表で、どのツールが ASL ベースの定義をサポートしているかを示します。

ツールによる定義フォーマットのサポート

	JSON	YAML	Amazon ステートメント言語の文字列		
Step Functions コンソール	✓				
HTTPS サービス API			✓		
AWS CLI			✓		
Step Functions Local			✓		
AWS Toolkit for Visual Studio Code	✓	✓			
AWS SAM	✓	✓			

	JSON	YAML	Amazon ステートメント言語の文字列		
AWS CloudFormation	✓	✓	✓		

> **Note**
>
> AWS CloudFormation AWS SAM また、ステートマシン定義を JSON または YAML 形式で Amazon S3 にアップロードしたり、定義の Amazon S3 ロケーションをテンプレートに指定したりすることもできます。これにより、ステートマシンの定義が複雑な場合に、テンプレートの読みやすさを向上できます。詳細については、S3 ロケーションページを参照してください。AWS::StepFunctions::StateMachine

AWS CloudFormation 以下のサンプルテンプレートは、異なる入力形式を使用して同じステートマシン定義を提供する方法を示しています。

JSON with Definition

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS Step Functions sample template.",
  "Resources": {
    "MyStateMachine": {
      "Type": "AWS::StepFunctions::StateMachine",
      "Properties": {
        "RoleArn": {
          "Fn::GetAtt": [ "StateMachineRole", "Arn" ]
        },
        "TracingConfiguration": {
          "Enabled": true
        },
        "Definition": {
          "StartAt": "HelloWorld",
          "States": {
            "HelloWorld": {
```

```
            "Type": "Pass",
            "End": true
          }
        }
      }
    }
  },
  "StateMachineRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Action": [
              "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "states.amazonaws.com"
              ]
            }
          }
        ]
      },
      "ManagedPolicyArns": [],
      "Policies": [
        {
          "PolicyName": "StateMachineRolePolicy",
          "PolicyDocument": {
            "Statement": [
              {
                "Action": [
                  "lambda:InvokeFunction"
                ],
                "Resource": "*",
                "Effect": "Allow"
              }
            ]
          }
        }
      ]
    }
```

```
    }
  },
  "Outputs": {
    "StateMachineArn": {
      "Value": {
        "Ref": "MyStateMachine"
      }
    }
  }
}
```

JSON with DefinitionString

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS Step Functions sample template.",
  "Resources": {
    "MyStateMachine": {
      "Type": "AWS::StepFunctions::StateMachine",
      "Properties": {
        "RoleArn": {
          "Fn::GetAtt": [ "StateMachineRole", "Arn" ]
        },
        "TracingConfiguration": {
          "Enabled": true
        },
        "DefinitionString": "{\n  \"StartAt\": \"HelloWorld\",\n  \"States\": {\n
 \"HelloWorld\": {\n      \"Type\": \"Pass\",\n      \"End\": true\n    }\n  }\n}"
      }
    },
    "StateMachineRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Action": [
                "sts:AssumeRole"
              ],
              "Effect": "Allow",
              "Principal": {
                "Service": [
```

```
                  "states.amazonaws.com"
                ]
              }
            }
          ]
        },
        "ManagedPolicyArns": [],
        "Policies": [
          {
            "PolicyName": "StateMachineRolePolicy",
            "PolicyDocument": {
              "Statement": [
                {
                  "Action": [
                    "lambda:InvokeFunction"
                  ],
                  "Resource": "*",
                  "Effect": "Allow"
                }
              ]
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "StateMachineArn": {
      "Value": {
        "Ref": "MyStateMachine"
      }
    }
  }
}
```

YAML with Definition

```
AWSTemplateFormatVersion: 2010-09-09
Description: AWS Step Functions sample template.
Resources:
  MyStateMachine:
    Type: 'AWS::StepFunctions::StateMachine'
    Properties:
```

```
      RoleArn: !GetAtt
        - StateMachineRole
        - Arn
      TracingConfiguration:
        Enabled: true
      Definition:
        # This is a YAML comment. This will not be preserved in the state machine
 resource's definition.
        Comment: This is an ASL comment. This will be preserved in the state machine
 resource's definition.
        StartAt: HelloWorld
        States:
          HelloWorld:
            Type: Pass
            End: true
  StateMachineRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - states.amazonaws.com
      ManagedPolicyArns: []
      Policies:
        - PolicyName: StateMachineRolePolicy
          PolicyDocument:
            Statement:
              - Action:
                  - 'lambda:InvokeFunction'
                Resource: "*"
                Effect: Allow

Outputs:
  StateMachineArn:
    Value:
      Ref: MyStateMachine
```

YAML with DefinitionString

```
AWSTemplateFormatVersion: 2010-09-09
Description: AWS Step Functions sample template.
Resources:
  MyStateMachine:
    Type: 'AWS::StepFunctions::StateMachine'
    Properties:
      RoleArn: !GetAtt
        - StateMachineRole
        - Arn
      TracingConfiguration:
        Enabled: true
      DefinitionString: |
        {
            "StartAt": "HelloWorld",
            "States": {
                "HelloWorld": {
                    "Type": "Pass",
                    "End": true
                }
            }
        }
  StateMachineRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - states.amazonaws.com
      ManagedPolicyArns: []
      Policies:
        - PolicyName: StateMachineRolePolicy
          PolicyDocument:
            Statement:
              - Action:
                  - 'lambda:InvokeFunction'
                Resource: "*"
                Effect: Allow
```

```
Outputs:
  StateMachineArn:
    Value:
      Ref: MyStateMachine
```

AWS Step Functions および AWS SAM

AWS Toolkit for Visual Studio Code 統合エクスペリエンスの一部として AWS SAM CLI をと組み合わせて使用すると、ステートマシンをローカルに作成できます。 AWS SAMを使用してサーバーレスアプリケーションを構築し、VS Code IDE でステートマシンを構築できます。その後、リソースを検証、パッケージ化、デプロイできます。オプションで、 AWS Serverless Application Repositoryに公開することもできます。

> **Tip**
>
> を使用して Step Functions AWS SAM ワークフローを開始するサンプルサーバーレスアプリケーションをにデプロイするには AWS アカウント、「 AWS Step Functions ワークショップ」の「モジュール 11-Deploy with AWS SAMでデプロイ」を参照してください。

トピック

- AWS SAMでStep Functions を使用する理由
- Step Functions と AWS SAM 仕様 との統合
- SAM CLI との Step Functions の統合
- DefinitionSubstitutions テンプレートで AWS SAM
- 次のステップ

AWS SAMでStep Functions を使用する理由

Step Functions AWS SAM を一緒に使用すると、次のことができます。

- AWS SAM サンプルテンプレートを使い始めましょう。
- ステートマシンをサーバーレスアプリケーションにビルドします。
- 変数置換を使用して、デプロイ時に ARN をステートマシンに置き換えます。

AWS CloudFormation は、CloudFormation テンプレートに指定した値にワークフロー定義内の動的な参照を追加できる `DefinitionSubstitutions` をサポートしています。`${dollar_sign_brace}` 表記を使用してワークフロー定義に代替を追加することで、動的な参照を追加できます。また、`DefinitionSubstitutions` StateMachine CloudFormationこれらの動的参照はテンプレート内のリソースのプロパティで定義する必要があります。これらの置換は、CloudFormation スタックの作成プロセス中に実際の値に置き換えられます。詳細については、「DefinitionSubstitutions テンプレートで AWS SAM」を参照してください。

- AWS SAM ポリシーテンプレートを使用してステートマシンのロールを指定します。
- API Gateway、 EventBridge イベント、またはテンプレート内のスケジュールに基づいて、ステートマシンの実行を開始します。 AWS SAM

Step Functions と AWS SAM 仕様 との統合

使用可能な AWS SAM ポリシーテンプレート を使用すると、ステートマシンにアクセス許可を追加できます。これらのアクセス許可では、Lambda 関数やその他の AWS リソースをオーケストレートして、複雑で堅牢なワークフローを形成を有効にできます。

SAM CLI との Step Functions の統合

Step Functions は AWS SAM CLI と統合されています。これを使用して、ステートマシンをサーバーレスアプリケーションにすばやく開発します。

AWS SAM を使用して Step Functions ステートマシンを作成するチュートリアルを試して、 AWS SAM ステートマシンを作成する方法を学んでください。

サポートされている AWS SAM CLI 機能には以下が含まれます。

CLI コマンド	説明
sam init	テンプレートを使用してサーバーレスアプリケーションを初期化します。 AWS SAM Step Functions 用の SAM テンプレートで使用できます。
sam validate	テンプレートを検証します。 AWS SAM

CLI コマンド	説明
sam package	AWS SAM アプリケーションをパッケージ化します。コードと依存関係の ZIP ファイルを作成し、次にそれを Amazon S3 にアップロードします。次に、 AWS SAM テンプレートのコピーを返し、ローカルのアーティファクトへの参照を、コマンドがアーティファクトをアップロードした Amazon S3 の場所に置き換えます。
sam deploy	AWS SAM アプリケーションをデプロイします。
sam publish	AWS SAM アプリケーションをに公開します。AWS Serverless Application Repositoryこのコマンドは、 AWS SAM パッケージ化されたテンプレートを使用して、指定されたリージョンにアプリケーションを公開します。

> **Note**
>
> AWS SAM ローカルを使用するときは、Lambda および API Gateway をローカルでエミュレートできます。ただし、 AWS SAMを使用してStep Functionsをローカルでエミュレートすることはできません。

DefinitionSubstitutions テンプレートで AWS SAM

ステートマシンは、AWS SAM で CloudFormation テンプレートを使用して定義できます。AWS SAM では、ステートマシンをテンプレート内でインラインで定義するか、別個のファイルに定義できます。以下の AWS SAM テンプレートには、株取引のワークフローをシミュレートするステートマシンが含まれています。このステートマシンは 3 つの Lambda 関数を呼び出して、株の価格を確認し、株を売買するかを決定します。その後、この取引は Amazon DynamoDB テーブルに記録されます。次のテンプレートの Lambda 関数と DynamoDB テーブルの ARN は、`DefinitionSubstitutions` を使用して指定されます。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: |
  step-functions-stock-trader
  Sample SAM Template for step-functions-stock-trader
Resources:
  StockTradingStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionSubstitutions:
        StockCheckerFunctionArn: !GetAtt StockCheckerFunction.Arn
        StockSellerFunctionArn: !GetAtt StockSellerFunction.Arn
        StockBuyerFunctionArn: !GetAtt StockBuyerFunction.Arn
        DDBPutItem: !Sub arn:${AWS::Partition}:states:::dynamodb:putItem
        DDBTable: !Ref TransactionTable
      Policies:
        - DynamoDBWritePolicy:
            TableName: !Ref TransactionTable
        - LambdaInvokePolicy:
            FunctionName: !Ref StockCheckerFunction
        - LambdaInvokePolicy:
            FunctionName: !Ref StockBuyerFunction
        - LambdaInvokePolicy:
            FunctionName: !Ref StockSellerFunction
      DefinitionUri: statemachine/stock_trader.asl.json
  StockCheckerFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: functions/stock-checker/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  StockSellerFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: functions/stock-seller/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  StockBuyerFunction:
    Type: AWS::Serverless::Function
```

```
    Properties:
      CodeUri: functions/stock-buyer/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  TransactionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
```

以下のコードは、AWS SAM を使用して Step Functions ステートマシンを作成する チュートリアルで使用する、stock_trader.asl.json ファイルのステートマシン定義です。このステートマシン定義には、${dollar_sign_brace} 表記で示される複数の DefinitionSubstitutions が含まれています。例えば、Check Stock Value タスクに静的 Lambda 関数 ARN を指定する代わりに、置換 ${StockCheckerFunctionArn} が使用されます。この置換はテンプレートの DefinitionSubstitutions プロパティで定義されます。DefinitionSubstitutions はステートマシンリソースのキーと値のペアのマップです。ではDefinitionSubstitutions、${StockCheckerFunctionArn} StockCheckerFunction CloudFormation は組み込み関数を使用してリソースの ARN にマップされます。!GetAttAWS SAM テンプレートをデプロイすると、テンプレート内の DefinitionSubstitutions は実際の値に置き換えられます。

```
{
    "Comment": "A state machine that does mock stock trading.",
    "StartAt": "Check Stock Value",
    "States": {
        "Check Stock Value": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
                "Payload.$": "$",
                "FunctionName": "${StockCheckerFunctionArn}"
            },
            "Next": "Buy or Sell?"
        },
        "Buy or Sell?": {
            "Type": "Choice",
            "Choices": [
                {
```

```
                "Variable": "$.stock_price",
                "NumericLessThanEquals": 50,
                "Next": "Buy Stock"
            }
        ],
        "Default": "Sell Stock"
    },
    "Buy Stock": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
            "Payload.$": "$",
            "FunctionName": "${StockBuyerFunctionArn}"
        },
        "Retry": [
            {
                "ErrorEquals": [
                    "Lambda.ServiceException",
                    "Lambda.AWSLambdaException",
                    "Lambda.SdkClientException",
                    "Lambda.TooManyRequestsException"
                ],
                "IntervalSeconds": 1,
                "MaxAttempts": 3,
                "BackoffRate": 2
            }
        ],
        "Next": "Record Transaction"
    },
    "Sell Stock": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
            "Payload.$": "$",
            "FunctionName": "${StockSellerFunctionArn}"
        },
        "Next": "Record Transaction"
    },
    "Record Transaction": {
        "Type": "Task",
        "Resource": "arn:aws:states:::dynamodb:putItem",
        "Parameters": {
```

```
                "TableName": "${DDBTable}",
                "Item": {
                    "Id": {
                        "S.$": "$.id"
                    },
                    "Type": {
                        "S.$": "$.type"
                    },
                    "Price": {
                        "N.$": "$.price"
                    },
                    "Quantity": {
                        "N.$": "$.qty"
                    },
                    "Timestamp": {
                        "S.$": "$.timestamp"
                    }
                }
            },
            "End": true
        }
    }
}
```

次のステップ

Step Functions の使用方法の詳細については、以下のリソースを参照してください。 AWS SAM

- AWS SAM を使用して Step Functions ステートマシンを作成するチュートリアルを完了して、AWS SAMでステートマシンを作成してください。
- AWS::Serverless::StateMachineリソースを指定してください。
- 使用する AWS SAM ポリシーテンプレート を検索します。
- Step Functions と共に AWS Toolkit for Visual Studio Code を使用
- AWS SAMで利用できる機能の詳細については「AWS SAM CLI リファレンス」を確認してください。

Application Composer の Workflow Studio などのビジュアルビルダーを使用して、Infrastructure as Code (IaC) でワークフローを設計および構築することもできます。詳細については、「Application Composer の Workflow Studio を使用する」を参照してください。

Application Composer の Workflow Studio を使用する

AWS Application Composer は、シンプルなグラフィカルインターフェイスを使用して AWS SAM および AWS CloudFormation テンプレートを開発するのに役立つビジュアルビルダーです。Application Composer では、ビジュアルキャンバスで AWS のサービス をドラッグ、グループ化、接続することでアプリケーションアーキテクチャを設計できます。Application Composer は、AWS SAM コマンドラインインターフェイス (AWS SAM CLI) または CloudFormation を使用してアプリケーションをデプロイするために使用できる Infrastructure as Code (IaC) テンプレートを設計から作成します。Application Composer の詳細については、「Application Composer とは」を参照してください。

Application Composer で使用できる Workflow Studio は、ワークフローの設計と構築に役立ちます。Application Composer の Workflow Studio は、視覚的な IaC 環境を提供するため、CloudFormation テンプレートなどの IaC ツールを使用して構築されたサーバーレスアプリケーションにワークフローを簡単に組み込むことができます Application Composer の Workflow Studio を使用すると、個々のワークフローステップが AWS リソースに接続され、AWS SAM テンプレートにリソース構成が生成されます。また、ワークフローの実行に必要な IAM アクセス許可も追加されます。Application Composer の Workflow Studio を使用すると、アプリケーションのプロトタイプを作成して、本番環境ですぐに使えるアプリケーションに転換することができます。

Application Composer の Workflow Studio を使用する際は、Application Composer キャンバスと Workflow Studio を自由に切り替えることができます。

トピック

- Application Composer の Workflow Studio を使用してサーバーレスワークフローを構築する
- Workflow Studio で CloudFormation の定義の置換を使用してリソースを動的に参照する
- サービス統合タスクを拡張コンポーネントカードに接続する
- 既存のプロジェクトをインポートしてローカルに同期する
- AWS Application Composer の Workflow Studio で使用できない機能

Application Composer の Workflow Studio を使用してサーバーレスワークフローを構築する

1. Application Composer コンソールを開き、[プロジェクトを作成] を選択してプロジェクトを作成します。
2. [リソース] パレットの検索フィールドに、「**`state machine`**」と入力します。

3. [Step Functions ステートマシン] リソースをキャンバスにドラッグします。
4. [Workflow Studio で編集] を選択し、ステートマシンリソースを編集します。

   次のアニメーションは、Workflow Studio に切り替えてステートマシン定義を編集する方法を示しています。
   Application Composer の Workflow Studio の使用方法を示すアニメーション。

   Application Composer で作成されたステートマシンのリソースを編集するための Workflow Studio との統合は、`AWS::Serverless::StateMachine` リソースでのみ可能です。この統合は、`AWS::StepFunctions::StateMachine` リソースを使用するテンプレートでは使用できません。

Workflow Studio で CloudFormation の定義の置換を使用してリソースを動的に参照する

Workflow Studio では、ワークフロー定義で CloudFormation 定義の置換を使用して、IaC テンプレートで定義したリソースを動的に参照できます。`${dollar_sign_brace}` 表記を使用してワークフロー定義にプレースホルダー置換を追加すると、CloudFormation スタックの作成プロセス中に実際の値に置き換えられます。定義の置換の詳細については、「DefinitionSubstitutions テンプレートで AWS SAM」を参照してください。

次のアニメーションは、ステートマシンの定義におけるリソースのプレースホルダー置換の方法を示します。
Application Composer の Workflow Studio を使用する際に、AWS Lambda 関数、定義の置換などのリソースを動的に参照する方法を示すアニメーション。

サービス統合タスクを拡張コンポーネントカードに接続する

最適化されたサービス統合を呼び出すタスクを、Application Composer キャンバス内の拡張コンポーネントカードに接続できます。これにより、ワークフロー定義の `${dollar_sign_brace}` 表記で指定されたプレースホルダー置換と `StateMachine` リソースの `DefinitionSubstitution` プロパティが自動的にマッピングされます。また、ステートマシンに適した AWS SAM ポリシーも追加されます。

最適化されたサービス統合タスクを標準コンポーネントカードにマッピングすると、接続ラインは Application Composer キャンバスに表示されません。

次のアニメーションは、最適化されたタスクを拡張コンポーネントカードに接続し、[Change Inspector] で表示する方法を示しています。
Application Composer の Workflow Studio を使用する際に、最適化されたサービス統合を呼び出すタスクを拡張コンポーネントカードに接続する方法を示すアニメーション。

タスクステートの AWS SDK 統合 を拡張コンポーネントカードや最適化されたサービス統合と標準コンポーネントカードと接続することはできません。これらのタスクでは、Application Composer キャンバスの [リソースプロパティ] パネルで代替をマッピングし、AWS SAM テンプレートにポリシーを追加できます。

> **Tip**
>
> または、[リソースプロパティ] パネルの [定義の置換] で、ステートマシンのプレースホルダー置換をマッピングすることもできます。その場合は、AWS のサービス のタスクステート呼び出しに必要なアクセス許可をステートマシン実行ロールに追加する必要があります。実行ロールに必要なアクセス許可については、「Workflow Studio での実行ロール」を参照してください。

以下のアニメーションは、[リソースプロパティ] パネルのプレースホルダー置換マッピングを手動で更新する方法を示しています。
Application Composer の Workflow Studio を使用する際に、[リソースプロパティ] パネルのプレースホルダー代替マッピングを手動で更新する方法を示すアニメーション。

既存のプロジェクトをインポートしてローカルに同期する

Application Composer で既存の CloudFormation プロジェクトや AWS SAM プロジェクトを開いて視覚化し、理解を深めたり、デザインを変更したりできます。Application Composer のローカル同期機能を使用すると、テンプレートとコードファイルをローカルビルドマシンに自動的に同期して保存できます。ローカル同期モードを使用すると、既存の開発フローを補完できます。ブラウザが File System Access API をサポートしていることを確認してください。これにより、Web アプリケーションでローカルファイルシステム内のファイルの読み取り、書き込み、保存を行うことができます。Google Chrome または Microsoft Edge の使用が推奨されます。

AWS Application Composer の Workflow Studio で使用できない機能

Application Composer の Workflow Studio を使用すると、Workflow Studio の一部の機能が使用できなくなります。さらに、Inspector パネルにある [API パラメータ] セクションでは CloudFormation 定義の置換がサポートされています。`${dollar_sign_brace}` 表記を使用して置換を コードモー

ド に追加できます。この表記の詳細については、「DefinitionSubstitutions テンプレートで AWS SAM」を参照してください。

次のリストでは、Application Composer の Workflow Studio を使用する際に使用できない Workflow Studio の機能について説明します。

- スターターテンプレート — スターターテンプレートは、ワークフローのプロトタイプと定義を自動的に作成する、すぐに実行できるサンプルプロジェクトです。これらのテンプレートは、プロジェクトに必要なすべての関連する AWS リソースを AWS アカウント にデプロイします。
- 設定モード — このモードではステートマシンの設定を管理できます。IaC テンプレートでステートマシンの設定を更新するか、Application Composer キャンバスの [リソースプロパティ] パネルを使用できます。[リソースプロパティ] パネルの構成を更新する方法については、「サービス統合タスクを拡張コンポーネントカードに接続する」を参照してください。
- TestState API
- Workflow Studio の [アクション] ドロップダウンボタンからワークフロー定義をインポートまたはエクスポートするオプション。代わりに、Application Composer [メニュー] から [開く] > [プロジェクトフォルダ] を選択します。Application Composer キャンバス内の変更がローカルマシンに直接自動的に保存されるように、ローカル同期モードを有効にしていることを確認してください。
- [実行] ボタン。Application Composer の Workflow Studio を使用すると、Application Composer はワークフロー用の IaC コードを生成します。したがって、テンプレートをデプロイする必要があります。次に、コンソールまたは AWS Command Line Interface (AWS CLI) を使用してワークフローを実行します。

AWS CloudFormationを使用して Step Functions 用の Lambda ステートマシンを作成する

このチュートリアルでは、 AWS Lambda を使用して基本的な関数を作成する方法を説明します AWS CloudFormation。 AWS CloudFormation コンソールと YAML テンプレートを使用してスタック (IAM ロール、Lambda 関数、ステートマシン) を作成します。次に、 AWS Step Functions コンソールを使用してステートマシンの実行を開始します。

詳細については、『AWS CloudFormation ユーザーガイド』の「 CloudFormation AWS::StepFunctions::StateMachineテンプレートとリソースの操作」を参照してください。

トピック

- ステップ 1: AWS CloudFormation テンプレートを設定する

- ステップ 2: AWS CloudFormation テンプレートを使用して Lambda ステートマシンを作成する
- ステップ 3: ステートマシンの実行をスタートする

ステップ 1: AWS CloudFormation テンプレートを設定する

サンプルテンプレートを使用する前に、 AWS CloudFormation テンプレートのさまざまな部分を宣言する方法を理解しておく必要があります。

トピック

- Lambda 用に IAM ロールを作成するには
- Lambda 関数を作成するには
- ステートマシンの実行用の IAM ロールを作成するには
- Lambda ステートマシンを作成するには

Lambda 用に IAM ロールを作成するには

Lambda 関数の IAM ロールに関連付けられた信頼ポリシーを定義します。以下の例では、YAML または JSON を使用して信頼ポリシーを定義しています。

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"
                    }
                ]
            }
        }
```

Lambda 関数を作成するには

`Hello World` というメッセージを印刷する Lambda 関数の以下のプロパティを定義します。

> **Important**
>
> Lambda AWS AWS 関数がステートマシンと同じアカウントとリージョンにあることを確認してください。

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: "nodejs12.x"
    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": "nodejs12.x",
                "Timeout": "25"
            }
        },
```

ステートマシンの実行用の IAM ロールを作成するには

ステートマシンの実行の IAM ロールに関連付けられた信頼ポリシーを定義します。

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": "*"
                                }
                            ]
                        }
                    }
```

```
                ]
            }
        },
```

Lambda ステートマシンを作成するには

Lambda ステートマシンを定義します。

YAML

```
MyStateMachine:
  Type: "AWS::StepFunctions::StateMachine"
  Properties:
    DefinitionString:
      !Sub
        - |-
          {
            "Comment": "A Hello World 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

```
        "MyStateMachine": {
            "Type": "AWS::StepFunctions::StateMachine",
            "Properties": {
                "DefinitionString": {
                    "Fn::Sub": [
                        "{\n  \"Comment\": \"A Hello World 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"
                ]
            }
        }
    }
```

ステップ 2: AWS CloudFormation テンプレートを使用して Lambda ステートマシンを作成する

AWS CloudFormation テンプレートのコンポーネントを理解したら、それらをまとめてテンプレートを使用してスタックを作成できます。 AWS CloudFormation

Lambda ステートマシンを作成するには

1. 以下のサンプルデータを YAML 例の `MyStateMachine.yaml`、あるいは JSON の `MyStateMachine.json` という名前のファイルにコピーします。

   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: "nodejs12.x"
      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 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",
```

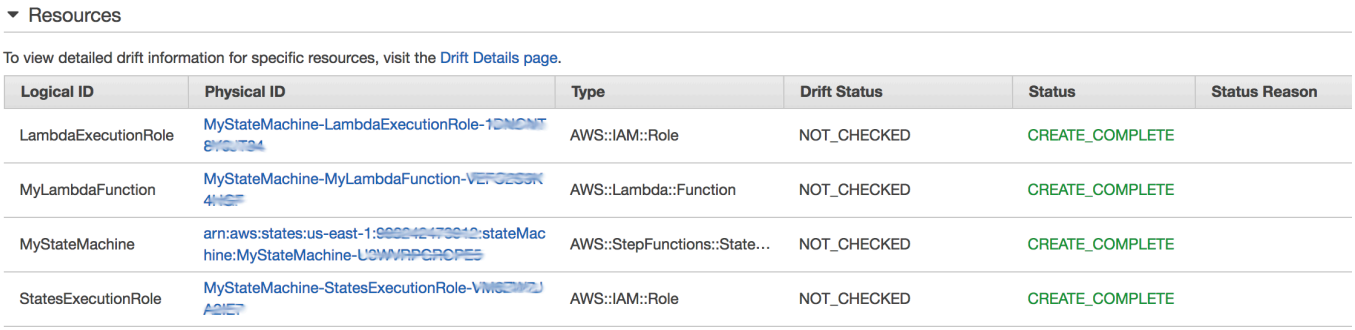
```
                "Role": {
                    "Fn::GetAtt": [
                        "LambdaExecutionRole",
                        "Arn"
                    ]
                },
                "Code": {
                    "ZipFile": "exports.handler = (event, context, callback) =>
 {\n    callback(null, \"Hello World!\");\n};\n"
                },
                "Runtime": "nodejs12.x",
                "Timeout": "25"
            }
        },
        "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": "*"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "MyStateMachine": {
        "Type": "AWS::StepFunctions::StateMachine",
        "Properties": {
            "DefinitionString": {
                "Fn::Sub": [
                    "{\n  \"Comment\": \"A Hello World 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"
                ]
            }
        }
    }
}
```

2. AWS CloudFormation コンソールを開き、[Create Stack] (スタックの作成) を選択します。

3. [Select Template] (テンプレートを選択) ページで、[Upload a template to Amazon S3] (Amazon S3 にテンプレートをアップロード) を選択します。`MyStateMachine` ファイルを選択し、[Next] (次へ) を選択します。
4. [Specify Details] (詳細の指定) ページで、[Stack name] (スタック名) に `MyStateMachine` を入力してから、[Next] (次へ) を選択します。
5. [Options] (オプション) ページで、[Next] (次へ) を選択します。
6. [Review] (レビュー) ページで、[I acknowledge that AWS CloudFormation might create resources] (によるIAM リソース作成の可能性があることを認める) を選択し、[Create] (作成) を選択します。

   AWS CloudFormation `MyStateMachine`スタックの作成を開始し、CREATE_IN_PROGRESS のステータスを表示します。プロセスが完了すると、 AWS CloudFormation に CREATE_COMPLETE ステータスが表示されます。
7. (省略可能) スタックのリソースを表示するには、スタックを選択して [Resources] (リソース) タブを選択します。



ステップ 3: ステートマシンの実行をスタートする

Lambda ステートマシンの作成後、実行を開始できます。

ステートマシンの実行をスタートするには

1. Step Functions コンソールを開き、を使用して作成したステートマシンの名前を選択します AWS CloudFormation。
2. ***MyStateMachine-ABCDEFGHIJ1K ####***、「新規実行」を選択します。

   [新しい実行] ページが表示されます。

3. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

4. [実行のスタート] を選択します。

   ステートマシンの新しい実行がスタートされ、実行中の実行が表示されている新しいページが表示されます。

5. (オプション) [Execution Details] (実行の詳細) で、[Execution Status] (実行ステータス) および [Started] (開始済み) と [Closed] (終了済み) のタイムスタンプをレビューします。

6. 実行結果を表示するには、[Output] (出力) タブを選択します。

AWS CDK を使用してStep Functions 用 Lambda ステートマシンを作成する

このチュートリアルでは、AWS Cloud Development Kit (AWS CDK) を使用する AWS Lambda 関数を含む AWS Step Functions ステートマシンの作成方法を示します。AWS CDKは、 AWS 本格的なプログラミング言語を使用してインフラストラクチャを定義できるコードとしてのインフラストラクチャ (IAC) フレームワークです。CDK のサポートされている言語の 1 つで、1 つ以上のスタックを含むアプリケーションを作成できます。その後、AWS CloudFormationそれをテンプレートにまとめて、アカウントにデプロイできます。 AWS Step FunctionsLambdaこのメソッドを使用して関数を含むステートマシンを定義し、を使用してステートマシンを実行します。 AWS Management Console

このチュートリアルを開始する前に、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「AWS CDK の使用開始 - 前提条件」の説明に従い、AWS CDK 開発環境をセットアップする必要があります。その後、AWS CLI で次のコマンドを使用して AWS CDK をインストールします。

```
npm install -g aws-cdk
```

このチュートリアルでは、the section called “を使用して Lambda ステートマシンを作成する AWS CloudFormation” と同じ結果になります。ただし、このチュートリアルでは、AWS CDK では IAM ロールの作成は必要なく、代わりに AWS CDK が作成します。AWS CDK バージョンには、ステートマシンにさらにステップを追加する方法を説明する 成功 ステップも含まれています。

> **Tip**
>
> AWS CDKwith Step Functions TypeScript を使用してワークフローを開始するサンプルサーバーレスアプリケーションをにデプロイするには AWS アカウント、「 AWS Step Functions ワークショップ」の「モジュール 10-Deploy with AWS CDK でデプロイ」を参照してください。

トピック

- ステップ 1:AWS CDK プロジェクトを設定する
- ステップ 2: AWS CDK を使用してステートマシンを作成する
- ステップ 3: ステートマシンの実行を開始する
- ステップ 4: クリーンアップする
- 次のステップ

ステップ 1:AWS CDK プロジェクトを設定する

1. ホームディレクトリか、必要に応じて別のディレクトリで、次のコマンドを実行して、新しい AWS CDK アプリケーション用のディレクトリを作成します。

   > **Important**
   >
   > 必ずディレクトリの名前はディレクトリ `step` としてください。AWS CDK アプリケーションテンプレートは、ディレクトリ名を使用し、ソースファイルとクラスの名前を生成します。別の名前を使用する場合は、アプリはこのチュートリアルと一致しません。

   TypeScript

   ```
   mkdir step && cd step
   ```

JavaScript

```
mkdir step && cd step
```

Python

```
mkdir step && cd step
```

Java

```
mkdir step && cd step
```

C#

.NET バージョン 6.0 以降がインストールされていることを確認してください。詳細については、「 サポートされるバージョン」を参照してください。

```
mkdir step && cd step
```

2. cdk init コマンドを使用してアプリケーションを初期化します。次の例に示すように、目的のテンプレート (「app」) とプログラミング言語を指定します。

TypeScript

```
cdk init --language typescript
```

JavaScript

```
cdk init --language javascript
```

Python

```
cdk init --language python
```

プロジェクトが初期化されたら、プロジェクトの仮想環境をアクティブにして、AWS CDK のベースラインの依存関係をインストールします。

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

Java

```
cdk init --language java
```

C#

```
cdk init --language csharp
```

ステップ 2: AWS CDK を使用してステートマシンを作成する

まず、Lambda 関数と Step Functions ステートマシンを定義する個々のコードを示します。次に、それらをまとめて AWS CDK アプリケーションに組み込む方法を説明します。最後に、これらのリソースを合成し、デプロイする方法を説明します。

Lambda 関数を作成するには

次のものは、Lambda 関数を定義する AWS CDK コードで、ソースコードをインラインで提供します。

TypeScript

```
const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
    code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
    runtime: lambda.Runtime.NODEJS_18_X,
    handler: "index.handler",
    timeout: cdk.Duration.seconds(3)
});
```

JavaScript

```
const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
    code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
```

```
          };
      `),
    runtime: lambda.Runtime.NODEJS_18_X,
    handler: "index.handler",
    timeout: cdk.Duration.seconds(3)
});
```

Python

```
hello_function = lambda_.Function(
            self, "MyLambdaFunction",
            code=lambda_.Code.from_inline("""
            exports.handler = (event, context, callback) => {
                callback(null, "Hello World!");
                }"""),
                runtime=lambda_.Runtime.NODEJS_18_X,
                handler="index.handler",
                timeout=Duration.seconds(25))
```

Java

```
final Function helloFunction = Function.Builder.create(this, "MyLambdaFunction")
        .code(Code.fromInline(
                "exports.handler = (event, context, callback) => { callback(null,
 'Hello World!' );}"))
        .runtime(Runtime.NODEJS_18_X)
        .handler("index.handler")
        .timeout(Duration.seconds(25))
        .build();
```

C#

```
var helloFunction = new Function(this, "MyLambdaFunction", new FunctionProps
{
    Code = Code.FromInline(@"`
      exports.handler = (event, context, callback) => {
        callback(null, 'Hello World!');
      }"),
    Runtime = Runtime.NODEJS_12_X,
    Handler = "index.handler",
    Timeout = Duration.Seconds(25)
});
```

以下に簡単なコード例を示します。

- 関数の論理名 `MyLambdaFunction`。
- AWS CDK アプリケーションのソースコードに文字列として埋め込まれている関数のソースコード。
- 使用するランタイム (Node 18.x)、関数のエントリポイント、タイムアウトなど、その他の関数属性。

ステートマシンを作成するには

ステートマシンには、Lambda 関数タスクと 成功 状態の 2 つの状態があります。この関数では、関数を呼び出す Step Functions the section called “タスク” を作成する必要があります。このタスク状態は、ステートマシンの最初のステップとして使用されます。成功状態は、そのタスク状態の `next()` メソッドを使用してステートマシンに追加されます。次のコードは、最初に `MyLambdaTask` という名前の関数を呼び出し、次に `next()` メソッドを使用して `GreetedWorld` という名前の成功状態を定義します。

TypeScript

```
const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
  definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
    lambdaFunction: helloFunction
  }).next(new sfn.Succeed(this, "GreetedWorld"))
});
```

JavaScript

```
const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
  definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
    lambdaFunction: helloFunction
  }).next(new sfn.Succeed(this, "GreetedWorld"))
});
```

Python

```
state_machine = sfn.StateMachine(
                                  self, "MyStateMachine",
                                  definition=tasks.LambdaInvoke(
                                  self, "MyLambdaTask",
                                  lambda_function=hello_function)
```

```
                                    .next(sfn.Succeed(self, "GreetedWorld")))
```

Java

```
final StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
        .definition(LambdaInvoke.Builder.create(this, "MyLambdaTask")
            .lambdaFunction(helloFunction)
            .build()
            .next(new Succeed(this, "GreetedWorld")))
        .build();
```

C#

```
var stateMachine = new StateMachine(this, "MyStateMachine", new StateMachineProps {
    DefinitionBody = DefinitionBody.FromChainable(new LambdaInvoke(this,
 "MyLambdaTask", new LambdaInvokeProps
    {
        LambdaFunction = helloFunction
    })
    .Next(new Succeed(this, "GreetedWorld")))
});
```

AWS CDK アプリケーションを構築してデプロイするには

新規に作成した AWS CDK プロジェクトで、次のコード例のように、スタック定義を含むファイルを編集します。Lambda 関数と Step Functions ステートマシンの定義については、前のセクションで説明しています。

1. 次の例に示すように、スタックを更新します。

   TypeScript

   次のコードを使用して、`lib/step-stack.ts` を更新します。

   ```
   import * as cdk from 'aws-cdk-lib';
   import * as lambda from 'aws-cdk-lib/aws-lambda';
   import * as sfn from 'aws-cdk-lib/aws-stepfunctions';
   import * as tasks from 'aws-cdk-lib/aws-stepfunctions-tasks';

   export class StepStack extends cdk.Stack {
   ```

```
  constructor(app: cdk.App, id: string) {
    super(app, id);

    const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
      code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "index.handler",
      timeout: cdk.Duration.seconds(3)
    });

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
      definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
        lambdaFunction: helloFunction
      }).next(new sfn.Succeed(this, "GreetedWorld"))
    });
  }
}
```

JavaScript

次のコードを使用して、lib/step-stack.js を更新します。

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as sfn from 'aws-cdk-lib/aws-stepfunctions';
import * as tasks from 'aws-cdk-lib/aws-stepfunctions-tasks';

export class StepStack extends cdk.Stack {
  constructor(app, id) {
    super(app, id);

    const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
      code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "index.handler",
```

```
      timeout: cdk.Duration.seconds(3)
    });

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
      definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
        lambdaFunction: helloFunction
      }).next(new sfn.Succeed(this, "GreetedWorld"))
    });
  }
}
```

Python

次のコードを使用して、step/step_stack.py を更新します。

```
from aws_cdk import (
    Duration,
    Stack,
    aws_stepfunctions as sfn,
    aws_stepfunctions_tasks as tasks,
    aws_lambda as lambda_
)
class StepStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        hello_function = lambda_.Function(
            self, "MyLambdaFunction",
            code=lambda_.Code.from_inline("""
            exports.handler = (event, context, callback) => {
                callback(null, "Hello World!");
                }"""),
                runtime=lambda_.Runtime.NODEJS_18_X,
                handler="index.handler",
                timeout=Duration.seconds(25))

        state_machine = sfn.StateMachine(
            self, "MyStateMachine",
            definition=tasks.LambdaInvoke(
            self, "MyLambdaTask",
            lambda_function=hello_function)
            .next(sfn.Succeed(self, "GreetedWorld")))
```

Java

次のコードを使用して、src/main/java/com.myorg/StepStack.java を更新します。

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.Duration;
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.stepfunctions.StateMachine;
import software.amazon.awscdk.services.stepfunctions.Succeed;
import software.amazon.awscdk.services.stepfunctions.tasks.LambdaInvoke;

public class StepStack extends Stack {
    public StepStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public StepStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        final Function helloFunction = Function.Builder.create(this,
 "MyLambdaFunction")
                .code(Code.fromInline(
                        "exports.handler = (event, context, callback) =>
 { callback(null, 'Hello World!' );}"))
                .runtime(Runtime.NODEJS_18_X)
                .handler("index.handler")
                .timeout(Duration.seconds(25))
                .build();

        final StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
                .definition(LambdaInvoke.Builder.create(this, "MyLambdaTask")
                        .lambdaFunction(helloFunction)
                        .build()
                        .next(new Succeed(this, "GreetedWorld")))
                .build();
```

```
    }
}
```

C#

次のコードを使用して、scr/Step/StepStack.cs を更新します。

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.StepFunctions.Tasks;

namespace Step
{
    public class StepStack : Stack
    {
        internal StepStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var helloFunction = new Function(this, "MyLambdaFunction", new
 FunctionProps
            {
                Code = Code.FromInline(@"exports.handler = (event, context,
 callback) => {
                    callback(null, 'Hello World!');
                }"),
                Runtime = Runtime.NODEJS_18_X,
                Handler = "index.handler",
                Timeout = Duration.Seconds(25)
            });

            var stateMachine = new StateMachine(this, "MyStateMachine", new
 StateMachineProps
            {
                DefinitionBody = DefinitionBody.FromChainable(new
 LambdaInvoke(this, "MyLambdaTask", new LambdaInvokeProps
                {
                    LambdaFunction = helloFunction
                })
                .Next(new Succeed(this, "GreetedWorld")))
            });
        }
```

```
    }
}
```

2. ソースファイルを保存し、アプリケーションのメインディレクトリで `cdk synth` コマンドを実行します。

   AWS CDK でアプリケーションを実行し、そのアプリケーションから AWS CloudFormation テンプレートを合成し、AWS CDK によってテンプレートを表示します。

   > **Note**
   >
   > TypeScript AWS CDK以前にプロジェクトを作成していた場合、`cdk synth`コマンドを実行すると次のエラーが返されることがあります。
   >
   > ```
   > TSError: # Unable to compile TypeScript:
   > bin/step.ts:7:33 - error TS2554: Expected 2 arguments, but got 3.
   > ```
   >
   > このエラーを解決するには、次の例に示すように `bin/step.ts` ファイルを変更します。
   >
   > ```
   > #!/usr/bin/env node
   > import 'source-map-support/register';
   > import * as cdk from 'aws-cdk-lib';
   > import { StepStack } from '../lib/step-stack';
   >
   > const app = new cdk.App();
   > new StepStack(app, 'StepStack');
   > app.synth();
   > ```

3. Lambda 関数と Step Functions ステートマシンを AWS アカウントにデプロイするには、`cdk deploy` を発行します。が生成した IAM ポリシーを承認するよう求められます。 AWS CDK

ステップ 3: ステートマシンの実行を開始する

ステートマシンを作成したら、実行を開始できます。

ステートマシンの実行をスタートするには

1. Step Functions コンソール を開き、AWS CDK を使用して作製したステートマシンを選択します。

2. [ステートマシン] ページで、[実行を開始] を選択します。

   [実行を開始] ダイアログが表示されます。

3. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

4. [実行のスタート] を選択します。

   ステートマシンの実行が開始され、実行中の実行が表示されている新しいページが表示されます。

5. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステップ 4: クリーンアップする

ステートマシンをテストが完了したら、ステートマシンと関連する Lambda 関数の両方を削除して、 AWS アカウント内のリソースを解放することを推奨します。ステートマシンを削除するには、アプリケーションのメインディレクトリで `cdk destroy` コマンドを実行します。

次のステップ

AWS を使用したインフラストラクチャーの開発について詳しくはAWS CDK、『AWS CDK開発者ガイド』を参照してください。

お好みの言語で AWS CDK アプリケーションを書く方法についての情報は、以下を参照してください:

TypeScript

AWS CDKin での作業 TypeScript

JavaScript

AWS CDKin での作業 JavaScript

Python

Python で AWS CDK を使用する

Java

Java で AWS CDK を使用する

C#

C# で AWS CDK を使用する

このチュートリアルで使用している AWS Construct Library モジュールの詳細については、以下の AWS CDK API リファレンスの概要を参照してください。

- aws-lambda
- aws-stepfunctions
- aws-stepfunctions-tasks

を使用した同期 Express ステートマシンを使用した API Gateway REST API の作成 AWS CDK

このチュートリアルでは、 AWS Cloud Development Kit (AWS CDK) を使用してバックエンドの統合として同期高速ステートマシンで API Gateway REST API を作成する方法について説明します。このチュートリアルでは、`StepFunctionsRestApi` コンストラクトを使用してステートマシンを API Gateway に接続します。`StepFunctionsRestApi` コンストラクトは、必要なアクセス許可と HTTP「ANY」メソッドを使用して、デフォルトの入出力マッピングと API Gateway REST API を設定します。 AWS CDK は、フルエッジのプログラミング言語を使用して AWS インフラストラクチャを定義できるコードとしてのインフラストラクチャ (IAC) フレームワークです。CDK がサポートする言語の 1 つでアプリケーションを記述し、1 つ以上のスタックを含めて、それを AWS

CloudFormation テンプレートに合成して AWS アカウントにデプロイします。これを使用して、同期 Express ステートマシンをバックエンドとして統合されている API Gateway REST API を定義し、 を使用して実行 AWS Management Console を開始します。

このチュートリアルを開始する前に、「 AWS CDK - の前提条件」の説明に従って AWS CDK 開発環境をセットアップし、発行 AWS CDK して をインストールします。

```
npm install -g aws-cdk
```

トピック

- ステップ 1: AWS CDK プロジェクトを設定する
- ステップ 2: AWS CDK を使用して、同期 Express ステートマシンバックエンド統合で API Gateway REST API を作成する
- ステップ 3: API Gateway をテストする
- ステップ 4: クリーンアップする

ステップ 1: AWS CDK プロジェクトを設定する

まず、新しい AWS CDK アプリケーションのディレクトリを作成し、プロジェクトを初期化します。

TypeScript

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language typescript
```

JavaScript

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language javascript
```

Python

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language python
```

プロジェクトの初期化が完了したら、プロジェクトの仮想環境をアクティブ化し、 AWS CDKのベースライン依存関係をインストールします。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language java
```

C#

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language csharp
```

Go

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language go
```

> **Note**
>
> 必ずディレクトリの名前はディレクトリ `stepfunctions-rest-api` としてください。 AWS CDK アプリケーションテンプレートは、 ディレクトリの名前を使用してソースファイルとクラスの名前を生成します。別の名前を使用する場合は、アプリはこのチュートリアルと一致しません。

次に、 AWS Step Functions と Amazon API Gateway のコンストラクトライブラリモジュールをインストールします。

TypeScript

```
npm install @aws-cdk/aws-stepfunctions @aws-cdk/aws-apigateway
```

JavaScript

```
npm install @aws-cdk/aws-stepfunctions @aws-cdk/aws-apigateway
```

Python

```
python -m pip install aws-cdk.aws-stepfunctions
python -m pip install aws-cdk.aws-apigateway
```

Java

プロジェクトの pom.xml を編集し、既存の <dependencies>コンテナ 内に、以下の依存関係を追加します。

```
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>stepfunctions</artifactId>
            <version>${cdk.version}</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigateway</artifactId>
            <version>${cdk.version}</version>
        </dependency>
```

Mavenは、次回アプリケーションを構築するときに、これらの依存関係を自動的にインストールします。構築するには、mvn compile を使用するか、Java IDEの構築するコマンドを使用します。

C#

```
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.Stepfunctions
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.APIGateway
```

表示されたパッケージは、ツール > NuGet パッケージマネージャー > ソリューションの NuGet パッケージの管理から入手できる Visual Studio NuGet GUI を使用してインストールすることもできます。

モジュールをインストールしたら、次のパッケージをインポートすることで、それらを AWS CDK アプリで使用できます。

TypeScript

```
@aws-cdk/aws-stepfunctions
@aws-cdk/aws-apigateway
```

JavaScript

```
@aws-cdk/aws-stepfunctions
@aws-cdk/aws-apigateway
```

Python

```
aws_cdk.aws_stepfunctions
aws_cdk.aws_apigateway
```

Java

```
software.amazon.awscdk.services.apigateway.StepFunctionsRestApi
software.amazon.awscdk.services.stepfunctions.Pass
software.amazon.awscdk.services.stepfunctions.StateMachine
software.amazon.awscdk.services.stepfunctions.StateMachineType
```

C#

```
Amazon.CDK.AWS.StepFunctions
Amazon.CDK.AWS.APIGateway
```

Go

`stepfunctions-rest-api.go` 内の `import` に次のものを追加します。

```
"github.com/aws/aws-cdk-go/awscdk/awsapigateway"
"github.com/aws/aws-cdk-go/awscdk/awsstepfunctions"
```

ステップ 2: AWS CDK を使用して、同期 Express ステートマシンバックエンド統合で API Gateway REST API を作成する

まず、同期 Express ステートマシンと API Gateway REST API を定義する個々のコードを提示し、それらを AWS CDK アプリにまとめる方法について説明します。次に、これらのリソースを合成してデプロイする方法を見ていきます。

> **Note**
>
> ここで紹介するステートマシンは、Pass ステートのシンプルなステートマシンです。

高速ステートマシンを作成するには

これは、 Pass状態の単純なステートマシンを定義する AWS CDK コードです。

TypeScript

```
const machineDefinition = new stepfunctions.Pass(this, 'PassState', {
    result: {value:"Hello!"},
})

const stateMachine = new stepfunctions.StateMachine(this, 'MyStateMachine', {
    definition: machineDefinition,
    stateMachineType: stepfunctions.StateMachineType.EXPRESS,
});
```

JavaScript

```
const machineDefinition = new sfn.Pass(this, 'PassState', {
    result: {value:"Hello!"},
})

const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
    definition: machineDefinition,
    stateMachineType: stepfunctions.StateMachineType.EXPRESS,
});
```

Python

```
machine_definition = sfn.Pass(self,"PassState",
```

```
                        result = sfn.Result("Hello"))

state_machine = sfn.StateMachine(self, 'MyStateMachine',
        definition = machine_definition,
        state_machine_type = sfn.StateMachineType.EXPRESS)
```

Java

```
Pass machineDefinition = Pass.Builder.create(this, "PassState")
                        .result(Result.fromString("Hello"))
                        .build();

StateMachine stateMachine = StateMachine.Builder.create(this, "MyStateMachine")
                            .definition(machineDefinition)
                            .stateMachineType(StateMachineType.EXPRESS)
                            .build();
```

C#

```
var machineDefinition = new Pass(this, "PassState", new PassProps
{
    Result = Result.FromString("Hello")
});

var stateMachine = new StateMachine(this, "MyStateMachine", new StateMachineProps
{
    Definition = machineDefinition,
    StateMachineType = StateMachineType.EXPRESS
});
```

Go

```
var machineDefinition = awsstepfunctions.NewPass(stack, jsii.String("PassState"),
 &awsstepfunctions.PassProps
{
    Result: awsstepfunctions.NewResult(jsii.String("Hello")),
})

var stateMachine = awsstepfunctions.NewStateMachine(stack,
 jsii.String("StateMachine"), &awsstepfunctions.StateMachineProps
{
    Definition: machineDefinition,
    StateMachineType: awsstepfunctions.StateMachineType_EXPRESS,
```

```
})
```

この短いスニペットで見ることができます:

- PassState という名前のマシン定義 (Pass ステート)。
- ステートマシンの論理名、MyStateMachine。
- マシンの定義がステートマシン定義として使用されます。
- StepFunctionsRestApi は同期高速ステートマシンのみを許可するため、ステートマシンタイプは EXPRESS に設定されています。

StepFunctionsRestApi コンストラクトを使用して API Gateway REST API を作成するには

StepFunctionsRestApi コンストラクトを使用して、必要なアクセス許可とデフォルトの入力/出力マッピングを持つ API Gateway REST API を作成します。

TypeScript

```
const api = new apigateway.StepFunctionsRestApi(this,
  'StepFunctionsRestApi', { stateMachine: stateMachine });
```

JavaScript

```
const api = new apigateway.StepFunctionsRestApi(this,
  'StepFunctionsRestApi', { stateMachine: stateMachine });
```

Python

```
api = apigw.StepFunctionsRestApi(self, "StepFunctionsRestApi",
                             state_machine = state_machine)
```

Java

```
StepFunctionsRestApi api = StepFunctionsRestApi.Builder.create(this,
 "StepFunctionsRestApi")
                            .stateMachine(stateMachine)
                            .build();
```

C#

```
var api = new StepFunctionsRestApi(this, "StepFunctionsRestApi", new
 StepFunctionsRestApiProps
{
    StateMachine = stateMachine
});
```

Go

```
awsapigateway.NewStepFunctionsRestApi(stack, jsii.String("StepFunctionsRestApi"),
 &awsapigateway.StepFunctionsRestApiProps
{
    StateMachine = stateMachine,
})
```

AWS CDK アプリケーションを構築してデプロイするには

作成した AWS CDK プロジェクトで、スタックの定義を含むファイルを以下のコードのように編集します。Step Functions ステートマシンと API Gateway については上述されています。

TypeScript

以下のように `lib/stepfunctions-rest-api-stack.ts` を更新します。

```
import * as cdk from 'aws-cdk-lib';
import * as stepfunctions from 'aws-cdk-lib/aws-stepfunctions'
import * as apigateway from 'aws-cdk-lib/aws-apigateway';


export class StepfunctionsRestApiStack extends cdk.Stack {
    constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const machineDefinition = new stepfunctions.Pass(this, 'PassState', {
        result: {value:"Hello!"},
    });

    const stateMachine = new stepfunctions.StateMachine(this, 'MyStateMachine', {
        definition: machineDefinition,
        stateMachineType: stepfunctions.StateMachineType.EXPRESS,
```

```
    });

    const api = new apigateway.StepFunctionsRestApi(this,
        'StepFunctionsRestApi', { stateMachine: stateMachine });
```

JavaScript

以下のように lib/stepfunctions-rest-api-stack.js を更新します。

```
const cdk = require('@aws-cdk/core');
const stepfunctions = require('@aws-cdk/aws-stepfunctions');
const apigateway = require('@aws-cdk/aws-apigateway');


class StepfunctionsRestApiStack extends cdk.Stack {
    constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const machineDefinition = new stepfunctions.Pass(this, "PassState", {
        result: {value:"Hello!"},
    })

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
        definition: machineDefinition,
        stateMachineType: stepfunctions.StateMachineType.EXPRESS,
    });

    const api = new apigateway.StepFunctionsRestApi(this,
        'StepFunctionsRestApi', { stateMachine: stateMachine });

    }
}

module.exports = { StepStack }
```

Python

以下のように stepfunctions_rest_api/stepfunctions_rest_api_stack.py を更新します。

```
from aws_cdk import App, Stack
from constructs import Construct
from aws_cdk import aws_stepfunctions as sfn
```

```
from aws_cdk import aws_apigateway as apigw

class StepfunctionsRestApiStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)


        machine_definition = sfn.Pass(self,"PassState",
                                result = sfn.Result("Hello"))

        state_machine = sfn.StateMachine(self, 'MyStateMachine',
                definition = machine_definition,
                state_machine_type = sfn.StateMachineType.EXPRESS)

        api = apigw.StepFunctionsRestApi(self,
                    "StepFunctionsRestApi",
                    state_machine = state_machine)
```

Java

以下のように src/main/java/com.myorg/StepfunctionsRestApiStack.java を更新します。

```
package com.myorg;


import software.amazon.awscdk.core.Construct;
import software.amazon.awscdk.core.Stack;
import software.amazon.awscdk.core.StackProps;
import software.amazon.awscdk.services.stepfunctions.Pass;
import software.amazon.awscdk.services.stepfunctions.StateMachine;
import software.amazon.awscdk.services.stepfunctions.StateMachineType;
import software.amazon.awscdk.services.apigateway.StepFunctionsRestApi;

public class StepfunctionsRestApiStack extends Stack {
    public StepfunctionsRestApiStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public StepfunctionsRestApiStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);
```

```
        Pass machineDefinition = Pass.Builder.create(this, "PassState")
                                 .result(Result.fromString("Hello"))
                                 .build();

        StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
                                     .definition(machineDefinition)
                                     .stateMachineType(StateMachineType.EXPRESS)
                                     .build();

        StepFunctionsRestApi api = StepFunctionsRestApi.Builder.create(this,
 "StepFunctionsRestApi")
                                    .stateMachine(stateMachine)
                                    .build();

    }
}
```

C#

以下のように src/StepfunctionsRestApi/StepfunctionsRestApiStack.cs を更新します。

```
using Amazon.CDK;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.APIGateway;

namespace StepfunctionsRestApi
{
    public class StepfunctionsRestApiStack : Stack
    {
        internal StepfunctionsRestApi(Construct scope, string id, IStackProps props
 = null) : base(scope, id, props)
        {
            var machineDefinition = new Pass(this, "PassState", new PassProps
            {
                Result = Result.FromString("Hello")
            });

            var stateMachine = new StateMachine(this, "MyStateMachine", new
 StateMachineProps
            {
```

```
                Definition = machineDefinition,
                StateMachineType = StateMachineType.EXPRESS
            });

            var api = new StepFunctionsRestApi(this, "StepFunctionsRestApi", new
 StepFunctionsRestApiProps
            {
                StateMachine = stateMachine
            });

        }
    }
}
```

Go

以下のように stepfunctions-rest-api.go を更新します。

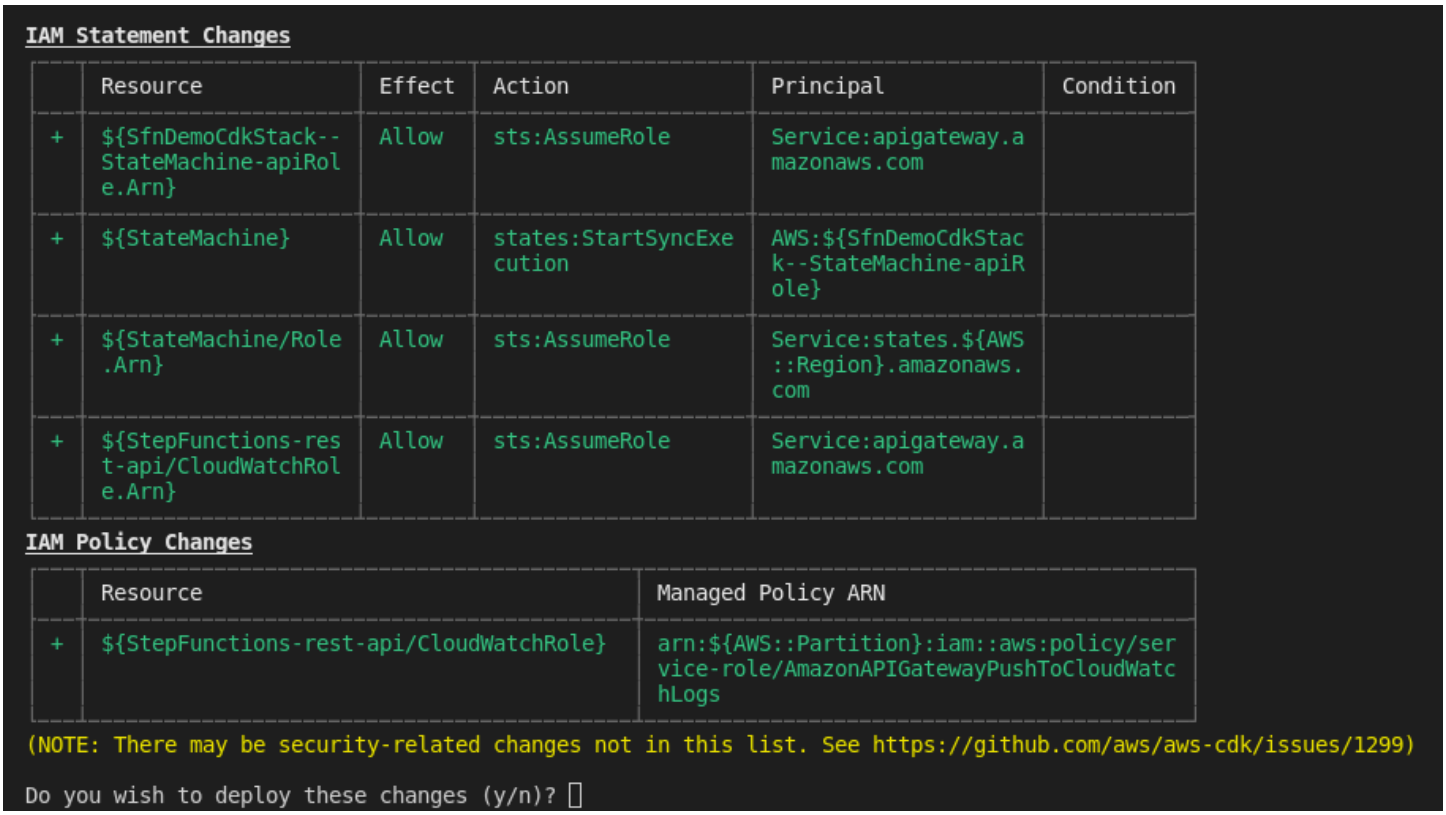
```
package main
import (
    "github.com/aws/aws-cdk-go/awscdk"
    "github.com/aws/aws-cdk-go/awscdk/awsapigateway"
    "github.com/aws/aws-cdk-go/awscdk/awsstepfunctions"
    "github.com/aws/constructs-go/constructs/v3"
    "github.com/aws/jsii-runtime-go"
)


type StepfunctionsRestApiGoStackProps struct {
    awscdk.StackProps
}

func NewStepfunctionsRestApiGoStack(scope constructs.Construct, id string, props
 *StepfunctionsRestApiGoStackProps) awscdk.Stack {
    var sprops awscdk.StackProps
    if props != nil {
        sprops = props.StackProps
    }
    stack := awscdk.NewStack(scope, &id, &sprops)

    // The code that defines your stack goes here
    var machineDefinition = awsstepfunctions.NewPass(stack,
 jsii.String("PassState"), &awsstepfunctions.PassProps
    {
```

```
        Result: awsstepfunctions.NewResult(jsii.String("Hello")),
    })

    var stateMachine = awsstepfunctions.NewStateMachine(stack,
 jsii.String("StateMachine"), &awsstepfunctions.StateMachineProps{
        Definition: machineDefinition,
        StateMachineType: awsstepfunctions.StateMachineType_EXPRESS,
    });

    awsapigateway.NewStepFunctionsRestApi(stack,
 jsii.String("StepFunctionsRestApi"), &awsapigateway.StepFunctionsRestApiProps{
        StateMachine = stateMachine,
    })

    return stack
}

func main() {
    app := awscdk.NewApp(nil)

    NewStepfunctionsRestApiGoStack(app, "StepfunctionsRestApiGoStack",
 &StepfunctionsRestApiGoStackProps{
        awscdk.StackProps{
            Env: env(),
        },
    })

    app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
    // If unspecified, this stack will be "environment-agnostic".
    // Account/Region-dependent features and context lookups will not work, but a
    // single synthesized template can be deployed anywhere.
    //---------------------------------------------------------------------------
    return nil

    // Uncomment if you know exactly what account and region you want to deploy
    // the stack to. This is the recommendation for production stacks.
    //---------------------------------------------------------------------------
    // return &awscdk.Environment{
```

```
    //  Account: jsii.String("123456789012"),
    //  Region:  jsii.String("us-east-1"),
    // }

    // Uncomment to specialize this stack for the AWS Account and Region that are
    // implied by the current CLI configuration. This is recommended for dev
    // stacks.
    //-------------------------------------------------------------------------
    // return &awscdk.Environment{
    //  Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
    //  Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
    // }
}
```

ソースファイルを保存し、cdk synth アプリケーションのメインディレクトリで発行します。はアプリケーション AWS CDK を実行し、そこから AWS CloudFormation テンプレートを合成して、テンプレートを表示します。

Amazon API Gateway と AWS Step Functions ステートマシンを実際に AWS アカウントにデプロイするには、 を発行しますcdk deploy。が生成 AWS CDK した IAM ポリシーを承認するように求められます。作成されるポリシーは、次のようになります。

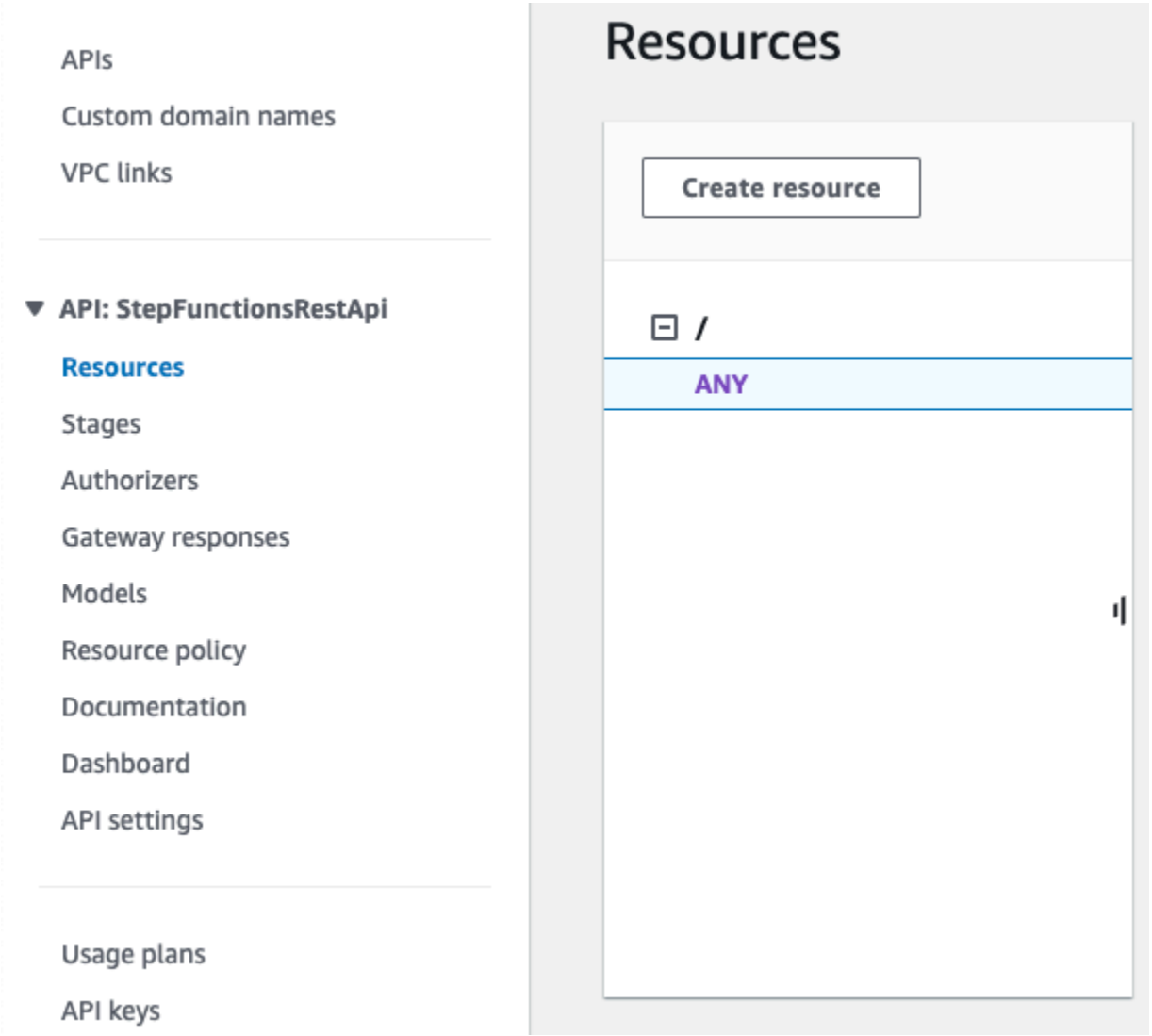


ステップ 3: API Gateway をテストする

同期高速ステートマシンをバックエンド統合として使用して API Gateway REST API を作成したら、API Gateway をテストできます。

API Gateway を使用してデプロイされた API Gateway をテストするには

1. Amazon API Gateway コンソールを開き、サインインします。
2. `StepFunctionsRestApi` という名前の REST API を選択します。
3. [リソース] ペインで、`ANY` メソッドを選択します。



4. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
5. [Method (メソッド)] で [POST] を選択します。

6. [リクエスト本文] には、以下のリクエストパラメータをコピーします。

```
{
    "key": "Hello"
}
```

7. [テスト] を選択します。次の情報が表示されます。

- [リクエスト]: メソッド用に呼び出されたリソースのパスです。
- [ステータス]: 応答の HTTP ステータスコードです。
- [レイテンシー]: 発信者からリクエストを受信してから応答を返すまでの時間です。
- [レスポンス本文] は HTTP レスポンスの本文です。
- [レスポンスヘッダー] は HTTP レスポンスのヘッダーです。
- ログには、このメソッドが API Gateway コンソールの外部で呼び出された場合に書き込まれた、シミュレートされた Amazon CloudWatch Logs エントリが表示されます。

> **Note**
>
> CloudWatch Logs エントリがシミュレートされますが、メソッド呼び出しの結果は実在です。

[レスポンス本文] の出力は、次のようになります。

```
"Hello"
```

> **Tip**
>
> さまざまなメソッドと無効な入力で API Gateway を試して、エラー出力を確認してください。特定のキーを検索するステートマシンを変更し、テスト中に間違ったキーを指定してステートマシンの実行を失敗させ、[レスポンス本文] の出力にエラーメッセージを生成させる場合があります。

cURL を使用して デプロイされた API をテストするには

1. ターミナルウィンドウを開きます。

2. 次の cURL コマンドをコピーしてターミナルウィンドウに貼り付け、<api-id> を API の API ID に、<region> を API がデプロイされているリージョンに置き換えます。

```
curl -X POST\
 'https://<api-id>.execute-api.<region>.amazonaws.com/prod' \
 -d '{"key":"Hello"}' \
 -H 'Content-Type: application/json'
```

[レスポンス本文] の出力は、次のようになります。

```
"Hello"
```

> **Tip**
>
> さまざまなメソッドと無効な入力で API Gateway を試して、エラー出力を確認してください。特定のキーを検索するステートマシンを変更し、テスト中に間違ったキーを指定してステートマシンの実行を失敗させ、[レスポンス本文] の出力にエラーメッセージを生成させる場合があります。

ステップ 4: クリーンアップする

API Gateway の試行が完了したら、AWS CDK を使用してステートマシンと API Gateway の両方をティアダウンできます。アプリケーションのメインディレクトリで `cdk destroy` を発行します。

AWS Step Functions Python 用データサイエンス SDK

AWS Step Functions データサイエンス SDK はデータサイエンティスト向けのオープンソースライブラリです。この SDK を使用すると、と Step Functions SageMaker を使用して機械学習モデルを処理および公開するワークフローを作成できます。また、 AWS サービスを個別にプロビジョニングして統合しなくても、 AWS 大規模なインフラストラクチャを調整するマルチステップの機械学習ワークフローを Python で作成することもできます。

AWS Step Functions データサイエンス SDK は、Step Functions ワークフローを作成して呼び出すことができる Python API を提供します。これらのワークフローは、Python および Jupyter ノートブックで直接管理および実行できます。

AWS Step Functions Data Science SDK では、本番環境ですぐに使えるワークフローを Python で直接作成できるだけでなく、そのワークフローをコピーし、新しいオプションを試して、改良したワークフローを本番環境に導入することもできます。

AWS Step Functions データサイエンス SDK の詳細については、以下を参照してください。

- GitHub のプロジェクト
- SDK ドキュメント
- 以下のサンプルノートブックは、SageMaker コンソールおよび関連プロジェクトの Jupyter ノートブックインスタンスで使用できます。GitHub
  - `hello_world_workflow.ipynb`
  - `machine_learning_workflow_abalone.ipynb`
  - `training_pipeline_pytorch_mnist.ipynb`

Terraform を使用する、ステートマシンのデプロイ

Terraform by HashiCorp は、Infrastructure as Code (IaC) を使用してアプリケーションを構築するためのフレームワークです。Terraform では、ステートマシンを作成したり、インフラストラクチャデプロイのプレビューや再利用可能なテンプレートの作成などの機能を使用したりできます。Terraform テンプレートを使用すると、コードが小さなチャンクに分割されるため、コードを保守して再利用しやすくなります。

Terraform に精通していれば、Terraform でステートマシンを作成してデプロイするためのモデルとして、このトピックで説明する開発ライフサイクルをたどることができます。Terraform に慣れていない場合は、まず「AWS Introduction to Terraform on 」というワークショップを完了して Terraform をよく知っておくことをお勧めします。

> **Tip**
>
> Terraform を使用して構築されたステートマシンの例を AWS アカウント にデプロイするには、「AWS Step Functions ワークショップ」の「コードとしてのインフラストラクチャによるステートマシンの管理」モジュールを参照してください。

このトピックの内容

- 前提条件
- Terraform を使用したステートマシンの開発ライフサイクル

- ステートマシンの IAM ロールとポリシー

前提条件

開始するには、以下の前提条件を満たしていることを確認する必要があります。

- Terraform をマシンにインストールします。Terraform のインストールについて詳しくは、「Terraform をインストールする」を参照してください。
- マシンに Step Functions Local をインストールします。Step Functions Local を使用するには、Step Functions Local の Docker イメージをインストールすることをお勧めします。詳細については「ステートマシンのローカルテスト」を参照してください。
- AWS SAM CLI をインストールします。インストールの情報については、「AWS Serverless Application Model デベロッパーガイド」の「AWS SAM SDK for JavaScript のインストール」を参照してください。
- AWS Toolkit for Visual Studio Code をインストールすると、ステートマシンのワークフロー図が表示されます。インストールの情報については、「AWS Toolkit for Visual Studio Code ユーザーガイド」の「AWS Toolkit for Visual Studio Code のインストール」を参照してください。

Terraform を使用したステートマシンの開発ライフサイクル

以下の手順では、Step Functions コンソールの Workflow Studio を使用してビルドしたステートマシンプロトタイプを、Terraform と AWS Toolkit for Visual Studio Code によるローカル開発の開始点として使用する方法について説明します。

Terraform を使ったステートマシン開発について説明してベストプラクティスの詳細を示す完全な例を見るには、「Best Practices for Writing Step Functions Terraform Projects」を参照してください。

Terraform を使ってステートマシンの開発ライフサイクルを開始するには

1. 次のコマンドを使用して、新しい Terraform プロジェクトをブートストラップします。

   ```
   terraform init
   ```

2. Step Functions コンソールを開き、ステートマシンのプロトタイプを作成します。
3. Workflow Studio で、以下を実行します。

   a. ワークフロープロトタイプを作成します。

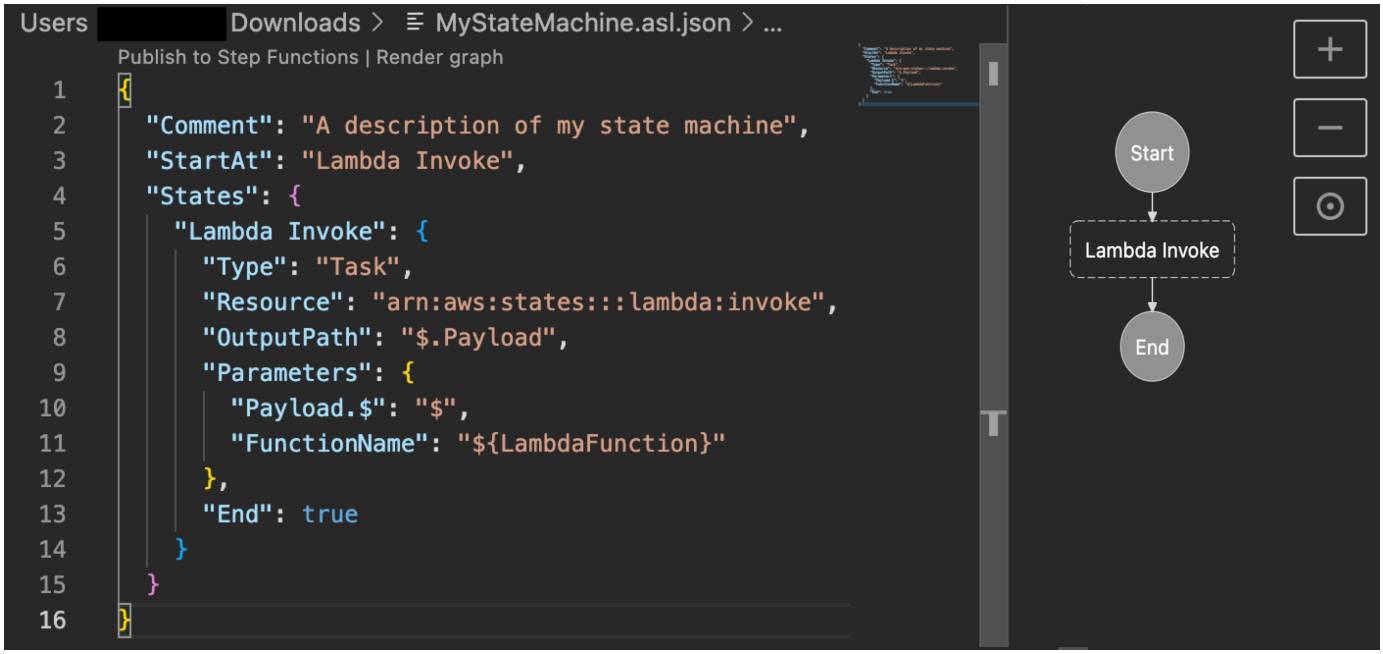
b. ワークフローの Amazon States Language (ASL) 定義をエクスポートします。これを行うには、[インポート / エクスポート] ドロップダウンリストを選択して、[JSON 定義をエクスポート] を選択します。

4. エクスポートした ASL 定義をプロジェクトディレクトリに保存します。

   エクスポートした ASL 定義を、`templatefile` 関数を使用する `aws_sfn_state_machine` Terraform リソースに入力パラメータとして渡します。この関数は、エクスポートされた ASL 定義と変数置換を渡す定義フィールド内で使用されます。

   > **Tip**
   >
   > ASL 定義ファイルには長いテキストブロックが含まれることがあるため、インライン EOF メソッドを避けることをお勧めします。これにより、ステートマシン定義へのパラメータ代入が容易になります。

5. (オプション) IDE 内の ASL 定義を更新し、AWS Toolkit for Visual Studio Code を使用して変更内容を視覚化します。



   定義を継続的にエクスポートしてプロジェクトにリファクタリングすることを避けるために、IDE でローカルに更新を行い、その更新を Git で追跡することをお勧めします。

6. Step Functions Local を使用してワークフローをテストします。

> **Tip**
>
> AWS SAM CLI Local を使用して、ステートマシンで Lambda 関数および API Gateway API とのサービス統合をローカルでテストすることもできます。

7. ステートマシンをデプロイする前に、ステートマシンやその他の AWS リソースをプレビューしてください。これを行うには、次のコマンドを実行します。

```
terraform plan
```

8. 以下のコマンドを使用して、ローカル環境または CI/CD パイプラインからステートマシンをデプロイします。

```
terraform apply
```

9. (オプション) 以下のコマンドを使用して、リソースをクリーンアップし、ステートマシンを削除します。

```
terraform destroy
```

ステートマシンの IAM ロールとポリシー

Terraform サービス統合ポリシーを使用して、必要な IAM アクセス許可 (Lambda 関数を呼び出す許可など) をステートマシンに追加します。ロールやポリシーを明示的に定義し、ステートマシンに関連付けることもできます。

IAM ポリシーの以下の例では、*myFunction* という Lambda 関数を呼び出すためのアクセス権をステートマシンに付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myFunction"
    }
```

```
  ]
}
```

Terraform でステートマシンの IAM ポリシーを定義するときにもこの [aws_iam_policy_document](#) データソースを使用することをお勧めします。これにより、ポリシーの形式に誤りがないかどうかを確認し、リソースを変数に置き換えることができます。

IAM ポリシーの以下の例では、aws_iam_policy_document データソースを使用して、*myFunction* という Lambda 関数を呼び出すためのアクセス権をステートマシンに付与します。

```
data "aws_iam_policy_document" "state_machine_role_policy" {

  statement {
    effect = "Allow"

    actions = [
      "lambda:InvokeFunction"
    ]

    resources = ["${aws_lambda_function.[[myFunction]].arn}:*"]
  }

}
```

> **Tip**
>
> Terraform でデプロイされた、より高度な AWS アーキテクチャパターンを表示するには、「Terraform examples at Serverless Land Workflows Collection」を参照してください。

テストとデバッグ

Step Functions には、ステートマシンをテストおよびデバッグするためのさまざまな方法があります。例えば、ステートマシンをコンソールでテストしてデバッグしたり、TestState API を使用して個々のステートをテストしたり、Step Functions Local を使用してステートマシンをローカルでテストしたりできます。

TestState API を使用して、1 つのステートの定義を指定して実行します。ステートマシンを作成したり、既存のステートマシンを更新したりしなくても、単一のステートをテストできます。

Step Functions Local は Step Functions のダウンロード可能なバージョンです。これを使用すると、独自の開発環境で実行されている Step Functions のバージョンを使用してアプリケーションを開発およびテストできます。Step Functions Local では、ステートマシンを実行して、その入出力データフロー、サポートされているサービスとの統合などをローカル開発環境でテストできます。

トピック

- TestState API を使用して状態をテストする
- ステートマシンのローカルテスト

TestState API を使用して状態をテストする

TestState API は、単一の状態の定義を受け入れて実行します。ステートマシンを作成したり、既存のステートマシンを更新したりしなくても、ステートをテストできます。

TestState API を使用して、以下をテストできます。

- ステートの入出力処理データフロー。
- 他の AWS のサービス リクエストやレスポンスとAWS のサービス の統合
- HTTP タスクのリクエストとレスポンス

ステートをテストするには、Step Functions コンソール、AWS Command Line Interface (AWS CLI)、または SDK を使用することもできます。

`TestState` API は IAM ロールを想定しており、そのロールにはステートがアクセスするリソースに必要な IAM アクセス許可が含まれている必要があります。ステートに必要なアクセス許可については、「IAM TestState API を使用するための アクセス許可」を参照してください。

トピック

- TestState API の使用に関する考慮事項
- TestState API での検査レベルの使用
- IAM TestState API を使用するための アクセス許可
- ステートのテスト (コンソール)
- AWS CLI を使用してステートをテストする
- 入力データと出力データフローのテストとデバッグ

TestState API の使用に関する考慮事項

TestState API を使用すると、一度に 1 つの状態のみをテストできます。テストできるステートは次のとおりです。

- アクティビティを除くすべてのタスクタイプ
- パス
- 待機
- 選択
- 成功
- 失敗

TestState API を使用する場合は、次の考慮事項に留意してください。

- TestState API には、以下のサポートは含まれていません。
  - 以下のリソースタイプを使用する タスク ステート:
    - アクティビティ
    - .sync または .waitForTaskToken のサービス統合パターン
  - 並行 ステート
  - マッピング ステート
- テストは最大 5 分間実行できます。テストがこの期間を超えると、States.Timeout エラーで失敗します。

TestState API での検査レベルの使用

TestState API を使用して状態をテストするには、その状態の定義を指定します。その後、テストは出力を返します。ステートごとに、テスト結果に表示したい詳細の量を指定できます。これらの情報は、テストしているステートに関する追加情報を提供します。例えば、`InputPath` または `ResultPath` ステート内で入出力データ処理フィルターを使用したことがある場合は、中間データ処理結果と最終データ処理結果を表示できます。

Step Functions には以下のレベルがあり、表示したい詳細を指定できます。

- INFO
- DEBUG
- TRACE

これらのレベルはすべて、`status` および `nextState` フィールドも返します。`status` は、ステート実行のステータスを示します。例えば、`SUCCEEDED`、`FAILED`、`RETRIABLE`、および `CAUGHT_ERROR` などです。`nextState` は、次に遷移するステートの名前を示します。定義に次のステートを定義していない場合、このフィールドは空の値を返します。

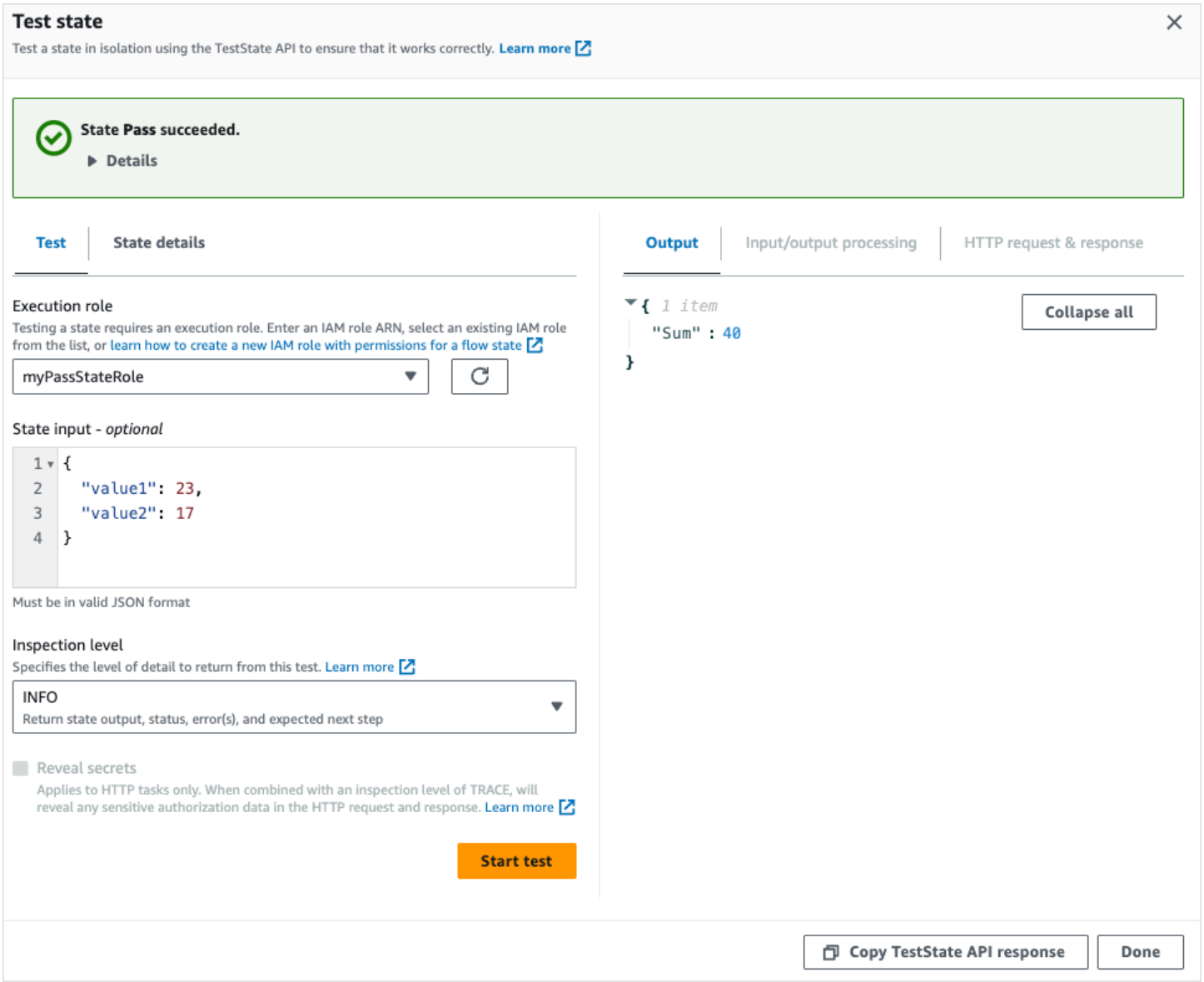
Step Functions コンソールおよび AWS CLI でこれらの検査レベルを使用してステートをテストする方法については、「ステートのテスト (コンソール)」および「AWS CLI を使用してステートをテストする」を参照してください。

INFO inspectionLevel

テストが成功すると、このレベルにはステート出力が表示されます。テストが失敗した場合、このレベルにはエラー出力が表示されます。レベルを指定しない場合、デフォルトで Step Functions は [検査レベル] を [INFO] に設定します。

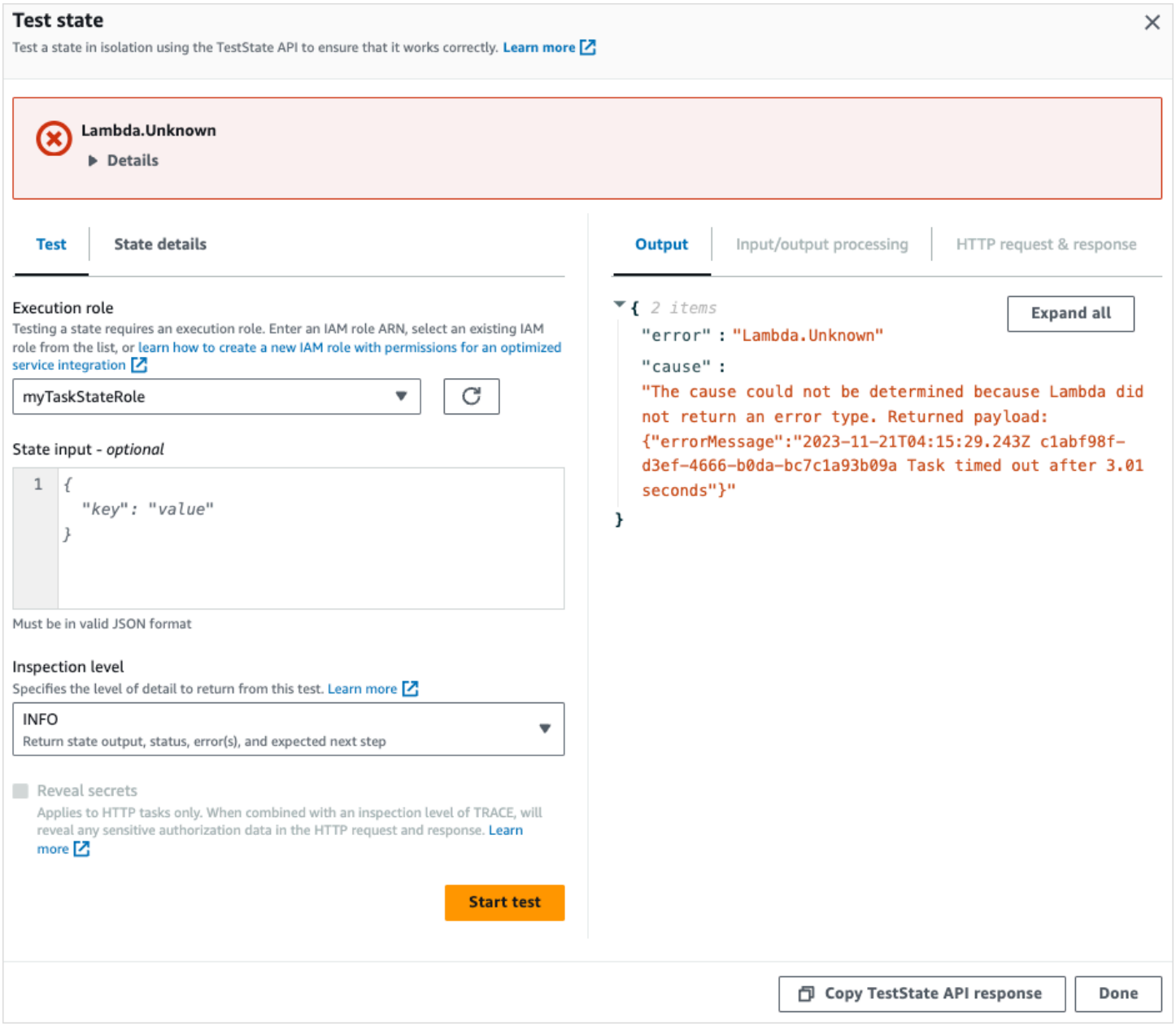
INFO レベルで成功したテストの例

次の図は、成功のテストを示しています。このステートの [検査レベル] は [INFO] に設定され、ステートの出力が [出力] タブに表示されます。



INFO レベルで失敗したテストの例

以下の画像は、[検査レベル] が [INFO] に設定されている場合に、タスクステートに対して失敗したテストを示しています。[出力] タブには、エラー名とそのエラーの原因の詳細な説明を含むエラー出力が表示されます。



DEBUG inspectionLevel

テストが成功すると、このレベルにはステート出力と入出力データ処理の結果が表示されます。

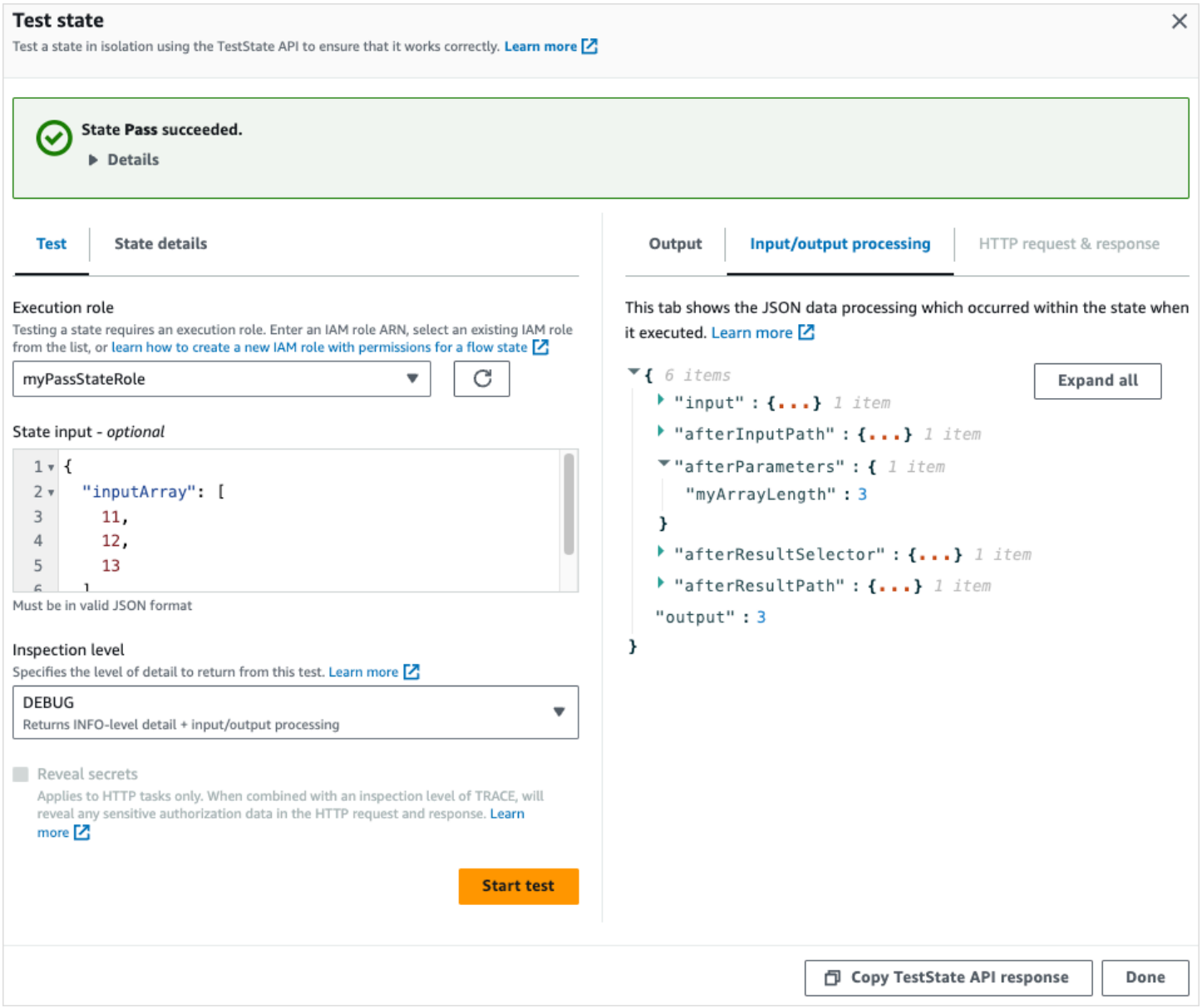
テストが失敗した場合、このレベルにはエラー出力が表示されます。このレベルには、障害発生時点までの中間データ処理結果が表示されます。例えば、Lambda 関数を呼び出すタスクステートをテストしたとします。タスクステートに、InputPath、パラメータ、ResultPath、および OutputPath フィルターを適用したとします。呼び出しが失敗したとします。この場合、DEBUG レベルにはフィルターの適用に基づくデータ処理結果が次の順序で表示されます。

- `input` — 未加工のステート入力
- `afterInputPath` — Step Functions が `InputPath` フィルターを適用した後の入力。
- `afterParameters` — Step Functions が `Parameters` フィルターを適用した後の有効な入力。

このレベルで利用できる診断情報は、定義したサービス統合や入出力データ処理フローに関連する問題のトラブルシューティングに役立ちます。

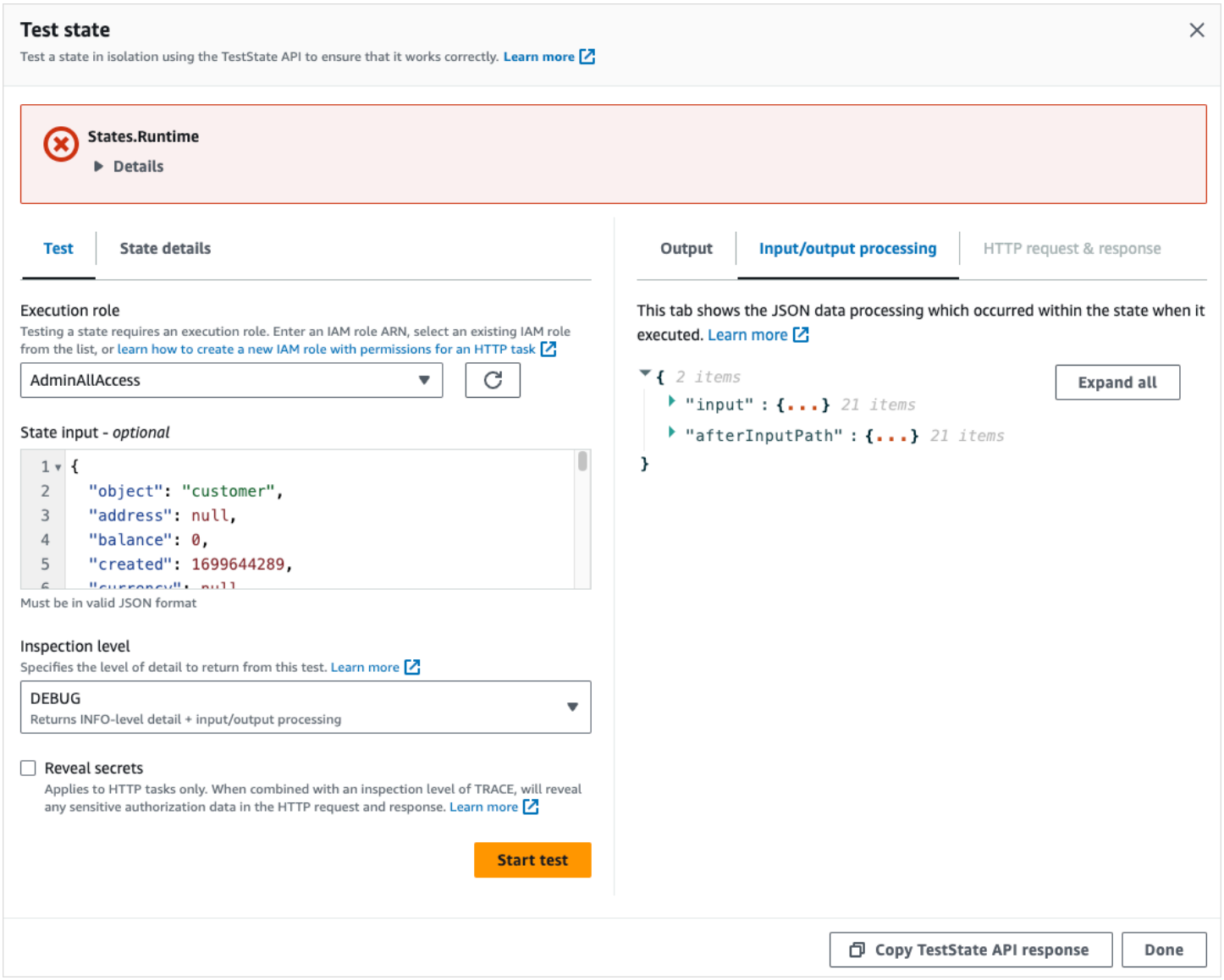
DEBUG レベルで成功したテストの例

次の図は、成功したテストのパスステートを示しています。このステートの [検査レベル] は [DEBUG] に設定されています。以下の画像の [入出力処理] タブには、このステートに提供された入力に `Parameters` を適用した結果が表示されます。



DEBUG レベルで失敗したテストの例

以下の画像は、[検査レベル] が [DEBUG] に設定されている場合にタスクステートに対して失敗したテストを示しています。以下の画像の [入出力処理タブ] には、障害発生時点までのステートの入出力データ処理結果が表示されます。



TRACE inspectionLevel

Step Functions は HTTP タスクをテストするための [TRACE] レベルを提供します。このレベルは、サードパーティー API が返す Step Functions が作成する HTTP リクエストとレスポンスに関する情報を返します。レスポンスには、ヘッダーやリクエスト本文などの情報が含まれる場合があります。さらに、このレベルでの入出力データ処理のステート出力と結果も表示できます。

テストが失敗した場合、このレベルにはエラー出力が表示されます。

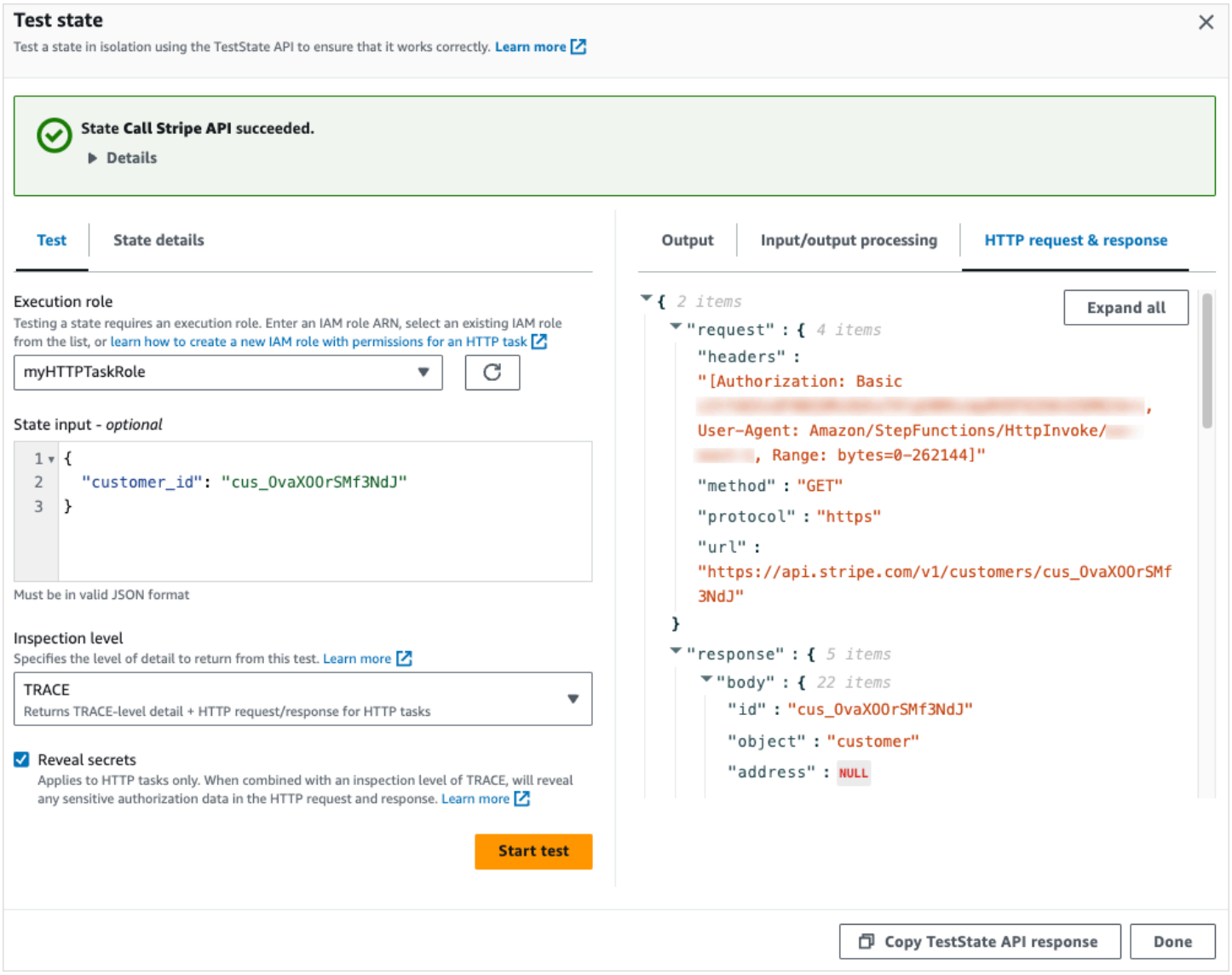
このレベルは HTTP タスクにのみ適用されます。このレベルを他のステートタイプに使用すると、Step Functions はエラーをスローします。

検査レベルを TRACE に設定すると、EventBridge 接続 に含まれるシークレットを表示することもできます。これを行うには、 TestState API `true`で `revealSecrets`パラメータを に設定する必要があります。さらに、 TestState API を呼び出すIAMユーザーに `states:RevealSecrets`アクションのアクセス許可があることを確認する必要があります。`states:RevealSecrets` アクセス許可を付与する IAM ポリシーの例については、「IAM TestState API を使用するための アクセス許可」を参照してください。このアクセス許可がない場合、Step Functions はアクセス拒否エラーをスローします。

`revealSecrets` パラメータを `false` に設定すると、Step Functions はHTTP リクエストとレスポンスのデータに含まれるすべてのシークレットを省略します。

TRACE レベルで成功したテストの例

次の図は、成功した HTTP タスクのテストを示しています。このステートの [検査レベル] は [TRACE] に設定されています。以下の画像の [HTTP リクエストおよびレスポンス] タブには、サードパーティー API コールの結果が表示されます。



IAM TestState API を使用するための アクセス許可

TestState API を呼び出す IAM ユーザーには、states:TestState および iam:PassRole アクションを実行するアクセス許可が必要です。さらに、RevealSecrets パラメータを true に設定する場合は、IAM ユーザーが states:RevealSecrets アクションを実行するアクセス許可を持っていることを確認する必要があります。このアクセス許可がない場合、Step Functions はアクセス拒否エラーをスローします。

また、IAM ステートがアクセスしているリソースに必要なアクセス許可が実行ロールに含まれていることも確認する必要があります。ステートが必要とするアクセス許可については、「実行ロールの管理」を参照してください。

次の IAM ポリシー例では、states:TestState、iam:PassRole、および states:RevealSecrets アクセス許可を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:TestState",
        "states:RevealSecrets",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

ステートのテスト (コンソール)

コンソールでステートをテストし、ステートの出力または入出力データの処理フローを確認できます。HTTP タスクでは、未加工の HTTP リクエストとレスポンスをテストできます。

ステートをテストするには

1. Step Functions コンソールを開きます。
2. [ステートマシンの作成] を選択してステートマシンの作成を開始するか、既存のステートマシンを選択します。
3. Workflow Studio の デザインモード で、テストするステートを選択します。
4. Workflow Studio の Inspector パネルで [ステートをテスト] を選択します。
5. [ステートをテスト] ダイアログボックスで、次の操作を行います。

    a. [実行ロール] では、ステートをテストする実行ロールを選択します。テストするステートに必要な IAM アクセス許可があることを確認してください。

    b. (オプション) 選択したステートをテストするために必要な JSON 入力を入力します。

    c. [検査レベル] では、表示したい値に基づいて以下のオプションのいずれかを選択します。

    - INFO — テストが成功すると、[出力] タブにステート出力が表示されます。テストが失敗した場合、[INFO] はエラー名とそのエラーの原因の詳細な説明を含むエラー出力を

表示します。レベルを選択しない場合、Step Functions はデフォルトで [検査レベル] を [INFO] に設定します。

- DEBUG — テストが成功した場合、ステート出力と入出力データ処理の結果を表示します。テストが失敗した場合、[DEBUG] はエラー名とそのエラーの原因の詳細な説明を含むエラー出力を表示します。
- TRACE — 未加工の HTTP リクエストとレスポンスを表示し、ヘッダー、クエリパラメータ、その他の API 固有の詳細を確認するのに役立ちます。このオプションは HTTP タスクでのみ使用できます。

  オプションで [シークレットを公開] を選択することもできます。この設定を [TRACE] と組み合わせると、API キーなど、EventBridge 接続によって挿入される機密データを表示できます。コンソールへのアクセスに使用する IAM ユーザー ID には、`states:RevealSecrets` アクションを実行するアクセス許可が必要です。このアクセス許可がない場合、Step Functions はテストの開始時にアクセス拒否エラーをスローします。`states:RevealSecrets` アクセス許可を付与する IAM ポリシーの例については、「IAM TestState API を使用するための アクセス許可」を参照してください。

d. [テストを開始] を選択します。

AWS CLI を使用してステートをテストする

サポートされている状態は、 の TestState API を使用してテストできますAWS CLI。この API はステートの定義を受け入れて実行します。

ステートごとに、テスト結果に表示したい詳細の量を指定できます。これらの詳細から、入出力データ処理結果や HTTP リクエスト/レスポンス情報など、ステートの実行に関する追加情報が得られます。次の例では、 TestState API に指定できるさまざまな検査レベルを示しています。`######`のテキストを、リソース固有の情報に必ず置き換えてください。

このセクションには、Step Functions が AWS CLI で提供するさまざまな検査レベルの使用方法を示す次の例が含まれています。

- INFO inspectionLevel の使用
- DEBUG inspectionLevel の使用
- DEBUG inspectionLevel の使用
- で jq ユーティリティを使用して TestState API が返す HTTP レスポンスをAWS CLIフィルタリングして出力する

例 1: INFO inspectionLevel を使用して Choice ステートをテストする

の INFO inspectionLevel を使用して状態をテストするには AWS CLI、次の例に示すように test-state コマンドを実行します。

```
aws stepfunctions test-state \
    --definition '{"Type": "Choice", "Choices": [{"Variable": "$.number",
 "NumericEquals": 1, "Next": "Equals 1"}, {"Variable": "$.number", "NumericEquals": 2,
 "Next": "Equals 2"}], "Default": "No Match"}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --input '{"number": 2}'
```

この例では、Choice ステートを使用して、指定した数値入力に基づいてステートの実行パスを決定します。レベルを設定しない場合、 Step Functions はデフォルトで inspectionLevel を INFO に設定します。

Step Functions は次の出力を返します。

```
{
    "output": "{\"number\": 2}",
    "nextState": "Equals 2",
    "status": "SUCCEEDED"
}
```

例 2: DEBUG InspectionLevel を使用して Pass ステートの入出力データ処理をデバッグする

の DEBUG inspectionLevel を使用して状態をテストするには AWS CLI、次の例に示すように test-state コマンドを実行します。

```
aws stepfunctions test-state \
    --definition '{"Type": "Pass", "InputPath": "$.payload", "Parameters": {"data": 1},
 "ResultPath": "$.result", "OutputPath": "$.result.data", "Next": "Another State"}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --input '{"payload": {"foo": "bar"}}' \
    --inspection-level DEBUG
```

この例では、パス ステートを使用して、Step Functions が入出力データ処理フィルターを使用して入力 JSON データをフィルタリングおよび操作する方法を示しています。この例では、InputPath、#####、ResultPath、および OutputPath フィルターを使用しています。

Step Functions は次の出力を返します。

```
{
    "output": "1",
    "inspectionData": {
        "input": "{\"payload\": {\"foo\": \"bar\"}}",
        "afterInputPath": "{\"foo\":\"bar\"}",
        "afterParameters": "{\"data\":1}",
        "afterResultSelector": "{\"data\":1}",
        "afterResultPath": "{\"payload\":{\"foo\":\"bar\"},\"result\":{\"data\":1}}"
    },
    "nextState": "Another State",
    "status": "SUCCEEDED"
}
```

例 3: TRACE InspectionLevel と RevealSecrets を使用して、サードパーティー API に送信された HTTP リクエストを検査する

TRACE inspectionLevel と の revealSecrets パラメータを使用して HTTP タスクをテストするには AWS CLI、次の例に示すように `test-state` コマンドを実行します。

```
aws stepfunctions test-state \
    --definition '{"Type": "Task", "Resource": "arn:aws:states:::http:invoke",
 "Parameters": {"Method": "GET", "Authentication": {"ConnectionArn":
 "arn:aws:events:us-
east-1:123456789012:connection/MyConnection/0000000-0000-0000-0000-000000000000"},
 "ApiEndpoint": "https://httpbin.org/get", "Headers": {"definitionHeader": "h1"},
 "RequestBody": {"message": "Hello from Step Functions!"}, "QueryParameters":
 {"queryParam": "q1"}}, "End": true}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --inspection-level TRACE \
    --reveal-secrets
```

この例では、HTTP タスクが指定されたサードパーティー API である `https://httpbin.org/` を呼び出しているかどうかをテストします。API コールの HTTP リクエストとレスポンスデータも表示されます。

```
{
    "output": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17 GMT\"],
\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],\"server\":
[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],\"content-
type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":
```

\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization
\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
    "inspectionData": {
        "input": "{}",
        "afterInputPath": "{}",
        "afterParameters": "{\"Method\":\"GET\",\"Authentication\":{\"ConnectionArn
\":\"arn:aws:events:us-east-1:123456789012:connection/foo/a59c10f0-a315-4c1f-
be6a-559b9a0c6250\"},\"ApiEndpoint\":\"https://httpbin.org/get\",\"Headers\":
{\"definitionHeader\":\"h1\"},\"RequestBody\":{\"message\":\"Hello from Step Functions!
\"},\"QueryParameters\":{\"queryParam\":\"q1\"}}",
        "result": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17 GMT\"],
\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],\"server\":
[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],\"content-
type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":
\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization
\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "afterResultSelector": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023
 00:06:17 GMT\"],\"access-control-allow-origin\":[\"*\"],\"content-length\":
[\"620\"],\"server\":[\"gunicorn/19.9.0\"],\"access-control-allow-credentials
\":[\"true\"],\"content-type\":[\"application/json\"]},\"ResponseBody\":{\"args
\":{\"QueryParam1\":\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":
{\"Authorization\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json;
 charset=UTF-8\",\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",
\"Host\":\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked
\",\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "afterResultPath": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17
 GMT\"],\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],
\"server\":[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],
\"content-type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":

```
\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization\":
\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "request": {
            "protocol": "https",
            "method": "GET",
            "url": "https://httpbin.org/get?
queryParam=q1&QueryParam1=QueryParamValue1",
            "headers": "[definitionHeader: h1, Authorization: Basic XXXXXXXX,
 CustomHeader1: CustomHeaderValue1, User-Agent: Amazon|StepFunctions|HttpInvoke|us-
east-1, Range: bytes=0-262144]",
            "body": "{\"message\":\"Hello from Step Functions!\",\"BodyKey1\":
\"BodyValue1\"}"
        },
        "response": {
            "protocol": "https",
            "statusCode": "200",
            "statusMessage": "OK",
            "headers": "[date: Tue, 21 Nov 2023 00:06:17 GMT, content-type:
 application/json, content-length: 620, server: gunicorn/19.9.0, access-control-allow-
origin: *, access-control-allow-credentials: true]",
            "body": "{\n  \"args\": {\n    \"QueryParam1\": \"QueryParamValue1\", \n
    \"queryParam\": \"q1\"\n  }, \n  \"headers\": {\n    \"Authorization\": \"Basic
 XXXXXXXX\", \n    \"Content-Type\": \"application/json; charset=UTF-8\", \n
 \"Customheader1\": \"CustomHeaderValue1\", \n    \"Definitionheader\": \"h1\", \n
   \"Host\": \"httpbin.org\", \n    \"Range\": \"bytes=0-262144\", \n    \"Transfer-
Encoding\": \"chunked\", \n    \"User-Agent\": \"Amazon|StepFunctions|HttpInvoke|us-
east-1\", \n    \"X-Amzn-Trace-Id\": \"Root=1-0000000-0000-0000-0000-000000000000\"\n
  }, \n  \"origin\": \"12.34.567.891\", \n  \"url\": \"https://httpbin.org/get?
queryParam=q1&QueryParam1=QueryParamValue1\"\n}\n"
        }
    },
    "status": "SUCCEEDED"
}
```

例 4: jq ユーティリティを使用して TestState API が返すレスポンスをフィルタリングして出力する

TestState API は、JSON データをエスケープされた文字列としてレスポンスに返します。次のAWS CLI例では、例 3 を拡張し、 jqユーティリティを使用して、 TestState API が返す HTTP レスポンスを人間が読める形式でフィルタリングして出力します。jq とそのインストール手順については、「」の「jq」を参照してくださいGitHub。

```
aws stepfunctions test-state \
    --definition '{"Type": "Task", "Resource": "arn:aws:states:::http:invoke",
 "Parameters": {"Method": "GET", "Authentication": {"ConnectionArn":
 "arn:aws:events:us-
east-1:123456789012:connection/MyConnection/0000000-0000-0000-0000-000000000000"},
 "ApiEndpoint": "https://httpbin.org/get", "Headers": {"definitionHeader": "h1"},
 "RequestBody": {"message": "Hello from Step Functions!"}, "QueryParameters":
 {"queryParam": "q1"}}, "End": true}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --inspection-level TRACE \
    --reveal-secrets \
    | jq '.inspectionData.response.body | fromjson'
```

次の例は、人間が読める形式で返される出力を示しています。

```
{
  "args": {
    "QueryParam1": "QueryParamValue1",
    "queryParam": "q1"
  },
  "headers": {
    "Authorization": "Basic XXXXXXXX",
    "Content-Type": "application/json; charset=UTF-8",
    "Customheader1": "CustomHeaderValue1",
    "Definitionheader": "h1",
    "Host": "httpbin.org",
    "Range": "bytes=0-262144",
    "Transfer-Encoding": "chunked",
    "User-Agent": "Amazon|StepFunctions|HttpInvoke|us-east-1",
    "X-Amzn-Trace-Id": "Root=1-0000000-0000-0000-0000-000000000000"
  },
  "origin": "12.34.567.891",
  "url": "https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1"
```

```
}
```

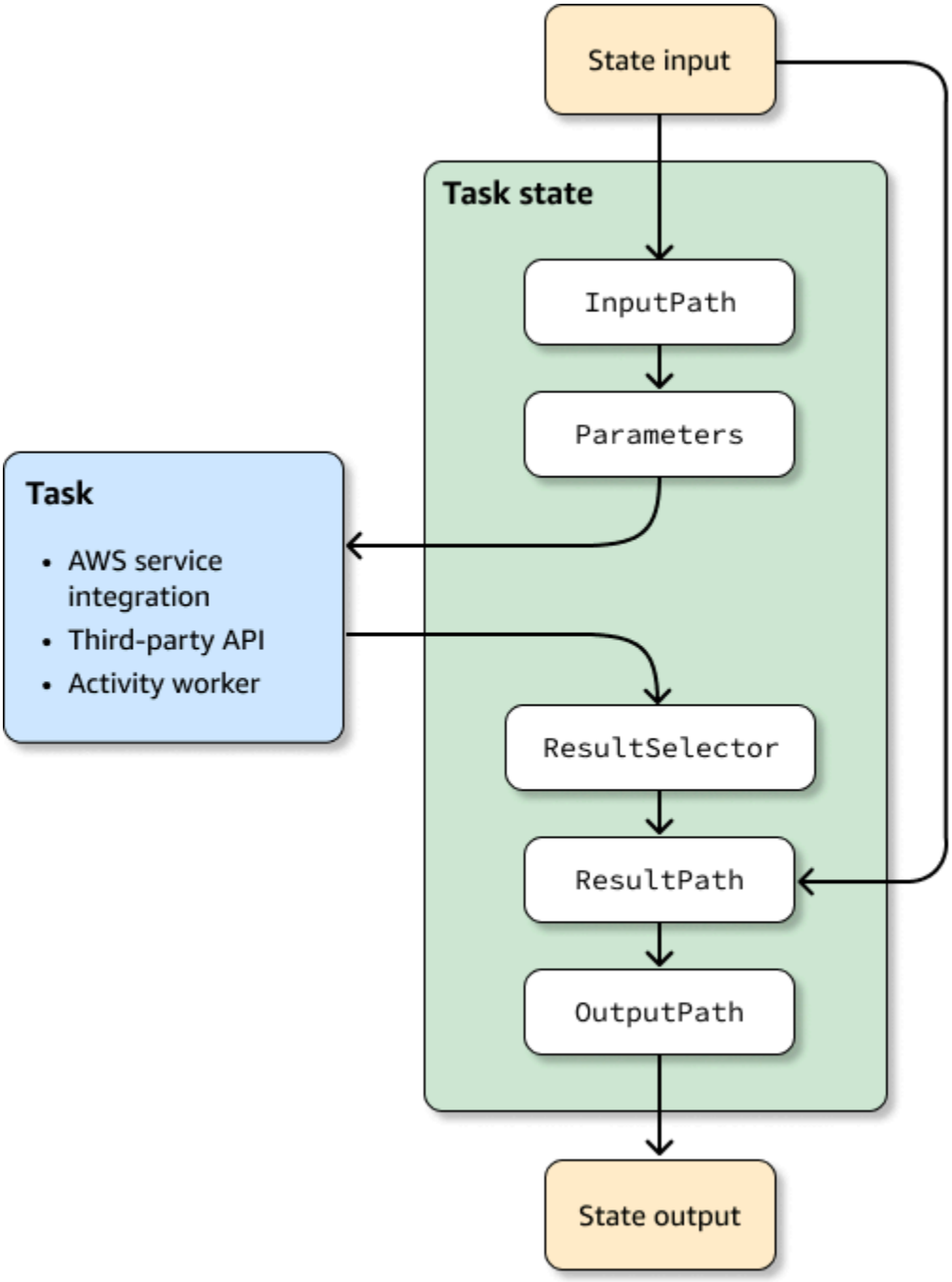
入力データと出力データフローのテストとデバッグ

TestState API は、ワークフローを移動するデータのテストとデバッグに役立ちます。このセクションでは、いくつかの重要な概念と、 TestState この目的で を使用する方法について説明します。

主要なコンセプト

Step Functions では、ステートマシンのステートを通過する JSON データをフィルタリングして操作するプロセスを入出力処理と呼びます。この仕組みについては、「Step Functions の入出力処理」を参照してください。

Amazon ステートメント言語 (ASL) のすべてのステートタイプ (Task、Parallel、Map、Pass、Wait、Choice、Succeed、Fail) は、それらを通過する JSON データをフィルタリングし、操作するための共通のフィールドセットを共有しています。これらのフィールドは、InputPath、パラメータ、ResultSelector、ResultPath、および OutputPath です。各フィールドのサポートはステートによって異なります。実行時には、Step Functions は各フィールドを特定の順序で適用します。以下の図は、これらのフィールドがタスクステート内のデータに適用される順序を示しています。



以下のリストは、図に示されている入力処理フィールドと出力処理フィールドの適用順序を示しています。

1. ステート入力は、前のステートから現在のステートに渡された JSON データです。
2. InputPath は未加工のステート入力の一部をフィルタリングします。
3. パラメータ はタスクに渡す値のセットを設定します。
4. タスクは作業を実行し、結果を返します。

5. ResultSelector はタスク結果から除外する値のセットを選択します。
6. ResultPath は結果を未加工のステート入力と結合するか、結果をそれに置き換えます。
7. OutputPath は出力の一部をフィルタリングして次のステートに渡します。
8. ステート出力は、現在のステートから次のステートに渡される JSON データです。

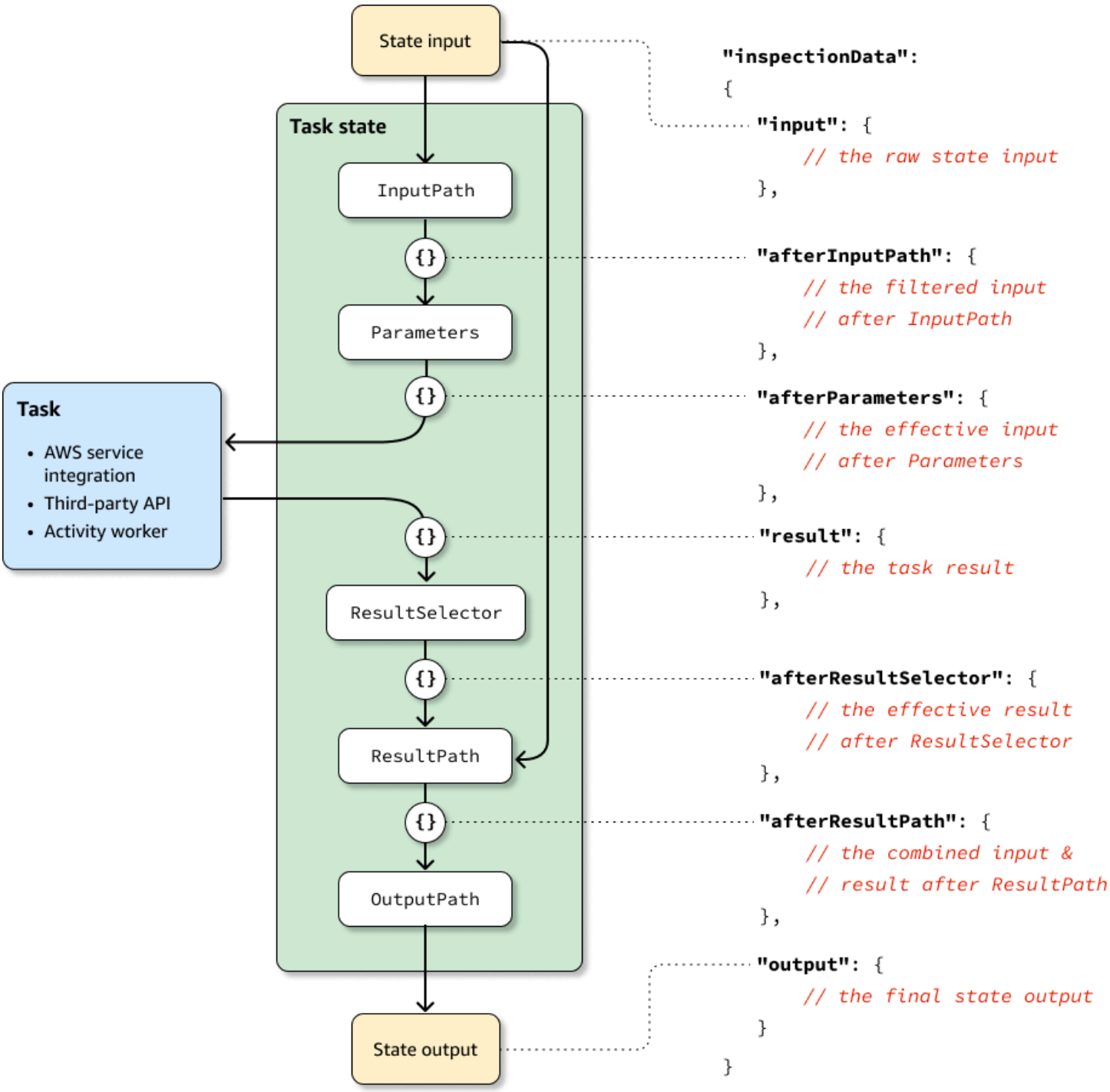
これらの入力処理フィールドと出力処理フィールドはオプションです。ステート定義にこれらのフィールドをまったく使用しない場合、タスクは未加工のステート入力を消費し、タスクの結果をステート出力として返します。

を使用して入出力処理を検査 TestState する

TestState API を呼び出して inspectionLevel パラメータを DEBUG に設定すると、API レスポンスには inspectionData というオブジェクトが含まれます。このオブジェクトには、実行時にステート内でデータがどのようにフィルタリングまたは操作されたかを調べるのに役立つフィールドが含まれています。次の例は、inspectionData オブジェクトのタスクステートを示しています。

```
"inspectionData":   {
  "input": string,
  "afterInputPath": string,
  "afterParameters": string,
  "result": string,
  "afterResultSelector": string,
  "afterResultPath": string,
  "output": string
}
```

この例では、after プレフィックスを含む各フィールドには、特定のフィールドが適用された後のデータが表示されます。例えば、afterInputPath は InputPath フィールドを適用して未加工のステート入力をフィルタリングしたときの効果を示しています。以下の図は、各 ASL 定義フィールドを inspectionData オブジェクト内の対応するフィールドにマッピングしています。



TestState API を使用して入出力処理をデバッグする例については、以下を参照してください。

- Step Functions コンソールの DEBUG 検査レベルを使用してステートをテストする
- の DEBUG 検査レベルを使用した状態のテスト AWS CLI

ステートマシンのローカルテスト

AWS Step Functions Local は Step Functions のダウンロード可能なバージョンです。これを使用すると、独自の開発環境で実行されている Step Functions のバージョンを使用してアプリケーションを開発およびテストできます。ローカルバージョンの Step Functions は、AWS で、およびローカルでの実行中に、AWS Lambda 関数を呼び出すことができます。また、他の サポートされている AWS のサービスを調整することもできます。

> **Note**
>
> Step Functions Local はダミーアカウントを使用して動作します。

Step Functions Local を実行している間は、以下のいずれかの方法でサービス統合を呼び出すことができます。

- AWS Lambda およびその他のサービスのローカルエンドポイントを設定します。サポートされているエンドポイントの情報については、「Step Functions Local の設定オプションを指定する」を参照してください。
- Step Functions Local から AWS サービスを直接呼び出します。
- サービス統合からのレスポンスをモックします。モックされたサービス統合の使用に関する情報については、「モックサービス統合の使用」を参照してください。

AWS Step Functions Local は、Microsoft Windows、Linux、macOS、および、Java または Docker をサポートするその他のプラットフォームで動作する JAR パッケージまたは自己完結型の Docker イメージとして利用できます。

> **Warning**
>
> ダウンロード可能なバージョンの AWS Step Functions は、テストにのみ使用することを意図したものであり、機密情報を処理するためには決して使用しないでください。

> **Tip**
>
> ワークフローにすべての組み込み関数を含めるには、Step Functions Local のバージョン 1.12.0 以降を使用していることを確認してください。

以下のトピックでは、Docker と JAR ファイルを使用して Step Functions Local を設定する方法と、Step Functions Local を実行して AWS Lambda、AWS Serverless Application Model (AWS SAM) CLI Local、またはサポートされている他のサービスと連携する方法について説明します。

トピック

- Step Functions Local (ダウンロード可能バージョン) と Docker のセットアップ
- Step Functions Local (ダウンロード可能バージョン) - Java バージョンのセットアップ
- Step Functions Local の設定オプションを指定する
- Step Functions Local を自分のコンピュータで実行する
- Step Functions と AWS SAM CLI Local のテスト
- モックサービス統合の使用

Step Functions Local (ダウンロード可能バージョン) と Docker のセットアップ

Step Functions Local Docker イメージでは、Docker イメージを必要なすべての依存関係とともに使用して、Step Functions Local の使用をすぐにスタートできます。この Docker イメージでは、Step Functions Local を継続的統合テストの一部としてコンテナ化されたビルドに組み込むことができます。

Step Functions Local の Docker イメージを取得するには、https://hub.docker.com/r/amazon/aws-stepfunctions-local を参照するか、次の Docker `pull` コマンドを入力します。

```
docker pull amazon/aws-stepfunctions-local
```

ダウンロード可能なバージョンの Step Functions を Docker で起動するには、次の Docker `run` コマンドを実行します。

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

AWS Lambda またはサポートされている他のサービスとやり取りするには、最初に認証情報と他の設定オプションを指定する必要があります。詳細については、次のトピックを参照してください。

- Step Functions Local の設定オプションを指定する
- Docker の認証情報と設定

Step Functions Local (ダウンロード可能バージョン) - Java バージョンのセットアップ

ダウンロード可能なバージョンの AWS Step Functions が、実行可能な JAR ファイルおよび Docker イメージとして提供されています。Java アプリケーションは、Windows、Linux、macOS、および Java をサポートする他のプラットフォームで動作します。Java に加えて、AWS Command Line Interface (AWS CLI) のインストールも必要です。AWS CLIのインストールおよび設定情報については、AWS Command Line Interface ユーザーガイドを参照してください。

コンピュータで Step Functions をセットアップして実行するには

1. 次のリンクを使用して Step Functions をダウンロードします。

ダウンロードリンク	チェックサム
.tar.gz	.tar.gz.md5
.zip	.zip.md5

2. `.zip` ファイルを抽出します。
3. ダウンロードをテストし、バージョン情報を表示します。

```
$ java -jar StepFunctionsLocal.jar -v
Step Function Local
Version: 1.0.0
Build: 2019-01-21
```

4. (オプション) 使用可能なコマンドのリストを表示します。

```
$ java -jar StepFunctionsLocal.jar -h
```

5. コンピュータで Step Functions をスタートするには、コマンドプロンプトを開き、`StepFunctionsLocal.jar` を展開したディレクトリに移動して、次のコマンドを入力します。

```
java -jar StepFunctionsLocal.jar
```

6. ローカルで実行中の Step Functions にアクセスするには、`--endpoint-url` パラメータを使用します。例えば、AWS CLI を使用して Step Functions コマンドを次のように指定します。

```
aws stepfunctions --endpoint-url http://localhost:8083 command
```

> **Note**
>
> デフォルトでは、Step Functions Local はローカルテストアカウントと認証情報を使用し、AWS リージョンは米国東部 (バージニア北部) に設定されます。AWS Lambda と共に Step Functions Local を使用またはその他のサポートされているサービスで使用するには、認証情報とリージョンを設定する必要があります。
> Express ワークフローを Step Functions Local で使用すると、実行履歴はログファイルに保存されます。これは CloudWatch ログには記録されません。ログファイルのパスは、ローカルステートマシンの作成時に指定された CloudWatch Logs ロググループ ARN に基づきます。ログファイルは、Step Functions Local を実行しているロケーションに相対的な `/aws/states/log-group-name/${execution_arn}.log` に保存されます。例えば、実行 ARN が以下であるとします。
>
> ```
> arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI
> ```
>
> ログファイルは以下のようになります。
>
> ```
> aws/states/log-group-name/arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI.log
> ```

Step Functions Local の設定オプションを指定する

JAR ファイルを使って AWS Step Functions Local を起動すると、AWS Command Line Interface (AWS CLI) を使用するかオプションをシステム環境に含めることにより設定オプションを指定できます。Docker の場合、Step Functions Local を起動するときに、リファレンスにするファイルでこれらのオプションを指定する必要があります。

設定オプション

Lambda エンドポイントや Batch エンドポイントなどのオーバーライドエンドポイントを使用するよう Step Functions Local コンテナを設定して、そのエンドポイントを呼び出す場合、Step Functions

Local では、指定した認証情報は使用されません。これらのエンドポイントオーバーライドの設定は任意です。

オプション	コマンドライン	環境
アカウント	-account, --aws-account	AWS_ACCOUNT_ID
リージョン	-region, --aws-region	AWS_DEFAULT_REGION
待機タイムスケール	-waitTimeScale, --wait-time-scale	WAIT_TIME_SCALE
Lambda エンドポイント	-lambdaEndpoint, --lambda-endpoint	LAMBDA_ENDPOINT
バッチエンドポイント	-batchEndpoint, --batch-endpoint	BATCH_ENDPOINT
DynamoDB エンドポイント	-dynamoDBEndpoint, --dynamodb-endpoint	DYNAMODB_ENDPOINT
ECS エンドポイント	-ecsEndpoint, --ecs-endpoint	ECS_ENDPOINT
Glue エンドポイント	-glueEndpoint, --glue-endpoint	GLUE_ENDPOINT
SageMaker エンドポイント	-sageMakerEndpoint, --sagemaker-endpoint	SAGE_MAKER_ENDPOINT
SQS エンドポイント	-sqsEndpoint, --sqs-endpoint	SQS_ENDPOINT
SNS エンドポイント	-snsEndpoint, --sns-endpoint	SNS_ENDPOINT
Step Functions エンドポイント	-stepFunctionsEndpoint, --step-functions-endpoint	STEP_FUNCTIONS_ENDPOINT

Docker の認証情報と設定

Step Functions Local for Docker を設定するには、次のファイル `aws-stepfunctions-local-credentials.txt` を作成します。

このファイルには、認証情報および他の設定オプションが含まれています。aws-stepfunctions-local-credentials.txt ファイル作成時のテンプレートとして、以下を使用できます。

```
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
STEP_FUNCTIONS_ENDPOINT=VALUE
```

認証情報と設定オプションを aws-stepfunctions-local-credentials.txt で設定したら、次のコマンドを使用して Step Functions を起動します。

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

> **Note**
>
> ホストが使用する内部 IP アドレス (http://host.docker.internal:8000 など) に対応する特別な DNS 名、host.docker.internal を使用することをお勧めします。詳細については、Networking features in Docker Desktop for Mac および Networking features in Docker Desktop for Windows で、Mac 用と Windows 用の Docker ドキュメンテーションをそれぞれ参照してください。

Step Functions Local を自分のコンピュータで実行する

ローカルバージョンの Step Functions を使用して、コンピュータでステートマシンを設定、開発、およびテストします。

HelloWorld ステートマシンをローカルで実行する

Step Functions を AWS Command Line Interface (AWS CLI) でローカルに実行したら、ステートマシンの実行をスタートできます。

1. ステートマシン定義をエスケープして、AWS CLI からステートマシンを作成します。

```
aws stepfunctions --endpoint-url 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**
>
> `role-arn` は Step Functions Local には使用されませんが、適切な構文に含める必要があります。前の例の Amazon リソースネーム (ARN) を使用できます。

ステートマシンを正常に作成すると、Step Functions は作成日とステートマシンの ARN で応答します。

```
{
    "creationDate": 1548454198.202,
    "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld"
}
```

2. 作成したステートマシンの ARN を使用して実行を開始します。

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

AWS SAM CLI Local 付きStep Functions

ローカルバージョンの AWS Lambda を、ローカルバージョンの Step Functions で使用できます。これを設定するには、AWS SAM をインストールして設定する必要があります。

AWS SAM の設定と実行の詳細については、以下を参照してください。

- AWS SAM のセットアップ
- AWS SAM CLI Local の起動

ローカルシステムで Lambda が実行されたら、Step Functions Local をスタートできます。Step Functions Local の JAR ファイルを展開したディレクトリから Step Functions Local を起動し、`--lambda-endpoint` パラメータを使用してローカルの Lambda エンドポイントを設定します。

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

AWS Lambda との Step Functions Local の実行の詳細については、Step Functions と AWS SAM CLI Local のテスト を参照してください。

Step Functions と AWS SAM CLI Local のテスト

ローカルマシンで AWS Step Functions および AWS Lambda の両方が実行されている状態で、コードを AWS にデプロイすることなく、ステートマシンと Lambda 関数をテストできます。

詳細については、次のトピックを参照してください。

- ステートマシンのローカルテスト
- AWS SAM のセットアップ

トピック

- ステップ 1: AWS SAM を設定する
- ステップ 2: AWS SAM CLI Local をテストする
- ステップ 3: AWS SAM CLI Local を起動する
- ステップ 4: Step Functions Local のスタート
- ステップ 5: AWS SAM CLI Local 関数を参照するステートマシンを作成する

- ステップ 6: ローカルステートマシンの実行を開始する

ステップ 1: AWS SAM を設定する

AWS Serverless Application Model (AWS SAM) CLI Local では、AWS Command Line Interface、AWS SAM、および Docker をインストールする必要があります。

1. AWS SAM CLI をインストールします。

   > **Note**
   >
   > AWS SAM CLI をインストールする前に、AWS CLI および Docker をインストールする必要があります。AWS SAM CLI インストールについては、[Prerequisites] (前提条件) を参照。

2. AWS SAM クイックスタートドキュメントをひととおり確認してください。必ず、以下のステップに従います。

   1. アプリケーションの初期化
   2. アプリケーションのローカルテスト

   これにより、sam-app ディレクトリが作成され、Python ベースの Hello World Lambda 関数を含む環境が構築されます。

ステップ 2: AWS SAM CLI Local をテストする

これで AWS SAM のインストールが完了し、Hello World Lambda 関数を作成できたため、この関数をテストします。sam-app ディレクトリで、次のコマンドを入力します。

```
sam local start-api
```

これにより、Lambda 関数のローカルインスタンスが起動されます。次のような出力が表示されます。

```
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)
```

ブラウザを開き、次のように入力します。

```
http://127.0.0.1:3000/hello
```

これにより、次のようなレスポンスが出力されます。

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

Ctrl+C キーを押して、Lambda API を終了します。

ステップ 3: AWS SAM CLI Local を起動する

関数が機能することをテストしたので、AWS SAM CLI Local を起動します。sam-app ディレクトリで、次のコマンドを入力します。

```
sam local start-lambda
```

これにより AWS SAM CLI Local が起動し、次の出力のように、使用するエンドポイントが表示されます。

```
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)
```

ステップ 4: Step Functions Local のスタート

JAR ファイル

Step Functions Local の .jar ファイルバージョンを使用している場合は、Lambda エンドポイントを指定して Step Functions をスタートします。.jar ファイルを展開したディレクトリで、次のコマンドを入力します。

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

Step Functions Local がスタートすると、環境に続いて、`~/.aws/credentials` ファイルで設定した認証情報がチェックされます。デフォルトでは、架空のユーザー ID を使用して起動し、`region` `us-east-1` としてリスト表示されます。

```
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

Docker バージョンの Step Functions Local を使用している場合、次のコマンドを使用して Step Functions を起動します。

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

Docker バージョンの Step Functions のインストールについては、Step Functions Local (ダウンロード可能バージョン) と Docker のセットアップ を参照してください。

> **Note**
>
> または `.jar` ファイルから Step Functions を起動する場合は環境変数を設定することで、コマンドラインを通じて、エンドポイントを指定できます。Docker バージョンの場合は、エンドポイントと認証情報をテキストファイルに指定する必要があります。Step Functions Local の設定オプションを指定する を参照してください。

ステップ 5: AWS SAM CLI Local 関数を参照するステートマシンを作成する

Step Functions Local が実行されたら、ステップ 1: AWS SAM を設定する で初期化した `HelloWorldFunction` を参照するステートマシンを作成します。

```
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"
```

これによりステートマシンが作成され、実行をスタートするために使用できる Amazon リソースネーム (ARN) が提供されます。

```
{
    "creationDate": 1548805711.403,
    "stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"
}
```

ステップ 6: ローカルステートマシンの実行を開始する

ステートマシンを作成したら、実行を開始します。次の **aws stepfunctions** コマンドを使用するときは、エンドポイントとステートマシンの ARN を参照する必要があります。

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

これにより、HelloWorld ステートマシンの test という実行が開始されます。

```
{
    "startDate": 1548810641.52,
    "executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test"
}
```

これで Step Functions がローカルに実行されたので、AWS CLI を使用して操作することができます。例えば、この実行に関する情報を取得するには、以下のコマンドを使用します。

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

実行のために describe-execution を呼び出すと、次のような出力により、完全な詳細が提供されます。

```
{
    "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": "{}"
}
```

モックサービス統合の使用

Step Functions Local では、モックサービス統合を使用して、統合サービスを実際に呼び出さなくてもステートマシンの実行パスをテストできます。モックサービス統合を使用するようにステートマシンを設定するには、モック設定ファイルを作成します。このファイルでは、サービス統合に必要な出力をモックレスポンスとして定義し、モックレスポンスを使用して実行パスをシミュレートする実行をテストケースとして定義します。

Step Functions Local にモック設定ファイルを提供して、実際のサービス統合呼び出しを行う代わりに、テストケースで指定されたモックレスポンスを使用するステートマシンを実行して、サービス統合呼び出しをテストできます。

> **Note**
>
> モック設定ファイルでモックサービス統合レスポンスを指定しない場合、Step Functions Local は Step Functions Local のセットアップ時に設定したエンドポイントを使用して AWS サービス統合を呼び出します。Step Functions Local のエンドポイントの設定については、「Step Functions Local の設定オプションを指定する」を参照してください。

トピック

- このトピックの主要概念
- ステップ 1: モック設定ファイルでモックサービス統合を指定する
- ステップ 2: Step Functions Local にモック設定ファイルを提供する
- ステップ 3: モックサービス統合テストを実行する

- モックサービス統合の設定ファイル

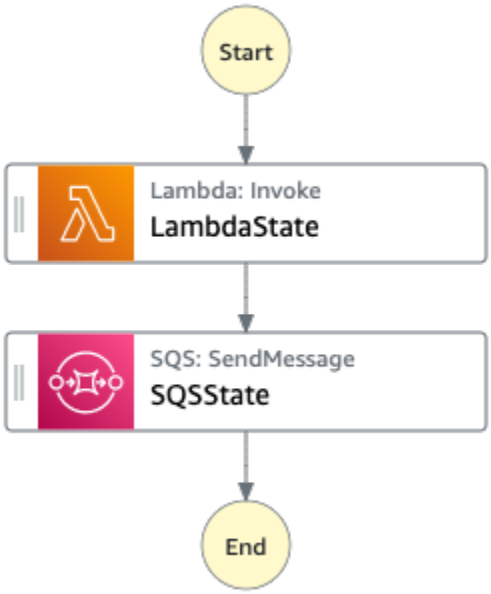
このトピックの主要概念

このトピックでは、以下のリストに定義されているいくつかの概念を使用しています。

- モックサービス統合 - 実際のサービス呼び出しを実行する代わりにモックレスポンスを使用するように設定された Task ステートを指します。
- モックレスポンス - Task ステートが使用するように設定できるモックデータを指します。
- テストケース - モックサービス統合を使用するように設定されたステートマシン実行を指します。
- モック設定ファイル - JSON を含むモック設定ファイルを指します。JSON はモックサービス統合、モックレスポンス、テストケースを定義します。

ステップ 1: モック設定ファイルでモックサービス統合を指定する

Step Functions AWS SDK と最適化されたサービス統合は、Step Functions Local を使用してテストできます。以下のイメージは、[ステートマシンの定義] タブで定義されているステートマシンを示しています。



そのためには、モック設定の構造の紹介 で定義されているセクションを含むモック設定ファイルを作成する必要があります。

1. モックサービス統合を含むテストを設定するための MockConfigFile.json という名前のファイルを作成します。

   次の例は、LambdaState と SQSState という名前の 2 つのステートが定義されているステートマシンを参照するモック設定ファイルを示しています。

   Mock configuration file example

   以下は、Lambda 関数を呼び出して Amazon SQS にメッセージを送信するレスポンスをモックする方法を示すモック設定ファイルの例です。この例では、LambdaSQSIntegration ステートマシンには、HappyPath、RetryPath、HybridPath という名前の 3 つのテストケースがあり、これらは LambdaState および SQSState という名前の Task ステートをモックします。これらのステートは MockedLambdaSuccess、MockedSQSSuccess、MockedLambdaRetry およびモックサービスのレスポンスを使用します。これらのモックサービスのレスポンスはファイルの MockedResponses セクションで定義されています。

```
{
  "StateMachines":{
    "LambdaSQSIntegration":{
      "TestCases":{
        "HappyPath":{
          "LambdaState":"MockedLambdaSuccess",
          "SQSState":"MockedSQSSuccess"
        },
        "RetryPath":{
          "LambdaState":"MockedLambdaRetry",
          "SQSState":"MockedSQSSuccess"
        },
        "HybridPath":{
          "LambdaState":"MockedLambdaSuccess"
        }
      }
    }
  },
  "MockedResponses":{
    "MockedLambdaSuccess":{
      "0":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
```

```
            "body":"Hello from Lambda!"
          }
        }
      }
    },
    "LambdaMockedResourceNotReady":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      }
    },
    "MockedSQSSuccess":{
      "0":{
        "Return":{
          "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
          "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
        }
      }
    },
    "MockedLambdaRetry":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      },
      "1-2":{
        "Throw":{
          "Error":"Lambda.TimeoutException",
          "Cause":"Lambda timed out."
        }
      },
      "3":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    }
```

```
  }
}
```

State machine definition

以下は、LambdaState および SQSState という名前の 2 つのサービス統合タスクステートを定義する LambdaSQSIntegration というステートマシン定義の例です。LambdaState には、States.ALL に基づく再試行ポリシーが含まれています。

```
{
  "Comment":"This state machine is called: LambdaSQSIntegration",
  "StartAt":"LambdaState",
  "States":{
    "LambdaState":{
      "Type":"Task",
      "Resource":"arn:aws:states:::lambda:invoke",
      "Parameters":{
        "Payload.$":"$",
        "FunctionName":"HelloWorldFunction"
      },
      "Retry":[
        {
          "ErrorEquals":[
            "States.ALL"
          ],
          "IntervalSeconds":2,
          "MaxAttempts":3,
          "BackoffRate":2
        }
      ],
      "Next":"SQSState"
    },
    "SQSState":{
      "Type":"Task",
      "Resource":"arn:aws:states:::sqs:sendMessage",
      "Parameters":{
        "QueueUrl":"https://sqs.us-east-1.amazonaws.com/123456789012/myQueue",
        "MessageBody.$":"$"
      },
      "End": true
    }
  }
```

```
}
```

以下のテストケースのいずれかを使用して、モック設定ファイルで参照されている LambdaSQSIntegration ステートマシン定義を実行できます。

- HappyPath - このテストでは、それぞれ MockedLambdaSuccess と MockedSQSSuccess を使用して LambdaState と SQSState の出力をモックします。
  - LambdaState は次の値を返します。

```
"0":{
  "Return":{
    "StatusCode":200,
    "Payload":{
      "StatusCode":200,
      "body":"Hello from Lambda!"
    }
  }
}
```

  - SQSState は次の値を返します。

```
"0":{
  "Return":{
    "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
    "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
  }
}
```

- RetryPath - このテストでは、それぞれ MockedLambdaRetry と MockedSQSSuccess を使用して LambdaState と SQSState の出力をモックします。また、LambdaState は 4 回再試行するように設定されています。これらの試行に対するモックレスポンスは MockedLambdaRetry ステートで定義され、インデックス化されます。
  - 最初の試行は、次の例に示すように、原因とエラーメッセージを含むタスク失敗で終了します。

```
"0":{
  "Throw": {
    "Error": "Lambda.ResourceNotReadyException",
    "Cause": "Lambda resource is not ready."
  }
```

```
}
```

- 1 回目と 2 回目の再試行は、次の例に示すように、原因とエラーメッセージを含むタスク失敗で終了します。

```
"1-2":{
  "Throw": {
    "Error": "Lambda.TimeoutException",
    "Cause": "Lambda timed out."
  }
}
```

- 3 回目の再試行は、モックされた Lambda レスポンスの Payload セクションからのステート結果を含むタスク成功で終了します。

```
"3":{
  "Return": {
    "StatusCode": 200,
    "Payload": {
      "StatusCode": 200,
      "body": "Hello from Lambda!"
    }
  }
}
```

> **Note**
>
> - 再試行ポリシーが設定されているステートでは、Step Functions Local は成功レスポンスを受け取るまで、ポリシーに設定された再試行回数を使い果たします。つまり、モックの再試行を連続する試行回数で示し、成功レスポンスを返すまですべての再試行を実施する必要があります。
> - 再試行「3」のように、特定の再試行に対してモックレスポンスを指定しないと、ステートマシンの実行は失敗します。

- `HybridPath` - このテストは `LambdaState` の出力をモックします。`LambdaState` が正常に実行され、モックされたデータをレスポンスとして受信したら、`SQSState` は本番環境で指定されたリソースに対して実際のサービス呼び出しを実行します。

モックサービス統合を使用してテスト実行を開始する方法については、「ステップ 3: モックサービス統合テストを実行する」を参照してください。

2. モックレスポンスの構造が、統合サービス呼び出しを行うときに受け取る実際のサービスレスポンスの構造と一致していることを確認してください。モックレスポンスの構造要件については、「モックサービス統合の設定」を参照してください。

   前述のモック設定ファイルの例では、`MockedLambdaSuccess` および `MockedLambdaRetry` で定義されたモック応答は、`HelloFromLambda` の呼び出しから返される実際の応答の構造に準拠しています。

   > **Important**
   >
   > AWS サービスレスポンスの構造は、サービスによって異なる場合があります。Step Functions Local は、モックレスポンス構造が実際のサービスレスポンス構造に準拠しているかどうかを検証しません。テストする前に、モックレスポンスが実際のレスポンスと一致していることを確認する必要があります。サービスレスポンスの構造を確認するには、Step Functions を使用して実際のサービス呼び出しを実行するか、それらのサービスのドキュメントを参照してください。

ステップ 2: Step Functions Local にモック設定ファイルを提供する

Step Functions Local には、次のいずれかの方法でモック設定ファイルを提供できます。

Docker

> **Note**
>
> Docker バージョンの Step Functions Local を使用している場合、環境変数のみを使用してモック設定ファイルを指定できます。また、サーバーの初回起動時に Step Functions Local コンテナにモック設定ファイルをマウントする必要があります。

Step Functions Local コンテナ内の任意のディレクトリにモック設定ファイルをマウントします。次に、コンテナ内のモック設定ファイルへのパスを含む `SFN_MOCK_CONFIG` という名前の環境変数を設定します。この方法では、環境変数にファイルパスと名前が含まれている限り、モック設定ファイルに任意の名前を付けることができます。

以下のコマンドは Docker イメージを開始する形式を示しています。

```
docker run -p 8083:8083
--mount type=bind,readonly,source={absolute path to mock config file},destination=/
home/StepFunctionsLocal/MockConfigFile.json
-e SFN_MOCK_CONFIG="/home/StepFunctionsLocal/MockConfigFile.json" amazon/aws-
stepfunctions-local
```

次の例では、コマンドを使用して Docker イメージを開始します。

```
docker run -p 8083:8083
--mount type=bind,readonly,source=/Users/admin/Desktop/workplace/
MockConfigFile.json,destination=/home/StepFunctionsLocal/MockConfigFile.json
-e SFN_MOCK_CONFIG="/home/StepFunctionsLocal/MockConfigFile.json" amazon/aws-
stepfunctions-local
```

JAR File

以下のいずれかの方法を使用してモック設定ファイルを Step Functions Local に提供します。

- モック設定ファイルを Step FunctionsLocal.jar と同じディレクトリに置きます。この方法を使用するときは、モック設定ファイル MockConfigFile.json に名前を付ける必要があります。
- Step Functions Local を実行しているセッションで、SFN_MOCK_CONFIG という名前の環境変数をモック設定ファイルのフルパスに設定します。この方法では、環境変数にそのファイルパスと名前が含まれている限り、モック設定ファイルに任意の名前を付けることができます。次の例では、SFN_MOCK_CONFIG 変数は /home/workspace ディレクトリにある EnvSpecifiedMockConfig.json という名前のモック設定ファイルを指すように設定されています。

  ```
  export SFN_MOCK_CONFIG="/home/workspace/EnvSpecifiedMockConfig.json"
  ```

> **Note**
>
> - Step Functions Local に環境変数 SFN_MOCK_CONFIG を指定しない場合、デフォルトでは、Step Functions Local を起動したディレクトリにある MockConfigFile.json という名前のモック設定ファイルを読み込もうとします。

- モック設定ファイルを Step FunctionsLocal.jar と同じディレクトリに配置して環境変数 SFN_MOCK_CONFIG を設定すると、Step Functions Local は環境変数で指定されたファイルを読み取ります。

ステップ 3: モックサービス統合テストを実行する

モック設定ファイルを作成して Step Functions Local に提供したら、モック設定ファイルに設定されたステートマシンをモックサービス統合を使用して実行します。次に API アクションを使用して実行結果を確認します。

1. モック設定ファイル内の前述の定義に基づいてステートマシンを作成します。

```
aws stepfunctions create-state-machine \
    --endpoint http://localhost:8083 \
    --definition "{\"Comment\":\"Thisstatemachineiscalled:LambdaSQSIntegration
\",\"StartAt\":\"LambdaState\",\"States\":{\"LambdaState\":{\"Type\":
\"Task\",\"Resource\":\"arn:aws:states:::lambda:invoke\",\"Parameters
\":{\"Payload.$\":\"$\",\"FunctionName\":\"arn:aws:lambda:us-
east-1:123456789012:function:HelloWorldFunction\"},\"Retry\":[{\"ErrorEquals
\":[\"States.ALL\"],\"IntervalSeconds\":2,\"MaxAttempts\":3,\"BackoffRate
\":2}],\"Next\":\"SQSState\"},\"SQSState\":{\"Type\":\"Task\",\"Resource\":
\"arn:aws:states:::sqs:sendMessage\",\"Parameters\":{\"QueueUrl\":\"https://
sqs.us-east-1.amazonaws.com/123456789012/myQueue\",\"MessageBody.$\":\"$\"},\"End
\":true}}}" \
    --name "LambdaSQSIntegration" --role-arn "arn:aws:iam::123456789012:role/
service-role/LambdaSQSIntegration"
```

2. モックサービス統合を使用してステートマシンを実行します。

   モック設定ファイルを使用するには、モック設定ファイルに設定されているステートマシンで StartExecution API 呼び出しを行います。これを行うには、StartExecution が使用するステートマシン ARN にサフィックス、#*test_name* を追加します。*test_name* はテストケースで、同じモック設定ファイルでステートマシン用に設定されます。

   以下のコマンドは、LambdaSQSIntegration ステートマシンとモックコンフィグレーションを使用する例です。この例では、ステップ 1: モック設定ファイルでモックサービス統合を指定する で定義された HappyPath テストを使用して LambdaSQSIntegration ステートマシンが実行されます。HappyPath テストには、LambdaState ステートおよび SQSState ステートが

MockedLambdaSuccess および MockedSQSSuccess のモックサービスのレスポンスを使用して行うモックサービス統合呼び出しを処理するための実行設定が含まれています。

```
aws stepfunctions start-execution \
    --endpoint http://localhost:8083 \
    --name executionWithHappyPathMockedServices \
    --state-machine arn:aws:states:us-
east-1:123456789012:stateMachine:LambdaSQSIntegration#HappyPath
```

3. ステートマシンの実行レスポンスを表示します。

   モックサービス統合テストを使用した StartExecution の呼び出しに対するレスポンスは、通常の StartExecution の呼び出しに対するレスポンスと同じであり、実行 ARN と開始日を返します。

   以下は、モックサービス統合テストを使用した StartExecution の呼び出しに対するレスポンスの例です。

```
{
   "startDate":"2022-01-28T15:03:16.981000-05:00",
   "executionArn":"arn:aws:states:us-
east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices"
}
```

4. ListExecutions、DescribeExecution、または GetExecutionHistory API の呼び出しを実行して、実行の結果を確認します。

```
aws stepfunctions get-execution-history \
    --endpoint http://localhost:8083 \
    --execution-arn arn:aws:states:us-
east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices
```

   次の例は、ステップ 2 で示したレスポンス例の実行 ARN を使用して GetExecutionHistory の呼び出しに対する応答の一部を示しています。この例では、LambdaState と SQSState の出力は、モック設定ファイルの MockedLambdaSuccess と MockedSQSSuccess で定義されているモックデータです。また、モックデータは、実際のサービス統合呼び出しを実行して返されるデータと同じ方法で使用されます。また、この例では、LambdaState からの出力が入力として SQSState に渡されます。

```
{
```

```
    "events": [
        ...
        {
            "timestamp": "2021-12-02T19:39:48.988000+00:00",
            "type": "TaskStateEntered",
            "id": 2,
            "previousEventId": 0,
            "stateEnteredEventDetails": {
                "name": "LambdaState",
                "input": "{}",
                "inputDetails": {
                    "truncated": false
                }
            }
        },
        ...
        {
            "timestamp": "2021-11-25T23:39:10.587000+00:00",
            "type": "LambdaFunctionSucceeded",
            "id": 5,
            "previousEventId": 4,
            "lambdaFunctionSucceededEventDetails": {
                "output": "{\"statusCode\":200,\"body\":\"\\\"Hello from Lambda!\\
\"\"}",
                "outputDetails": {
                    "truncated": false
                }
            }
        },
        ...
            "timestamp": "2021-12-02T19:39:49.464000+00:00",
            "type": "TaskStateEntered",
            "id": 7,
            "previousEventId": 6,
            "stateEnteredEventDetails": {
                "name": "SQSState",
                "input": "{\"statusCode\":200,\"body\":\"\\\"Hello from Lambda!\\
\"\"}",
                "inputDetails": {
                    "truncated": false
                }
            }
        },
        ...
```

```
        {
            "timestamp": "2021-11-25T23:39:10.652000+00:00",
            "type": "TaskSucceeded",
            "id": 10,
            "previousEventId": 9,
            "taskSucceededEventDetails": {
                "resourceType": "sqs",
                "resource": "sendMessage",
                "output": "{\"MD5OfMessageBody\":\"3bcb6e8e-7h85-4375-
b0bc-1a59812c6e51\",\"MessageId\":\"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51\"}",
                "outputDetails": {
                    "truncated": false
                }
            }
        },
        ...
    ]
}
```

モックサービス統合の設定ファイル

モックサービス統合を使用するには、まずモック設定を含む MockConfigFile.json という名前のモック設定ファイルを作成する必要があります。次に、Step Functions Local にモック設定ファイルを提供します。この設定ファイルは、モックサービス統合レスポンスを使用するモックステートを含むテストケースを定義します。以下のセクションには、モックステートとモックレスポンスを含むモック設定の構造に関する情報が含まれています。

トピック

- モック設定の構造の紹介
- モックサービス統合の設定

モック設定の構造の紹介

モック設定は、次の最上位のフィールドを含む JSON オブジェクトです。

- StateMachines - このオブジェクトのフィールドは、モックサービス統合を使用するように設定されたステートマシンを表します。
- MockedResponse - このオブジェクトのフィールドは、サービス統合呼び出しのモックレスポンスを表します。

以下は、StateMachine 定義と MockedResponse を含むモック設定ファイルの例です。

```
{
  "StateMachines":{
    "LambdaSQSIntegration":{
      "TestCases":{
        "HappyPath":{
          "LambdaState":"MockedLambdaSuccess",
          "SQSState":"MockedSQSSuccess"
        },
        "RetryPath":{
          "LambdaState":"MockedLambdaRetry",
          "SQSState":"MockedSQSSuccess"
        },
        "HybridPath":{
          "LambdaState":"MockedLambdaSuccess"
        }
      }
    }
  },
  "MockedResponses":{
    "MockedLambdaSuccess":{
      "0":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    },
    "LambdaMockedResourceNotReady":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      }
    },
    "MockedSQSSuccess":{
      "0":{
        "Return":{
          "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
```

```
          "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
        }
      }
    },
    "MockedLambdaRetry":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      },
      "1-2":{
        "Throw":{
          "Error":"Lambda.TimeoutException",
          "Cause":"Lambda timed out."
        }
      },
      "3":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    }
  }
}
```

モック設定フィールドリファレンス

以下のセクションでは、モック設定で定義する必要がある最上位のオブジェクトフィールドについて説明します。

- StateMachines
- MockedResponses

StateMachines

StateMachines オブジェクトは、どのステートマシンがモックサービス統合を使用するかを定義します。各ステートマシンの設定は、StateMachines の最上位フィールドとして表されます。

フィールド名はステートマシンの名前で、値は TestCases という名前の 1 つのフィールドを含むオブジェクトです。そのフィールドはそのステートマシンのテストケースを表します。

次の構文は、2 つのテストケースを含むステートマシンを示しています。

```
"MyStateMachine": {
  "TestCases": {
    "HappyPath": {
      ...
    },
    "SadPath": {
      ...
    }
  }
```

TestCases

TestCases のフィールドはステートマシンの個々のテストケースを表します。各テストケースの名前はステートマシンにつき一意である必要があり、各テストケースの値はステートマシンの Task 状態に使用するモックレスポンスを指定するオブジェクトです。

次の TestCase の例は、2 つの Task 状態を 2 つの MockedResponses にリンクしています。

```
"HappyPath": {
  "SomeTaskState": "SomeMockedResponse",
  "AnotherTaskState": "AnotherMockedResponse"
}
```

MockedResponses

MockedResponses は、一意のフィールド名を持つ複数のモックレスポンスオブジェクトを含むオブジェクトです。モックレスポンスオブジェクトは、モックされた Task 状態が呼び出されるたびに、成功結果またはエラー出力を定義します。呼び出し番号は、「0」、「1」、「2」、「3」などの個別の整数文字列、または「0-1」、「2-3」などの整数の範囲を使用して指定します。

Task をモックする場合、呼び出しのたびにモックレスポンスを指定する必要があります。レスポンスには、Return または Throw という名前のフィールドが 1 つ含まれている必要があります。このフィールドの値は、モックされた Task 呼び出しの結果またはエラー出力です。モックレスポンスを指定しない場合、ステートマシンの実行は失敗します。

以下は、Throw および Return オブジェクトを含む MockedResponse の例です。この例では、ステートマシンを最初の 3 回実行すると、"0-2" で指定されたレスポンスが返され、4 回目にステートマシンが実行されると、"3" で指定されたレスポンスが返されます。

```
"SomeMockedResponse": {
  "0-2": {
    "Throw": {
      ...
    }
  },
  "3": {
    "Return": {
      ...
    }
  }
}
```

> **Note**
>
> Map ステートを使用していて、その Map ステートに対するレスポンスを予測できるようにする場合は、maxConcurrency の値を 1 に設定します。1 より大きい値を設定すると、Step Functions Local は複数の反復を同時に実行するため、反復にわたるステートの全体的な実行順序が予測できなくなります。これにより、Step Functions Local は、ある実行から次の実行までの反復状態に対して異なるモックレスポンスを使用する可能性があります。

戻る

Return は MockedResponse オブジェクトのフィールドとして表されます。Task ステートをモックした場合の成功結果を指定します。

以下は、Lambda 関数の Invoke を呼び出すためのモックレスポンスを含む Return オブジェクトの例です。

```
"Return": {
  "StatusCode": 200,
  "Payload": {
    "StatusCode": 200,
    "body": "Hello from Lambda!"
  }
```

```
}
```

Throw

Throw は MockedResponse オブジェクトのフィールドとして表されます。失敗したタスクのエラー出力を指定します。Throw の値は、文字列値を持つ Error および Cause フィールドを含むオブジェクトである必要があります。MockConfigFile.json の Error フィールドに指定する文字列値は、ステートマシンの Retry および Catch セクションで処理されるエラーと一致する必要があります。

以下は、Lambda 関数の Invoke を呼び出すためのモックレスポンスを含む Throw オブジェクトの例です。

```
"Throw": {
  "Error": "Lambda.TimeoutException",
  "Cause": "Lambda timed out."
}
```

モックサービス統合の設定

Step Functions Local を使用して任意のサービス統合をモックできます。ただし、Step Functions Local はモックを実際の API と同じにすることを強制しません。モックされた Task がサービスエンドポイントを呼び出すことはありません。モックレスポンスを指定しない場合、Task はサービスエンドポイントを呼び出そうとします。また、Step Functions Local は、.waitForTaskToken を使用してタスクをモックすると、自動的にタスクトークンを生成します。

Step Functions のベストプラクティス

AWS Step Functions ワークフローを実装するための以下のベストプラクティスを使用すると、実装のパフォーマンスを最適化できます。

トピック

- タイムアウトを使用して実行のスタックを回避する
- ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する
- 履歴のクォータに到達しないようにする
- Lambda サービスの例外を処理する
- アクティビティタスクのポーリング時のレイテンシーを回避する
- 標準ワークフローまたは Express ワークフローの選択
- Amazon CloudWatch Logs リソースポリシーのサイズ制限

タイムアウトを使用して実行のスタックを回避する

デフォルトでは、Amazon States Language はステートマシンの定義にタイムアウトを指定しません。明示的なタイムアウトが設定されていないと、Step Functions は多くの場合、アクティビティのワーカーからのレスポンスでしか、タスクが完了したことを知ることができません。エラーが発生した場合、`TimeoutSeconds` フィールドに `Activity` 状態または `Task` 状態が指定されていないと、実行は、返されることのないレスポンスを待ち続けるため、スタックします。

この状況を回避するには、ステートマシンで `Task` を作成するときに、適切なタイムアウトを指定します。例:

```
"ActivityState": {
  "Type": "Task",
  "Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld",
  "TimeoutSeconds": 300,
  "Next": "NextState"
}
```

タスクトークン (.waitForTaskToken) でコールバックを使用する場合は、ハートビートを使用して `Task` 状態定義に `HeartbeatSeconds` フィールドを追加することを推奨します。`HeartbeatSeconds` をタスクのタイムアウトよりも短く設定できるので、ワークフローがハー

トビートエラーで失敗した場合、タスクが完了するまでに時間がかかったのではなく、タスクの失敗が原因であることがわかります。

```
{
  "StartAt": "Push to SQS",
  "States": {
    "Push to SQS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
      "HeartbeatSeconds": 600,
      "Parameters": {
        "MessageBody": { "myTaskToken.$": "$$.Task.Token" },
        "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/push-based-queue"
      },
      "ResultPath": "$.SQS",
      "End": true
    }
  }
}
```

詳細については、Amazon States Language ドキュメントの タスク を参照してください。

> **Note**
>
> Amazon States Language の定義の `TimeoutSeconds` フィールドを使用してステートマシンのタイムアウトを設定できます。詳細については、「ステートマシン構造」を参照してください。

ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する

状態間でデータの大きいペイロードを渡す実行を終了できます。ステートの間で渡すデータが 256 KB を超える場合、Amazon Simple Storage Service (Amazon S3) を使用してデータを保存し、バケット名とキーバリューを取得するため、`Payload` パラメータでバケットの Amazon リソースネーム (ARN) を解析します。または、実行時に小さいペイロードを渡すように実装を調整します。

次の例では、ステートマシンは Amazon S3 バケットの JSON ファイルを処理する AWS Lambda 関数に入力を渡しています。このステートマシンを実行すると、Lambda 関数は JSON ファイルの内容を読み取り、ファイルの内容を出力として返します。

Lambda 関数を作成する

pass-large-payload という名前の次の Lambda 関数は、特定の Amazon S3 バケットに保存されている JSON ファイルの内容を読み取ります。

> **Note**
>
> この Lambda 関数を作成したら、必ず、その IAM ロールに Amazon S3 バケットから読み取るための適切なアクセス許可を付与してください。例えば、Lambda 関数のロールに AmazonS3ReadOnlyAccess のアクセス許可をアタッチします。

```
import json
import boto3
import io
import os

s3 = boto3.client('s3')

def lambda_handler(event, context):
    event = event['Input']
    final_json = str()

    s3 = boto3.resource('s3')
    bucket = event['bucket'].split(':')[-1]
    filename = event['key']
    directory = "/tmp/{}".format(filename)

    s3.Bucket(bucket).download_file(filename, directory)

    with open(directory, "r") as jsonfile:

        final_json = json.load(jsonfile)

    os.popen("rm -rf /tmp")

    return final_json
```

ステートマシンを作成する

次のステートマシンは、以前に作成した Lambda 関数を呼び出します。

```
{
```

```
   "StartAt":"Invoke Lambda function",
   "States":{
      "Invoke Lambda function":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-2:123456789012:function:pass-large-payload",
            "Payload":{
               "Input.$":"$"
            }
         },
         "OutputPath": "$.Payload",
         "End":true
      }
   }
}
```

大量のデータを入力で渡すのではなく、そのデータを Amazon S3 バケットに保存し、Payload パラメータでバケットの Amazon リソースネーム (ARN) を解析して、バケット名とキーバリューを取得します。Lambda 関数はその ARN を使用してデータに直接、アクセスできます。次に示しているステートマシン実行の入力例では、データが *large-payload-json* という名前の Amazon S3 バケット内の data.json に保存されています。

```
{
  "key": "data.json",
  "bucket": "arn:aws:s3:::large-payload-json"
}
```

履歴のクォータに到達しないようにする

AWS Step Functions のイベント履歴には、25,000 エントリのハードクォータがあります。実行イベントが 24,999 件に達すると、次のイベントが発生するまで待機します。

- イベント番号 25,000 が ExecutionSucceeded の場合、実行は正常に終了します。
- イベント番号 25,000 が ExecutionSucceeded 以外の場合、ExecutionFailed イベントは口グに記録され、履歴の上限に達したためステートマシンの実行は失敗します。

長時間の実行でこのクォータに達しないようにするには、次のいずれかの方法を試行できます。

- マップステートを分散モードで使用します。このモードでは、Map 状態は各反復を子ワークフロー実行として実行します。これにより、最大 10,000 件という高い並列性を持つの並列子ワークフロー実行が可能になります。それぞれの子ワークフローの実行には、親ワークフローとは別の実行履歴があります。
- 進行中の実行の Task 状態から、新しいステートマシンを直接実行します。このようなネストされたワークフロー実行を開始するには、親ステートマシンで必要なパラメータとともに、Step Functions の `StartExecution` API アクションを使用します。ネストされたワークフローの使用についての詳細は、「タスク状態からワークフロー実行を開始する」または「Step Functions API アクションを使用して新しい実行を継続する」のチュートリアルを参照してください。

  > **Tip**
  >
  > ネストされたワークフローの例を AWS アカウント にデプロイするには、「モジュール 13 - ネストされた Express ワークフロー」を参照してください。

- AWS Lambda 関数を使用するパターンを実装します。この関数は、ステートマシンの新しい実行を開始して、複数のワークフロー実行で進行中の作業を分割できます。詳細については、Lambda 関数を使用して新しい実行を継続する のチュートリアルを参照してください。

Lambda サービスの例外を処理する

AWS Lambda では、サービスエラーが発生することがあります。この場合、Lambda を呼び出すと、`ClientExecutionTimeoutException`、`ServiceException`、`AWSLambdaException`、または `SdkClientException` などの 500 エラーが生じます。ベストプラクティスとしては、Lambda 関数を呼び出す `Retry`、あるいはエラーを `Catch` するために、ステートマシンでのこのような例外を事前に処理します。

Lambda エラーは、`Lambda.`*`ErrorName`* として報告されます。Lambda サービス例外エラーを再試行するには、次の `Retry` コードを使用します。

```
"Retry": [ {
   "ErrorEquals": [ "Lambda.ClientExecutionTimeoutException",
 "Lambda.ServiceException", "Lambda.AWSLambdaException", "Lambda.SdkClientException"],
   "IntervalSeconds": 2,
   "MaxAttempts": 6,
   "BackoffRate": 2
} ]
```

> **Note**
>
> Lambda での未処理のエラーは、エラー出力で Lambda.Unknown として報告されます。これには、メモリ不足エラーと関数のタイムアウトが含まれます。Lambda.Unknown、States.ALL、または States.TaskFailed を一致させて、こういったエラーに処理できます。Lambda が最大呼び出し数に達すると、エラーは Lambda.TooManyRequestsException となります。Lambda 関数のエラーの詳細については、「AWS Lambda デベロッパーガイド」の「エラー処理と自動再試行」を参照してください。

詳細については、以下を参照してください。

- エラー後の再試行
- Step Functions ステートマシンを使用してエラー条件を処理する
- Lambda 呼び出しエラー

アクティビティタスクのポーリング時のレイテンシーを回避する

GetActivityTask API は、taskToken を 1 回のみ提供するように設計されています。アクティビティワーカーと通信している間に taskToken がドロップされた場合、GetActivityTask がタイムアウトするまで、レスポンスの待機のため複数の GetActivityTask リクエストが 60 秒ブロックされます。

レスポンス待ちのポーリングが少数しかない場合、ブロックされたリクエストの後ろにすべてのリクエストが追加され、停止する可能性があります。ただし、各アクティビティ Amazon リソースネーム (ARN) に多数の未処理のポーリングがあり、リクエストの一部が待機中のままになる場合、taskToken を取得して処理の実行を開始するリクエストがさらにたくさんあります。

本番稼働用システムでは、それぞれのアクティビティ ARN の各時点で少なくとも 100 のオープンポーリングを推奨します。1 つのポーリングがブロックされ、それらのポーリングの一部がその後ろに並んでいる場合、GetActivityTask のリクエストがブロックされている間に処理を実行するための taskToken を受け取るさらに多くのリクエストがあります。

タスクのポーリング時に、これらのレイテンシーの問題を回避する方法。

- アクティビティワーカーの実装の作業とは別のスレッドとしてポーラーを実装します。
- 各時点でのアクティビティ ARN あたり、少なくとも 100 のオープンポーリングが必要です。

> **Note**
>
> ARN あたり 100 のオープンポーリングにスケーリングすると、コストが高くなる可能性があります。例えば、ARN あたり 100 の Lambda 関数でポーリングする場合、1 つの Lambda 関数で 100 のポーリングスレッドを実行するよりも 100 倍コストが高くなります。レイテンシーを短縮しながらコストを最小限に抑えるには、非同期 I/O を使用する言語により、ワーカーごとに複数のポーリングスレッドを実装します。ポーラースレッドとワークスレッドが異なるアクティビティワーカーの例については、Ruby のサンプルアクティビティワーカー を参照してください

アクティビティおよびアクティビティワーカーの詳細については、アクティビティ を参照してください

標準ワークフローまたは Express ワークフローの選択

AWS Step Functions では、デフォルトのワークフロータイプとして標準ワークフローが提供されるとともに、Express ワークフローを選択するオプションがあります。

長時間実行され、耐久性が高く、監査可能なワークフローが必要な場合は、スタンダードなワークフローを選択できます。または、ハイボリュームのイベント処理ワークロードの場合に Express Workflow を選択できます。ステートマシンの実行の動作は、選択した Type によって異なります。ステートマシンが作成された後は、選択した Type を変更することはできません。

- Standard ワークフローと Express ワークフローの違いの詳細については、「標準ワークフロー対 Express ワークフロー」を参照してください。
- Step Functions を使用してサーバーレスワークフローを構築する際のコストの最適化については、「Express ワークフローによるコスト最適化」を参照してください。

Amazon CloudWatch Logs リソースポリシーのサイズ制限

CloudWatch Logs リソースポリシーは 5120 文字に制限されています。CloudWatch Logs は、ポリシーがこのサイズ制限に近づいていることを検出すると、`/aws/vendedlogs/` でスタートするロググループを自動的に有効にします。

ログが有効なステートマシンを作成する場合、Step Functions は、指定したロググループで CloudWatch Logs リソースポリシーを更新する必要があります。CloudWatch Logs リソースポ

リシーのサイズ制限に達しないようにするには、CloudWatch Logs ロググループ名の先頭にプレフィックスとして `/aws/vendedlogs/` を付けます。Step Function コンソールでロググループを作成すると、ロググループ名に `/aws/vendedlogs/states` のプレフィックスが付きます。詳細については、の特定の AWS のサービスから口グ記録を有効にするを参照してください。

CloudWatch Logs にアクセスできない場合は、「Unable to access the CloudWatch Logs」で情報を確認してください。

AWS Step Functions 他のサービスとの併用

他の API AWS のサービス との連携やサードパーティ API の呼び出しについて学んでください。AWS Step Functions

トピック

- AWS 他のサービスに電話してください。
- AWS SDK サービスインテグレーション
- Step Functions 用統合最適化
- サードパーティーの API を呼び出す
- サービス統合パターン
- サービス API にパラメータを渡す
- サポートされている AWS SDK インテグレーションの変更ログ

AWS 他のサービスに電話してください。

AWS Step Functions AWS サービスと統合し、ワークフローから各サービスの API アクションを呼び出すことができます。Step Functions のAWS SDK インテグレーションを使用すると、200 AWS を超えるサービスのいずれかをステートマシンから直接呼び出すことができ、9,000 を超える API アクションにアクセスできます。または、それぞれがワークフローに特別な機能を提供するようにカスタマイズされた Step Functions の最適化統合を使用できます。一部の API アクションは、両方のタイプの統合で使用できます。この場合は、最適化された統合を使用することをお勧めします。

Amazon ステートメント言語で `Task` 状態からこれらのサービスを直接調整します。例えば、Step Functions を使用して他のサービスを呼び出すことができます。

- 関数を呼び出す。 AWS Lambda
- AWS Batch ジョブを実行し、その結果に基づいてさまざまなアクションを実行する。
- Amazon DynamoDB から項目を挿入するか、項目を取得する
- Amazon Elastic Container Service (Amazon ECS) タスクを実行し、完了するまで待機します。
- Amazon Simple Notification Service (Amazon SNS) でトピックを発行します。
- Amazon Simple Queue Service (Amazon SQS) にメッセージを送信します。
- Amazon AWS Glue SageMaker ンの求人を管理します。

- Amazon EMR ジョブを実行するためのワークフローを構築します。
- AWS Step Functions ワークフロー実行を開始する。

最適化された統合

最適化された統合は、ワークフローコンテキストに特別な機能を提供するために Step Functions によってカスタマイズされています。例えば、Lambda `Invoke` は、API 出力をエスケープされた JSON から JSON オブジェクトに変換します。AWS Batch`SubmitJob` はジョブが完了するまで実行を一時停止できます。2018 年に最適化された統合が初めてリリースされ、現在 50 以上の API があります。

AWS SDK インテグレーション

AWS SDK インテグレーションは、SDK を使用する標準 API 呼び出しとまったく同じように機能します。 AWS これにより、200 AWS を超えるサービスの 9,000 を超える API をステートマシン定義から直接呼び出すことができます。 AWS SDK インテグレーションは 2021 年にリリースされました。

統合パターンのサポート

標準ワークフローとエクスプレスワークフローは同じ統合をサポートしていますが、同じ統合パターンはサポートしていません。

- 最適化された統合パターンのサポートは、各統合ごとに異なります。
- エクスプレスワークフローは、 Job 実行 (.sync) またはコールバックを待機 (. waitForTaskトークン)。
- 詳細については、「標準ワークフロー対 Express ワークフロー」を参照してください。

Standard Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	Amazon SageMaker	✓	✓	

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS SDK インテグレーション	200 以上	✓		✓

Express Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	Amazon SageMaker	✓		
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS SDK インテグレーション	200 以上	✓		

クロスアカウントアクセス

Step Functions は、 AWS アカウント ワークフロー内のさまざまな設定で設定されたリソースへのクロスアカウントアクセスを提供します。Step Functions サービスインテグレーションを使用すると、 AWS AWS のサービス リソースベースのポリシーやクロスアカウントコールをサポートしていない場合でも、あらゆるクロスアカウントリソースを呼び出すことができます。

詳細については、「AWS アカウント ワークフロー内の他のリソースへのアクセス」を参照してください。

AWS SDK サービスインテグレーション

AWS Step Functions と統合され AWS のサービス、ワークフローから各サービスの API アクションを呼び出すことができます。Step Functions のAWS SDK インテグレーションを使用して、ステートマシンからほとんどすべての AWS のサービス API アクションを呼び出すことができます。Step Functions の最適化統合を使用することもできます。このそれぞれが、ワークフローに特別な機能を提供するようカスタマイズされています。

一部のサービスや SDK は、一時的または永続的に AWS SDK インテグレーションとして使用できない場合があります。最近リリースされたサービスでは、今後のアップデートまで、SDK インタラクションを利用できない場合があります。一部のサービスでは、顧客固有のエンドポイントの指定など、カスタマイズされた構成が必要であり、最適化された統合に適しています。その他の SDK は、オーディオやビデオのストリーミングなど、ワークフローでの使用には適していません。最後に、一部のサービスは Step Functions によって実行される特定の内部検証に合格するまで保留される場合があります。

> **Tip**
>
> AWS SDK インテグレーションを使用するワークフローの例をデプロイするには AWS アカウント、『ワークショップ』の「モジュール 9- AWS SDK サービスインテグレーション」を参照してください。 AWS Step Functions

トピック

- AWS SDK サービスインテグレーションを使用する
- サポートされている AWS SDK サービス統合
- サポートされているサービスでサポートされていない API アクション
- 非推奨の AWS SDK サービス統合

AWS SDK サービスインテグレーションを使用する

AWS SDK 統合を使用するには、サービス名と API 呼び出しを指定し、オプションでサービス統合パターンを指定します。

> **Note**
>
> - ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。
> - Amazon EC2 の `DescribeLaunchTemplateVersions` API アクションなど、列挙パラメータを受け入れる API アクションの場合は、パラメータ名を複数形にして指定します。例えば、`DescribeLaunchTemplateVersions` API アクションの `Filter.N` パラメータには `Filters` を指定してください。

`Resource`タスクステートのフィールドにある Amazon ステート言語から AWS SDK サービスを直接呼び出すことができます。このためには、次の構文を使用します。

`arn:aws:states:::aws-sdk:`*`serviceName`*`:`*`apiAction`*`.`*`[serviceIntegrationPattern]`*

例えば、Amazon EC2 では、`arn:aws:states:::aws-sdk:ec2:describeInstances` を使用できます。これは、Amazon EC2 describeInstances API コールで定義されているような出力を返します。

AWS SDK インテグレーションでエラーが発生した場合、結果の Error フィールドは、サービス名とエラー名をピリオド文字で区切ったものになります。*`ServiceName`*`.`*`ErrorName`*サービス名とエラー名は両方とも Pascal ケースです。サービス名は、タスク状態の [リソース] フィールドにも小文字で表示されます。ターゲットサービスの API リファレンスドキュメントで潜在的なエラー名が見つかります。

たとえば、`arn:aws:states:::aws-sdk:acmpca:deleteCertificateAuthority` AWS SDK インテグレーションを使用できます。AWS Private Certificate Authority API リファレンスには、`DeleteCertificateAuthority` API アクションの結果として、例えば `ResourceNotFoundException` が発生する可能性があることが示されています。このエラーを処理するには、`AcmPca.ResourceNotFoundException` タスク状態の Retry またはCatcher で Error

を指定します。Step Functions 内のエラー処理の詳細については、Step Functions のエラー処理 を参照してください。

Step Functions は AWS SDK インテグレーションの IAM ポリシーを自動生成できません。ステートマシンを作成した後、IAM コンソールに移動してロールポリシーを設定する必要があります。詳細については、統合サービスの IAM ポリシー を参照してください。

SDK AWS インテグレーションの使用方法の例については、AWS SDK サービスインテグレーションを使用して Amazon S3 バケット情報を収集するチュートリアルを参照してください。

サポートされている AWS SDK サービス統合

次の表は、Step Functions がサポートする AWS SDK サービスインテグレーションの一覧です。*Task* 状態リソース列には、左側のサービス名列に示されたサービスの統合を使用するときの、特定の API アクションを呼び出すための構文がリストされています。サポート日列には、サービス統合がサポートされた日付に関する情報が表示されます。また、右側の例外プレフィックス列には、各サービス統合の例外プレフィックスが一覧表示されます。これらの例外プレフィックスは、Step Functions と AWS SDK サービスの統合を誤って実行したときに生成される例外に含まれています。

> **Note**
>
> - *** のマークが付いたサービスには、現時点では Step Functions ではサポートされていない API アクションがあります。サービスでサポートされていないアクションについては、サポートされているサービスでサポートされていない API アクション 表を参照してください。
> - AWS SDK 統合のサポートを拡大するためにリリースされるたびに行われる更新については、を参照してください。サポートされている AWS SDK インテグレーションの変更ログ

> **Important**
>
> API アクションのサポートは四半期ごとにリリースされます。新しいパラメータなど、すでにサポートされているアクションの更新は、すぐには利用できない場合があります。

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS AppFabric	arn:aws:states:::aws-sdk:appfabric: *[apiAction]*	2024 年 1 月 18 日	AppFabric
B2B Data Interchange	arn:aws:states:::aws-sdk:b2bi: *[apiAction]*	2024 年 1 月 18 日	B2Bi
AWS Data Exports	arn:aws:states:::aws-sdk:bcmdataexports: *[apiAction]*	2024 年 1 月 18 日	BcmDataExports
Amazon Bedrock	arn:aws:states:::aws-sdk:bedrock: *[apiAction]*	2024 年 1 月 18 日	Bedrock
Amazon Bedrock Agents	arn:aws:states:::aws-sdk:bedrockagent: *[apiAction]*	2024 年 1 月 18 日	BedrockAgent
Amazon Bedrock Runtime Agents	arn:aws:states:::aws-sdk:bedrockagentruntime: *[apiAction]*	2024 年 1 月 18 日	BedrockAgentRuntime
Amazon Bedrock Runtime	arn:aws:states:::aws-sdk:bedrockruntime: *[apiAction]*	2024 年 1 月 18 日	BedrockRuntime
AWS Clean Rooms	arn:aws:states:::aws-sdk:cleanroomsml: *[apiAction]*	2024 年 1 月 18 日	CleanRoomsMl
Amazon CloudFront KeyValueStore	arn:aws:states:::aws-sdk:cl	2024 年 1 月 18 日	CloudFrontKeyValueStore

サービス名	Task ステートリソース	サポート日	例外プレフィックス
	`oudfrontkeyvaluestore: [apiAction]`		
CodeGuru Security	`arn:aws:states:::aws-sdk:codegurusecurity: [apiAction]`	2024 年 1 月 18 日	CodeGuruSecurity
AWS Cost Optimization Hub	`arn:aws:states:::aws-sdk:costoptimizationhub: [apiAction]`	2024 年 1 月 18 日	CostOptimizationHub
Amazon DataZone	`arn:aws:states:::aws-sdk:datazone: [apiAction]`	2024 年 1 月 18 日	DataZone
Amazon EKS Auth	`arn:aws:states:::aws-sdk:eksauth: [apiAction]`	2024 年 1 月 18 日	EksAuth
AWS Entity Resolution	`arn:aws:states:::aws-sdk:entityresolution: [apiAction]`	2024 年 1 月 18 日	EntityResolution
AWS 無料利用枠	`arn:aws:states:::aws-sdk:freetier: [apiAction]`	2024 年 1 月 18 日	FreeTier
Amazon Inspector Scan	`arn:aws:states:::aws-sdk:inspectorscan: [apiAction]`	2024 年 1 月 18 日	InspectorScan
AWS Launch Wizard	`arn:aws:states:::aws-sdk:launchwizard: [apiAction]`	2024 年 1 月 18 日	LaunchWizard

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Managed Blockchain Query	`arn:aws:states:::aws-sdk:managedblockchainquery: [apiAction]`	2024 年 1 月 18 日	ManagedBlockchainQuery
AWS Elemental MediaPackage V2	`arn:aws:states:::aws-sdk:mediapackagev2: [apiAction]`	2024 年 1 月 18 日	MediaPackageV2
AWS HealthImaging	`arn:aws:states:::aws-sdk:medicalimaging: [apiAction]`	2024 年 1 月 18 日	MedicalImaging
Network Manager	`arn:aws:states:::aws-sdk:networkmanager: [apiAction]`	2024 年 1 月 18 日	NetworkManager
AWS Payment Cryptography	`arn:aws:states:::aws-sdk:paymentcryptography: [apiAction]`	2024 年 1 月 18 日	PaymentCryptography
AWS Payment Cryptography Data	`arn:aws:states:::aws-sdk:paymentcryptographydata: [apiAction]`	2024 年 1 月 18 日	PaymentCryptographyData
AWS Private CA Connector for Active Directory	`arn:aws:states:::aws-sdk:pcaconnectorad: [apiAction]`	2024 年 1 月 18 日	PcaConnectorAd
Amazon Q Business	`arn:aws:states:::aws-sdk:qbusiness: [apiAction]`	2024 年 1 月 18 日	QBusiness

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Q Connect	`arn:aws:states:::aws-sdk:qconnect: [apiAction]`	2024 年 1 月 18 日	QConnect
AWS re:Post	`arn:aws:states:::aws-sdk:repostspace: [apiAction]`	2024 年 1 月 18 日	Repostspace
Amazon Timestream Query	`arn:aws:states:::aws-sdk:timestreamquery: [apiAction]`	2024 年 1 月 18 日	TimestreamQuery
Amazon Timestream Write	`arn:aws:states:::aws-sdk:timestreamwrite: [apiAction]`	2024 年 1 月 18 日	TimestreamWrite
Trusted Advisor	`arn:aws:states:::aws-sdk:trustedadvisor: [apiAction]`	2024 年 1 月 18 日	TrustedAdvisor
Verified Permissions	`arn:aws:states:::aws-sdk:verifiedpermissions: [apiAction]`	2024 年 1 月 18 日	VerifiedPermissions
Amazon WorkSpaces Thin Client	`arn:aws:states:::aws-sdk:workspacesthinclient: [apiAction]`	2024 年 1 月 18 日	WorkSpacesThinClient
AWS CloudTrail Data	`arn:aws:states:::aws-sdk:cloudtraildata: [apiAction]`	2023 年 6 月 16 日	CloudTrailData

サービス名	**Task** ステートリソース	サポート日	例外プレフィックス
Amazon CloudWatch Internet Monitor	`arn:aws:states:::aws-sdk:internetmonitor: [apiAction]`	2023 年 6 月 16 日	InternetMonitor
Amazon Interactive Video Service RealTime	`arn:aws:states:::aws-sdk:ivsrealtime: [apiAction]`	2023 年 6 月 16 日	IvsRealTime
AWS IoT TwinMaker	`arn:aws:states:::aws-sdk:iottwinmaker: [apiAction]`	2023 年 6 月 16 日	IoTTwinMaker
Amazon OpenSearch Ingestion	`arn:aws:states:::aws-sdk:osis: [apiAction]`	2023 年 6 月 16 日	Osis
AWS Telco Network Builder	`arn:aws:states:::aws-sdk: tnb:[apiAction]`	2023 年 6 月 16 日	Tnb
Amazon VPC Lattice	`arn:aws:states:::aws-sdk:vpclattice: [apiAction]`	2023 年 6 月 16 日	VpcLattice
Amazon Chime Media Pipelines	`arn:aws:states:::aws-sdk:chimesdkmediapipelines: [apiAction]`	2023 年 2 月 17 日	ChimeSdkMediaPipelines
Amazon Chime Voice	`arn:aws:states:::aws-sdk:chimesdkvoice: [apiAction]`	2023 年 2 月 17 日	ChimeSdkVoice

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon CodeCatalyst	arn:aws:states:::aws-sdk:codecatalyst: *[apiAction]*	2023 年 2 月 17 日	CodeCatalyst
Amazon Connect Cases	arn:aws:states:::aws-sdk:connectcases: *[apiAction]*	2023 年 2 月 17 日	ConnectCases
Amazon DocumentDB Elastic Clusters	arn:aws:states:::aws-sdk:docdbelastic: *[apiAction]*	2023 年 2 月 17 日	DocDbElastic
Amazon EMR Serverless	arn:aws:states:::aws-sdk:emrserverless: *[apiAction]*	2023 年 2 月 17 日	EmrServerless
Amazon IVS Chat	arn:aws:states:::aws-sdk:ivs: *[apiAction]*	2023 年 2 月 17 日	Ivs
Amazon Kendra Intelligent Ranking	arn:aws:states:::aws-sdk:kendraranking: *[apiAction]*	2023 年 2 月 17 日	KendraRanking
AWS HealthOmics	arn:aws:states:::aws-sdk:omics: *[apiAction]*	2023 年 2 月 17 日	Omics
Amazon Redshift Serverless	arn:aws:states:::aws-sdk:redshiftserverless: *[apiAction]*	2023 年 2 月 17 日	RedshiftServerless
Amazon Security Lake	arn:aws:states:::aws-sdk:securitylake: *[apiAction]*	2023 年 2 月 17 日	SecurityLake

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Backup Storage	`arn:aws:states:::aws-sdk:backupstorage: [apiAction]`	2023 年 2 月 17 日	BackupStorage
AWS Clean Rooms	`arn:aws:states:::aws-sdk:cleanrooms: [apiAction]`	2023 年 2 月 17 日	CleanRooms
AWS Control Tower	`arn:aws:states:::aws-sdk:controltower: [apiAction]`	2023 年 2 月 17 日	ControlTower
AWS Health	`arn:aws:states:::aws-sdk:health: [apiAction]`	2023 年 2 月 17 日	Health
AWS IoT FleetWise	`arn:aws:states:::aws-sdk:iotfleetwise: [apiAction]`	2023 年 2 月 17 日	IotFleetWise
AWS IoT RoboRunner	`arn:aws:states:::aws-sdk:iotroborunner: [apiAction]`	2023 年 2 月 17 日	IotRoboRunner
AWS Mainframe Modernization	`arn:aws:states:::aws-sdk:m2: [apiAction]`	2023 年 2 月 17 日	M2
AWS Migration Hub Orchestrator	`arn:aws:states:::aws-sdk:migrationhuborchestrator: [apiAction]`	2023 年 2 月 17 日	Migration HubOrchestrator

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Private 5G	`arn:aws:states:::aws-sdk:privatenetworks: [apiAction]`	2023 年 2 月 17 日	PrivateNetworks
AWS Resource Explorer	`arn:aws:states:::aws-sdk:resourceexplorer2: [apiAction]`	2023 年 2 月 17 日	ResourceExplorer2
AWS SimSpace Weaver	`arn:aws:states:::aws-sdk:simspaceweaver: [apiAction]`	2023 年 2 月 17 日	SimSpaceWeaver
AWS Support App	`arn:aws:states:::aws-sdk:supportapp: [apiAction]`	2023 年 2 月 17 日	SupportApp
CloudWatch Observability Access Manager	`arn:aws:states:::aws-sdk:oam: [apiAction]`	2023 年 2 月 17 日	Oam
EventBridge Pipes	`arn:aws:states:::aws-sdk:pipes: [apiAction]`	2023 年 2 月 17 日	Pipes
EventBridge Scheduler	`arn:aws:states:::aws-sdk:scheduler: [apiAction]`	2023 年 2 月 17 日	Scheduler
IAM Roles Anywhere	`arn:aws:states:::aws-sdk:rolesanywhere: [apiAction]`	2023 年 2 月 17 日	RolesAnywhere

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Kinesis Video WebRTC Storage	arn:aws:states:::aws-sdk:kinesisvideowebrtcstorage: *[apiAction]*	2023 年 2 月 17 日	KinesisVideoWebRtcStorage
License Manager Linux Subscriptions	arn:aws:states:::aws-sdk:licensemanagerlinuxsubscriptions: *[apiAction]*	2023 年 2 月 17 日	LicenseManagerLinuxSubscriptions
License Manager User Subscriptions	arn:aws:states:::aws-sdk:licensemanagerusersubscriptions: *[apiAction]*	2023 年 2 月 17 日	LicenseManagerUserSubscriptions
OpenSearch Serverless	arn:aws:states:::aws-sdk:opensearchserverless: *[apiAction]*	2023 年 2 月 17 日	OpenSearchServerless
Route 53 ARC Zonal Shift	arn:aws:states:::aws-sdk:arczonalshift: *[apiAction]*	2023 年 2 月 17 日	ArcZonalShift
SageMaker Geospatial	arn:aws:states:::aws-sdk:sagemakergeospatial: *[apiAction]*	2023 年 2 月 17 日	SageMaker Geospatial
SageMaker Metrics	arn:aws:states:::aws-sdk:sagemakermetrics: *[apiAction]*	2023 年 2 月 17 日	SageMakerMetrics

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Systems Manager for SAP	arn:aws:states:::aws-sdk:ssmsap: *[apiAction]*	2023 年 2 月 17 日	SsmSap
AWS Account Management	arn:aws:states:::aws-sdk:account: *[apiAction]*	2022 年 4 月 19 日	Account
AWS Amplify	arn:aws:states:::aws-sdk:amplify: *[apiAction]*	2021 年 9 月 30 日	Amplify
AWS App Mesh	arn:aws:states:::aws-sdk:appmesh: *[apiAction]*	2021 年 9 月 30 日	AppMesh
AWS App Runner	arn:aws:states:::aws-sdk:apprunner: *[apiAction]*	2021 年 9 月 30 日	AppRunner
AWS AppConfig	arn:aws:states:::aws-sdk:appconfig: *[apiAction]*	2021 年 9 月 30 日	AppConfig
AWS AppConfig Data	arn:aws:states:::aws-sdk:appconfigdata: *[apiAction]*	2022 年 4 月 19 日	AppConfigData
AWS AppSync	arn:aws:states:::aws-sdk:appsync: *[apiAction]*	2021 年 9 月 30 日	AppSync

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Application Discovery Service	arn:aws:states:::aws-sdk:applicationdiscovery: *[apiAction]* ***	2021 年 9 月 30 日	ApplicationDiscovery
AWS Application Migration Service	arn:aws:states:::aws-sdk:mgn: *[apiAction]*	2021 年 9 月 30 日	Mgn
AWS Audit Manager	arn:aws:states:::aws-sdk:auditmanager: *[apiAction]*	2021 年 9 月 30 日	AuditManager
AWS Auto Scaling Plans	arn:aws:states:::aws-sdk:autoscalingplans: *[apiAction]*	2021 年 9 月 30 日	AutoScalingPlans
AWS Backup	arn:aws:states:::aws-sdk:backup: *[apiAction]*	2021 年 9 月 30 日	Backup
AWS Backup gateway	arn:aws:states:::aws-sdk:backupgateway: *[apiAction]*	2022 年 4 月 19 日	BackupGateway
AWS Batch	arn:aws:states:::aws-sdk:batch: *[apiAction]*	2021 年 9 月 30 日	Batch
AWS Billing Conductor	arn:aws:states:::aws-sdk:billingconductor: *[apiAction]*	2022 年 7 月 26 日	Billingconductor

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Budgets	`arn:aws:states:::aws-sdk:budgets: [apiAction]`	2021 年 9 月 30 日	Budgets
AWS Certificate Manager	`arn:aws:states:::aws-sdk:acm: [apiAction]`	2021 年 9 月 30 日	Acm
AWS Private Certificate Authority	`arn:aws:states:::aws-sdk:acmpca: [apiAction]`	2021 年 9 月 30 日	AcmPca
AWS Cloud Map	`arn:aws:states:::aws-sdk:servicediscovery: [apiAction]`	2021 年 9 月 30 日	ServiceDiscovery
AWS Cloud9	`arn:aws:states:::aws-sdk:cloud9: [apiAction]`	2021 年 9 月 30 日	Cloud9
AWS CloudFormation	`arn:aws:states:::aws-sdk:cloudformation: [apiAction]`	2021 年 9 月 30 日	CloudFormation
AWS CloudHSM	`arn:aws:states:::aws-sdk:cloudhsm: [apiAction]`	2021 年 9 月 30 日	CloudHsm
AWS CloudHSM	`arn:aws:states:::aws-sdk:cloudhsmv2: [apiAction]`	2021 年 9 月 30 日	CloudHsmV2
AWS CloudTrail	`arn:aws:states:::aws-sdk:cloudtrail: [apiAction]`	2021 年 9 月 30 日	CloudTrail

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Cloud Control	`arn:aws:states:::aws-sdk:cloudcontrol: [apiAction]`	2022 年 4 月 19 日	CloudControl
AWS CodeBuild	`arn:aws:states:::aws-sdk:codebuild: [apiAction]`	2021 年 9 月 30 日	CodeBuild
AWS CodeCommit	`arn:aws:states:::aws-sdk:codecommit: [apiAction]`	2021 年 9 月 30 日	CodeCommit
AWS CodeDeploy	`arn:aws:states:::aws-sdk:codedeploy: [apiAction]` ***	2021 年 9 月 30 日	CodeDeploy
AWS CodePipeline	`arn:aws:states:::aws-sdk:codepipeline: [apiAction]`	2021 年 9 月 30 日	CodePipeline
AWS CodeStar	`arn:aws:states:::aws-sdk:codestar: [apiAction]`	2021 年 9 月 30 日	CodeStar
AWS CodeStar	`arn:aws:states:::aws-sdk:codestarnotifications: [apiAction]`	2021 年 9 月 30 日	CodestarNotifications
AWS CodeStar	`arn:aws:states:::aws-sdk:codestarconnections: [apiAction]`	2021 年 9 月 30 日	CodeStarConnections

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Compute Optimizer	`arn:aws:states:::aws-sdk:computeoptimizer: [apiAction]`	2021 年 9 月 30 日	ComputeOptimizer
AWS Config	`arn:aws:states:::aws-sdk:config: [apiAction]`	2021 年 9 月 30 日	Config
AWS Cost Explorer Service	`arn:aws:states:::aws-sdk:costexplorer: [apiAction]`	2021 年 9 月 30 日	CostExplorer
AWS Cost and Usage Report	`arn:aws:states:::aws-sdk:costandusagereport: [apiAction]`	2021 年 9 月 30 日	CostAndUsageReport
AWS Data Exchange	`arn:aws:states:::aws-sdk:dataexchange: [apiAction]`	2021 年 9 月 30 日	DataExchange
AWS Data Pipeline	`arn:aws:states:::aws-sdk:datapipeline: [apiAction]`	2021 年 9 月 30 日	DataPipeline
AWS DataSync	`arn:aws:states:::aws-sdk:datasync: [apiAction]`	2021 年 9 月 30 日	DataSync
AWS Database Migration Service	`arn:aws:states:::aws-sdk:databasemigration: [apiAction]`	2021 年 9 月 30 日	DatabaseMigration
AWS Device Farm	`arn:aws:states:::aws-sdk:devicefarm: [apiAction]`	2021 年 9 月 30 日	DeviceFarm

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Direct Connect	`arn:aws:states:::aws-sdk:directconnect: [apiAction]` ***	2021 年 9 月 30 日	DirectConnect
AWS Directory Service	`arn:aws:states:::aws-sdk:directory: [apiAction]`	2021 年 9 月 30 日	Directory
AWS EC2 Instance Connect	`arn:aws:states:::aws-sdk:ec2instanceconnect: [apiAction]`	2021 年 9 月 30 日	Ec2InstanceConnect
AWS Elastic Beanstalk	`arn:aws:states:::aws-sdk:elasticbeanstalk: [apiAction]`	2021 年 9 月 30 日	ElasticBeanstalk
AWS Elemental MediaLive	`arn:aws:states:::aws-sdk:medialive: [apiAction]`	2021 年 9 月 30 日	MediaLive
AWS Elemental MediaPackage	`arn:aws:states:::aws-sdk:mediapackage: [apiAction]` ***	2021 年 9 月 30 日	MediaPackage
AWS Elemental MediaPackage VOD	`arn:aws:states:::aws-sdk:mediapackagevod: [apiAction]`	2021 年 9 月 30 日	MediaPackageVod
AWS Elemental MediaStore	`arn:aws:states:::aws-sdk:mediastore: [apiAction]`	2021 年 9 月 30 日	MediaStore
AWS Fault Injection Service	`arn:aws:states:::aws-sdk:fis: [apiAction]`	2021 年 9 月 30 日	Fis

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Firewall Manager	arn:aws:states:::aws-sdk:fms: *[apiAction]*	2021 年 9 月 30 日	Fms
AWS Glue	arn:aws:states:::aws-sdk:glue: *[apiAction]*	2021 年 9 月 30 日	Glue
AWS Glue DataBrew	arn:aws:states:::aws-sdk:databrew: *[apiAction]*	2021 年 9 月 30 日	DataBrew
AWS IoT Greengrass	arn:aws:states:::aws-sdk:greengrass: *[apiAction]*	2021 年 9 月 30 日	Greengrass
AWS Ground Station	arn:aws:states:::aws-sdk:groundstation: *[apiAction]*	2021 年 9 月 30 日	GroundStation
AWS Identity and Access Management	arn:aws:states:::aws-sdk:iam: *[apiAction]*	2021 年 9 月 30 日	Iam
AWS IoT	arn:aws:states:::aws-sdk:iot: *[apiAction]* ***	2021 年 9 月 30 日	Iot
AWS IoT 1-Click	arn:aws:states:::aws-sdk:iot1clickprojects: *[apiAction]*	2021 年 9 月 30 日	Iot1ClickProjects
AWS IoT Analytics	arn:aws:states:::aws-sdk:iotanalytics: *[apiAction]*	2021 年 9 月 30 日	IoTAnalytics

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS IoT Core Device Advisor	`arn:aws:states:::aws-sdk:iotdeviceadvisor: [apiAction]` *** —	2021 年 9 月 30 日	IotDeviceAdvisor
AWS IoT Events	`arn:aws:states:::aws-sdk:iotevents: [apiAction]`	2021 年 9 月 30 日	IotEvents
AWS IoT Events Data	`arn:aws:states:::aws-sdk:ioteventsdata: [apiAction]`	2021 年 9 月 30 日	IotEventsData
AWS IoT Fleet Hub	`arn:aws:states:::aws-sdk:iotfleethub: [apiAction]`	2021 年 9 月 30 日	IoTFleetHub
AWS IoT Greengrass Version 2	`arn:aws:states:::aws-sdk:greengrassv2: [apiAction]`	2021 年 9 月 30 日	GreengrassV2
AWS IoT ジョブデータ Plane	`arn:aws:states:::aws-sdk:iotjobsdataplane: [apiAction]`	2021 年 9 月 30 日	IotJobsDataPlane
AWS IoT Secure Tunneling	`arn:aws:states:::aws-sdk:iotsecuretunneling: [apiAction]`	2021 年 9 月 30 日	IoTSecure Tunneling
AWS IoT SiteWise	`arn:aws:states:::aws-sdk:iotsitewise: [apiAction]`	2021 年 9 月 30 日	IoTSiteWise
AWS IoT Wireless	`arn:aws:states:::aws-sdk:iotwireless: [apiAction]`	2021 年 9 月 30 日	IotWireless

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Key Management Service	`arn:aws:states:::aws-sdk:kms:` *[apiAction]*	2021 年 9 月 30 日	Kms
AWS Lake Formation	`arn:aws:states:::aws-sdk:lakeformation:` *[apiAction]*	2021 年 9 月 30 日	LakeFormation
AWS Lambda	`arn:aws:states:::aws-sdk:lambda:` *[apiAction]* *** —	2021 年 9 月 30 日	Lambda
AWS License Manager	`arn:aws:states:::aws-sdk:licensemanager:` *[apiAction]*	2021 年 9 月 30 日	LicenseManager
AWS Marketplace	`arn:aws:states:::aws-sdk:marketplacecatalog:` *[apiAction]*	2021 年 9 月 30 日	MarketplaceCatalog
AWS Marketplace Commerce Analytics	`arn:aws:states:::aws-sdk:marketplacecommerceanalytics:` *[apiAction]*	2021 年 9 月 30 日	MarketplaceCommerceAnalytics
AWS Marketplace Entitlement Service	`arn:aws:states:::aws-sdk:marketplaceentitlement:` *[apiAction]*	2021 年 9 月 30 日	MarketplaceEntitlement
AWS Elemental MediaTailor	`arn:aws:states:::aws-sdk:mediatailor:` *[apiAction]*	2021 年 9 月 30 日	MediaTailor

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Migration Hub	`arn:aws:states:::aws-sdk:migrationhub: [apiAction]`	2021 年 9 月 30 日	MigrationHub
AWS Migration Hub Config	`arn:aws:states:::aws-sdk:migrationhubconfig: [apiAction]`	2021 年 9 月 30 日	MigrationHubConfig
AWS Migration Hub Strategy Recommendations	`arn:aws:states:::aws-sdk:migrationhubstrategy: [apiAction]`	2022 年 4 月 19 日	MigrationHubStrategy
AWS Mobile	`arn:aws:states:::aws-sdk:mobile: [apiAction]`	2021 年 9 月 30 日	
AWS Network Firewall	`arn:aws:states:::aws-sdk:networkfirewall: [apiAction]`	2021 年 9 月 30 日	NetworkFirewall
AWS OpsWorks	`arn:aws:states:::aws-sdk:opsworks: [apiAction]`	2021 年 9 月 30 日	OpsWorks
AWS OpsWorks CM	`arn:aws:states:::aws-sdk:opsworkscm: [apiAction]`	2021 年 9 月 30 日	OpsWorksCm
AWS Organizations	`arn:aws:states:::aws-sdk:organizations: [apiAction]`	2021 年 9 月 30 日	Organizations

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Outposts	arn:aws:states:::aws-sdk:outposts: *[apiAction]*	2021 年 9 月 30 日	Outposts
AWS Panorama	arn:aws:states:::aws-sdk:panorama: *[apiAction]*	2022 年 4 月 19 日	Panorama
Amazon Relational Database Service Performance Insights	arn:aws:states:::aws-sdk:pi: *[apiAction]*	2021 年 9 月 30 日	Pi
AWS の料金表	arn:aws:states:::aws-sdk:pricing: *[apiAction]*	2021 年 9 月 30 日	Pricing
Amazon Relational Database Service	arn:aws:states:::aws-sdk:rdsdata: *[apiAction]* ***	2021 年 9 月 30 日	RdsData
AWS Resilience Hub	arn:aws:states:::aws-sdk:resiliencehub: *[apiAction]*	2022 年 4 月 19 日	Resiliencehub
AWS Resource Access Manager	arn:aws:states:::aws-sdk:ram: *[apiAction]*	2021 年 9 月 30 日	Ram
AWS Resource Groups	arn:aws:states:::aws-sdk:resourcegroups: *[apiAction]*	2021 年 9 月 30 日	ResourceGroups

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Resource Groups Tagging API	`arn:aws:states:::aws-sdk:resourcegroupstaggingapi: [apiAction]`	2021 年 9 月 30 日	ResourceGroupsTaggingApi
AWS RoboMaker	`arn:aws:states:::aws-sdk:robomaker: [apiAction]`	2021 年 9 月 30 日	RoboMaker
AWS IAM Identity Center	`arn:aws:states:::aws-sdk:identitystore: [apiAction]`	2021 年 9 月 30 日	Identitystore
IAM Identity Center OIDC	`arn:aws:states:::aws-sdk:ssooidc: [apiAction]`	2021 年 9 月 30 日	SsoOidc
AWS Secrets Manager	`arn:aws:states:::aws-sdk:secretsmanager: [apiAction]`	2021 年 9 月 30 日	SecretsManager
AWS Security Token Service	`arn:aws:states:::aws-sdk:sts: [apiAction]` [***] —	2021 年 9 月 30 日	Sts
AWS Security Hub	`arn:aws:states:::aws-sdk:securityhub: [apiAction]`	2021 年 9 月 30 日	SecurityHub
AWS Server Migration Service	`arn:aws:states:::aws-sdk:sms: [apiAction]`	2021 年 9 月 30 日	Sms

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Service Catalog	arn:aws:states:::aws-sdk:servicecatalog: *[apiAction]*	2021 年 9 月 30 日	ServiceCatalog
AWS Service Catalog AppRegistry	arn:aws:states:::aws-sdk:servicecatalogappregistry: *[apiAction]*	2021 年 9 月 30 日	ServiceCatalogAppRegistry
AWS Shield	arn:aws:states:::aws-sdk:shield: *[apiAction]* *** —	2021 年 9 月 30 日	Shield
AWS Signer	arn:aws:states:::aws-sdk:signer: *[apiAction]*	2021 年 9 月 30 日	Signer
IAM Identity Center	arn:aws:states:::aws-sdk:sso: *[apiAction]*	2021 年 9 月 30 日	Sso
IAM Identity Center Admin	arn:aws:states:::aws-sdk:ssoadmin: *[apiAction]*	2021 年 9 月 30 日	SsoAdmin
AWS Step Functions	arn:aws:states:::aws-sdk:sfn: *[apiAction]*	2021 年 9 月 30 日	Sfn
AWS Storage Gateway	arn:aws:states:::aws-sdk:storagegateway: *[apiAction]*	2021 年 9 月 30 日	StorageGateway

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Support	arn:aws:states:::aws-sdk:support: *[apiAction]*	2021 年 9 月 30 日	Support
AWS Systems Manager Incident Manager	arn:aws:states:::aws-sdk:ssmincidents: *[apiAction]*		SsmIncidents
AWS Transfer Family	arn:aws:states:::aws-sdk:transfer: *[apiAction]*	2021 年 9 月 30 日	Transfer
AWS WAF	arn:aws:states:::aws-sdk:waf: *[apiAction]*	2021 年 9 月 30 日	Waf
AWS WAF Regional	arn:aws:states:::aws-sdk:wafregional: *[apiAction]*	2021 年 9 月 30 日	WafRegional
AWS WAFV2	arn:aws:states:::aws-sdk:wafv2: *[apiAction]*	2021 年 9 月 30 日	Wafv2
AWS Well-Architected Tool	arn:aws:states:::aws-sdk:wellarchitected: *[apiAction]*	2021 年 9 月 30 日	WellArchitected
AWS X-Ray	arn:aws:states:::aws-sdk:xray: *[apiAction]*	2021 年 9 月 30 日	XRay

サービス名	Task ステートリソース	サポート日	例外プレフィックス
AWS Marketplace Metering Service	arn:aws:states:::aws-sdk:marketplacemetering: *[apiAction]*	2021 年 9 月 30 日	MarketplaceMetering
AWS Serverless Application Repository	arn:aws:states:::aws-sdk:serverlessapplicationrepository: *[apiAction]*	2021 年 9 月 30 日	ServerlessApplicationRepository
AWS Identity and Access Management Access Analyzer	arn:aws:states:::aws-sdk:accessanalyzer: *[apiAction]*	2021 年 9 月 30 日	AccessAnalyzer
Alexa for Business	arn:aws:states:::aws-sdk:alexaforbusiness: *[apiAction]*	2021 年 9 月 30 日	AlexaForBusiness
Amazon API Gateway	arn:aws:states:::aws-sdk:apigateway: *[apiAction]*	2021 年 9 月 30 日	ApiGateway
Amazon API Gateway	arn:aws:states:::aws-sdk:apigatewayv2: *[apiAction]*	2021 年 9 月 30 日	ApiGatewayV2
Amazon AppIntegrations	arn:aws:states:::aws-sdk:appintegrations: *[apiAction]*	2021 年 9 月 30 日	AppIntegrations
Amazon AppStream 2.0	arn:aws:states:::aws-sdk:appstream: *[apiAction]*	2021 年 9 月 30 日	AppStream

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon AppFlow	`arn:aws:states:::aws-sdk:appflow: [apiAction]`	2021 年 9 月 30 日	Appflow
Amazon Athena	`arn:aws:states:::aws-sdk:athena: [apiAction]`	2021 年 9 月 30 日	Athena
Amazon Augmented AI	`arn:aws:states:::aws-sdk:sagemakera2iruntime: [apiAction]`	2021 年 9 月 30 日	SageMaker A2IRuntime
Amazon Braket	`arn:aws:states:::aws-sdk:braket: [apiAction]`	2021 年 9 月 30 日	Braket
Amazon Chime	`arn:aws:states:::aws-sdk:chime: [apiAction]`	2021 年 9 月 30 日	Chime
Amazon Chime Meetings	`arn:aws:states:::aws-sdk:chimesdkmeetings: [apiAction]`	2022 年 4 月 19 日	ChimeSdkMeetings
Amazon Cloud Directory	`arn:aws:states:::aws-sdk:clouddirectory: [apiAction]`	2021 年 9 月 30 日	CloudDirectory
Amazon CloudFront	`arn:aws:states:::aws-sdk:cloudfront: [apiAction]`	2021 年 9 月 30 日	CloudFront

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon CloudSearch	`arn:aws:states:::aws-sdk:cloudsearch: [apiAction]`	2021 年 9 月 30 日	CloudSearch
Amazon CloudWatch	`arn:aws:states:::aws-sdk:cloudwatch: [apiAction]`	2021 年 9 月 30 日	CloudWatch
Amazon CloudWatch Application Insights	`arn:aws:states:::aws-sdk:applicationinsights: [apiAction]`	2021 年 9 月 30 日	ApplicationInsights
CloudWatch Evidently	`arn:aws:states:::aws-sdk:evidently: [apiAction]`	2022 年 4 月 19 日	Evidently
Amazon CloudWatch Logs	`arn:aws:states:::aws-sdk:cloudwatchlogs: [apiAction]`	2021 年 9 月 30 日	CloudWatchLogs
Amazon CloudWatch RUM	`arn:aws:states:::aws-sdk:rum: [apiAction]`	2022 年 4 月 19 日	Rum
Amazon CloudWatch Synthetics	`arn:aws:states:::aws-sdk:synthetics: [apiAction]`	2021 年 9 月 30 日	Synthetics
Amazon CodeGuru Profiler	`arn:aws:states:::aws-sdk:codeguruprofiler: [apiAction]`	2021 年 9 月 30 日	CodeGuruProfiler

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon CodeGuru Reviewer	`arn:aws:states:::aws-sdk:codegurureviewer: [apiAction]`	2021 年 9 月 30 日	CodeGuruReviewer
Amazon Cognito	`arn:aws:states:::aws-sdk:cognitoidentity: [apiAction]`	2021 年 9 月 30 日	CognitoIdentity
Amazon Cognito Identity Provider	`arn:aws:states:::aws-sdk:cognitoidentityprovider: [apiAction]`	2021 年 9 月 30 日	CognitoIdentityProvider
Amazon Cognito Sync	`arn:aws:states:::aws-sdk:cognitosync: [apiAction]`	2021 年 9 月 30 日	CognitoSync
Amazon Comprehend	`arn:aws:states:::aws-sdk:comprehend: [apiAction]`	2021 年 9 月 30 日	Comprehend
Amazon Comprehend Medical	`arn:aws:states:::aws-sdk:comprehendmedical: [apiAction]` *** —	2021 年 9 月 30 日	ComprehendMedical
Amazon Connect Contact Lens	`arn:aws:states:::aws-sdk:connectcontactlens: [apiAction]`	2021 年 9 月 30 日	ConnectContactLens
Amazon Connect Participant Service	`arn:aws:states:::aws-sdk:connectparticipant: [apiAction]`	2021 年 9 月 30 日	ConnectParticipant

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Connect	`arn:aws:states:::aws-sdk:connect: [apiAction]`	2021 年 9 月 30 日	Connect
Amazon Connect Voice ID	`arn:aws:states:::aws-sdk:voiceid: [apiAction]`	2022 年 4 月 19 日	VoiceId
Amazon Connect Wisdom	`arn:aws:states:::aws-sdk:wisdom: [apiAction]`	2022 年 4 月 19 日	Wisdom
Amazon Data Lifecycle Manager	`arn:aws:states:::aws-sdk:dlm: [apiAction]`	2021 年 9 月 30 日	Dlm
Amazon Detective	`arn:aws:states:::aws-sdk:detective: [apiAction]`	2021 年 9 月 30 日	Detective
Amazon DevOps Guru	`arn:aws:states:::aws-sdk:devopsguru: [apiAction]`	2021 年 9 月 30 日	DevOpsGuru
Amazon DocumentDB (with MongoDB compatibility)	`arn:aws:states:::aws-sdk:docdb: [apiAction]`	2021 年 9 月 30 日	DocDb
Amazon DynamoDB	`arn:aws:states:::aws-sdk:dynamodb: [apiAction]`	2021 年 9 月 30 日	DynamoDb

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon DynamoDB Streams	`arn:aws:states:::aws-sdk:dynamodbstreams: [apiAction]`	2021 年 9 月 30 日	DynamoDbStreams
Amazon EC2 Container Registry	`arn:aws:states:::aws-sdk:ecr: [apiAction]`	2021 年 9 月 30 日	Ecr
Amazon EC2 Container Service	`arn:aws:states:::aws-sdk:ecs: [apiAction]`	2021 年 9 月 30 日	Ecs
Amazon EC2 Systems Manager	`arn:aws:states:::aws-sdk:ssm: [apiAction]`	2021 年 9 月 30 日	Ssm
Amazon EMR	`arn:aws:states:::aws-sdk:emrcontainers: [apiAction]` *** —	2021 年 9 月 30 日	EmrContainers
Amazon ElastiCache	`arn:aws:states:::aws-sdk:elasticache: [apiAction]`	2021 年 9 月 30 日	ElastiCache
Amazon Elastic Inference	`arn:aws:states:::aws-sdk:elasticinference: [apiAction]`	2021 年 9 月 30 日	ElasticInference
Amazon Elastic Block Store	`arn:aws:states:::aws-sdk:ebs: [apiAction]`	2021 年 9 月 30 日	Ebs
Amazon Elastic Compute Cloud	`arn:aws:states:::aws-sdk:ec2: [apiAction]`	2021 年 9 月 30 日	Ec2

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Elastic Container Registry Public	`arn:aws:states:::aws-sdk:ecrpublic: [apiAction]`	2021 年 9 月 30 日	EcrPublic
Amazon Elastic File System	`arn:aws:states:::aws-sdk:efs: [apiAction]` ***	2021 年 9 月 30 日	Efs
Amazon Elastic Kubernetes Service	`arn:aws:states:::aws-sdk:eks: [apiAction]`	2021 年 9 月 30 日	Eks
Amazon EMR	`arn:aws:states:::aws-sdk:emr: [apiAction]`	2021 年 9 月 30 日	Emr
Amazon Elastic Transcoder	`arn:aws:states:::aws-sdk:elastictranscoder: [apiAction]` ***	2021 年 9 月 30 日	ElasticTranscoder
Amazon OpenSearch Service	`arn:aws:states:::aws-sdk:elasticsearch: [apiAction]`	2021 年 9 月 30 日	Elasticsearch
Amazon OpenSearch Service	`arn:aws:states:::aws-sdk:opensearch: [apiAction]`	2022 年 4 月 19 日	OpenSearch
Amazon EventBridge	`arn:aws:states:::aws-sdk:eventbridge: [apiAction]`	2021 年 9 月 30 日	EventBridge
Amazon FSx	`arn:aws:states:::aws-sdk:fsx: [apiAction]`	2021 年 9 月 30 日	FSx

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Forecast Query	`arn:aws:states:::aws-sdk:forecastquery: [apiAction]`	2021 年 9 月 30 日	Forecastquery
Amazon Forecast Service	`arn:aws:states:::aws-sdk:forecast: [apiAction]`	2021 年 9 月 30 日	Forecast
Amazon Fraud Detector	`arn:aws:states:::aws-sdk:frauddetector: [apiAction]`	2021 年 9 月 30 日	FraudDetector
Amazon GameLift	`arn:aws:states:::aws-sdk:gamelift: [apiAction]`	2021 年 9 月 30 日	Amazon GameLift
Amazon GameSparks	`arn:aws:states:::aws-sdk:gamesparks: [apiAction]`	2022 年 7 月 27 日	GameSparks
Amazon S3 Glacier	`arn:aws:states:::aws-sdk:glacier: [apiAction]`	2021 年 9 月 30 日	Glacier
Amazon GuardDuty	`arn:aws:states:::aws-sdk:guardduty: [apiAction]`	2021 年 9 月 30 日	GuardDuty
AWS HealthLake	`arn:aws:states:::aws-sdk:healthlake: [apiAction]`	2021 年 9 月 30 日	HealthLake
Amazon Honeycode	`arn:aws:states:::aws-sdk:honeycode: [apiAction]`	2021 年 9 月 30 日	Honeycode

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Inspector	`arn:aws:states:::aws-sdk:inspector: [apiAction]`	2021 年 9 月 30 日	Inspector
Amazon Inspector V2	`arn:aws:states:::aws-sdk:inspector2: [apiAction]`	2022 年 4 月 19 日	Inspector2
Amazon Interactive Video Service	`arn:aws:states:::aws-sdk:ivs: [apiAction]`	2021 年 9 月 30 日	Ivs
Amazon Kendra	`arn:aws:states:::aws-sdk:kendra: [apiAction]`	2021 年 9 月 30 日	Kendra
Amazon Kinesis	`arn:aws:states:::aws-sdk:kinesis: [apiAction]` ***	2021 年 9 月 30 日	Kinesis
Amazon Kinesis Analytics	`arn:aws:states:::aws-sdk:kinesisanalytics: [apiAction]`	2021 年 9 月 30 日	KinesisAnalytics
Amazon Kinesis Analytics V2	`arn:aws:states:::aws-sdk:kinesisanalyticsv2: [apiAction]`	2021 年 9 月 30 日	KinesisAnalyticsV2
Amazon Kinesis Firehose	`arn:aws:states:::aws-sdk:firehose: [apiAction]`	2021 年 9 月 30 日	Firehose

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Kinesis Video Signaling Channels	`arn:aws:states:::aws-sdk:kinesisvideosignaling:` *[apiAction]*	2021 年 9 月 30 日	KinesisVideoSignaling
Amazon Kinesis Video Streams	`arn:aws:states:::aws-sdk:kinesisvideo:` *[apiAction]*	2021 年 9 月 30 日	KinesisVideo
Amazon Kinesis Video Streams Archived Media	`arn:aws:states:::aws-sdk:kinesisvideoarchivedmedia:` *[apiAction]*	2021 年 9 月 30 日	KinesisVideoArchivedMedia
Amazon Kinesis video stream	`arn:aws:states:::aws-sdk:kinesisvideomedia:` *[apiAction]*	2021 年 9 月 30 日	KinesisVideoMedia
Amazon Lex Model Building Service	`arn:aws:states:::aws-sdk:lexmodelbuilding:` *[apiAction]*	2021 年 9 月 30 日	LexModelBuilding
Amazon Lex Model Building Service V2	`arn:aws:states:::aws-sdk:lexmodelsv2:` *[apiAction]*	2021 年 9 月 30 日	LexModelsV2
Amazon Lex	`arn:aws:states:::aws-sdk:lexruntime:` *[apiAction]*	2021 年 9 月 30 日	LexRuntime
Amazon Lex Runtime V2	`arn:aws:states:::aws-sdk:lexruntimev2:` *[apiAction]* *** —	2021 年 9 月 30 日	LexRuntimeV2

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Lightsail	arn:aws:states:::aws-sdk:lightsail: *[apiAction]*	2021 年 9 月 30 日	Lightsail
Amazon Location Service	arn:aws:states:::aws-sdk:location: *[apiAction]*	2021 年 9 月 30 日	Location
Amazon Lookout for Equipment	arn:aws::states:::aws-sdk:lookoutequipment: *[apiAction]*	2021 年 9 月 30 日	LookoutEquipment
Amazon Lookout for Metrics	arn:aws:states:::aws-sdk:lookoutmetrics: *[apiAction]*	2021 年 9 月 30 日	LookoutMetrics
Amazon Lookout for Vision	arn:aws:states:::aws-sdk:lookoutvision: *[apiAction]*	2021 年 9 月 30 日	LookoutVision
Amazon MQ	arn:aws:states:::aws-sdk:mq: *[apiAction]*	2021 年 9 月 30 日	Mq
Amazon Macie	arn:aws:states:::aws-sdk:macie: *[apiAction]*	2021 年 9 月 30 日	
Amazon Macie 2	arn:aws:states:::aws-sdk:macie2: *[apiAction]*	2021 年 9 月 30 日	Macie2
Amazon Managed Blockchain	arn:aws:states:::aws-sdk:managedblockchain: *[apiAction]*	2021 年 9 月 30 日	ManagedBlockchain

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Managed Grafana	`arn:aws:states:::aws-sdk:grafana: [apiAction]`	2022 年 4 月 19 日	Grafana
Amazon Managed Service for Prometheus	`arn:aws:states:::aws-sdk:amp: [apiAction]`	2021 年 9 月 30 日	Amp
Amazon Managed Streaming for Apache Kafka	`arn:aws:states:::aws-sdk:kafka: [apiAction]`	2021 年 9 月 30 日	Kafka
Amazon MSK Connect	`arn:aws:states:::aws-sdk:kafkaconnect: [apiAction]`	2022 年 4 月 19 日	KafkaConnect
Amazon Managed Workflows for Apache Airflow	`arn:aws:states:::aws-sdk:mwaa: [apiAction]`	2021 年 9 月 30 日	Mwaa
Amazon Mechanical Turk	`arn:aws:states:::aws-sdk:mturk: [apiAction]`	2021 年 9 月 30 日	MTurk
Amazon MemoryDB for Redis	`arn:aws:states:::aws-sdk:memorydb: [apiAction]`	2022 年 4 月 19 日	MemoryDB
Amazon Nimble Studio	`arn:aws:states:::aws-sdk:nimble: [apiAction]`	2021 年 9 月 30 日	Nimble
Amazon Personalize	`arn:aws:states:::aws-sdk:personalize: [apiAction]`	2021 年 9 月 30 日	Personalize

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Personalize Events	`arn:aws:states:::aws-sdk:personalizeevents: [apiAction]`	2021 年 9 月 30 日	PersonalizeEvents
Amazon Personalize Runtime	`arn:aws:states:::aws-sdk:personalizeruntime: [apiAction]`	2021 年 9 月 30 日	PersonalizeRuntime
Amazon Pinpoint	`arn:aws:states:::aws-sdk:pinpoint: [apiAction]`	2021 年 9 月 30 日	Pinpoint
Amazon Pinpoint Email Service	`arn:aws:states:::aws-sdk:pinpointemail: [apiAction]`	2021 年 9 月 30 日	PinpointEmail
Amazon Pinpoint SMS and Voice Service	`arn:aws:states:::aws-sdk:pinpointsmsvoice: [apiAction]`	2021 年 9 月 30 日	PinpointSmsVoice
Amazon Pinpoint SMS and Voice V2 Service	`arn:aws:states:::aws-sdk:pinpointsmsvoicev2: [apiAction]`	2022 年 7 月 27 日	PinpointSmsVoiceV2
Amazon Polly	`arn:aws:states:::aws-sdk:polly: [apiAction]`	2021 年 9 月 30 日	Polly
Amazon QLDB	`arn:aws:states:::aws-sdk:qldb: [apiAction]`	2021 年 9 月 30 日	Qldb
Amazon QLDB Session	`arn:aws:states:::aws-sdk:qldbsession: [apiAction]`	2021 年 9 月 30 日	QldbSession

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon QuickSight	`arn:aws:states:::aws-sdk:quicksight:` *[apiAction]*	2021 年 9 月 30 日	QuickSight
Amazon Redshift	`arn:aws:states:::aws-sdk:redshift:` *[apiAction]*	2021 年 9 月 30 日	Redshift
Amazon Redshift Data API	`arn:aws:states:::aws-sdk:redshiftdata:` *[apiAction]*	2021 年 9 月 30 日	RedshiftData
Amazon Rekognition	`arn:aws:states:::aws-sdk:rekognition:` *[apiAction]*	2021 年 9 月 30 日	Rekognition
Amazon Relational Database Service	`arn:aws:states:::aws-sdk:rds:` *[apiAction]*	2021 年 9 月 30 日	Rds
Amazon Route 53	`arn:aws:states:::aws-sdk:route53:` *[apiAction]*	2021 年 9 月 30 日	Route53
Amazon Route 53 Recovery Control Config	`arn:aws:states:::aws-sdk:route53recoverycontrolconfig:` *[apiAction]*	2021 年 9 月 30 日	Route53RecoveryControlConfig
Amazon Route 53 Domains	`arn:aws:states:::aws-sdk:route53domains:` *[apiAction]*	2021 年 9 月 30 日	Route53Domains

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Route 53 Resolver	arn:aws:states:::aws-sdk:route53resolver: *[apiAction]*	2021 年 9 月 30 日	Route53Resolver
Amazon S3 on Outposts	arn:aws:states:::aws-sdk:s3outposts: *[apiAction]*	2021 年 9 月 30 日	S3Outposts
Amazon SageMaker Runtime Feature Store Runtime	arn:aws:states:::aws-sdk:sagemakerfeaturestoreruntime: *[apiAction]*	2021 年 9 月 30 日	SageMaker RuntimeFeatureStoreRuntime
Amazon SageMaker Runtime Runtime	arn:aws:states:::aws-sdk:sagemakerruntime: *[apiAction]*	2021 年 9 月 30 日	SageMaker RuntimeRuntime
Amazon SageMaker	arn:aws:states:::aws-sdk:sagemaker: *[apiAction]*	2021 年 9 月 30 日	SageMaker Runtime
Amazon SageMaker Edge Manager	arn:aws:states:::aws-sdk:sagemakeredge: *[apiAction]*	2021 年 9 月 30 日	SagemakerEdge
Amazon Simple Email Service	arn:aws:states:::aws-sdk:ses: *[apiAction]*	2021 年 9 月 30 日	Ses
Amazon Simple Email Service V2	arn:aws:states:::aws-sdk:sesv2: *[apiAction]*	2021 年 9 月 30 日	SesV2

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon Simple Notification Service	`arn:aws:states:::aws-sdk:sns:` *`[apiAction]`*	2021 年 9 月 30 日	Sns
Amazon Simple Queue Service	`arn:aws:states:::aws-sdk:sqs:` *`[apiAction]`*	2021 年 9 月 30 日	Sqs
Amazon Simple Storage Service	`arn:aws:states:::aws-sdk:s3:` *`[apiAction]`* ***	2021 年 9 月 30 日	S3
Amazon Simple Workflow Service	`arn:aws:states:::aws-sdk:swf:` *`[apiAction]`*	2021 年 9 月 30 日	Swf
Amazon Textract	`arn:aws:states:::aws-sdk:textract:` *`[apiAction]`*	2021 年 9 月 30 日	Textract
Amazon Transcribe	`arn:aws:states:::aws-sdk:transcribe:` *`[apiAction]`*	2021 年 9 月 30 日	Transcribe
Amazon Translate	`arn:aws:states:::aws-sdk:translate:` *`[apiAction]`*	2021 年 9 月 30 日	Translate
Amazon WorkDocs	`arn:aws:states:::aws-sdk:workdocs:` *`[apiAction]`*	2021 年 9 月 30 日	WorkDocs
Amazon WorkMail	`arn:aws:states:::aws-sdk:workmail:` *`[apiAction]`*	2021 年 9 月 30 日	WorkMail

サービス名	Task ステートリソース	サポート日	例外プレフィックス
Amazon WorkMail Message Flow	`arn:aws:states:::aws-sdk:workmailmessageflow:` *`[apiAction]`*	2021 年 9 月 30 日	WorkMailMessageFlow
Amazon WorkSpaces	`arn:aws:states:::aws-sdk:workspaces:` *`[apiAction]`*	2021 年 9 月 30 日	WorkSpaces
Amazon WorkSpaces Web	`arn:aws:states:::aws-sdk:workspacesweb:` *`[apiAction]`*	2022 年 4 月 19 日	WorkSpacesWeb
Amplify	`arn:aws:states:::aws-sdk:amplifybackend:` *`[apiAction]`*	2021 年 9 月 30 日	AmplifyBackend
Amplify UI Builder	`arn:aws:states:::aws-sdk:amplifyuibuilder:` *`[apiAction]`*	2022 年 4 月 19 日	AmplifyUiBuilder
Application Auto Scaling	`arn:aws:states:::aws-sdk:applicationautoscaling:` *`[apiAction]`*	2021 年 9 月 30 日	ApplicationAutoScaling
Amazon EC2 Auto Scaling	`arn:aws:states:::aws-sdk:autoscaling:` *`[apiAction]`*	2021 年 9 月 30 日	Auto Scaling
CodeArtifact	`arn:aws:states:::aws-sdk:codeartifact:` *`[apiAction]`*	2021 年 9 月 30 日	Codeartifact

サービス名	Task ステートリソース	サポート日	例外プレフィックス
DynamoDB Accelerator	`arn:aws:states:::aws-sdk:dax:` *`[apiAction]`*	2021 年 9 月 30 日	Dax
EC2 Image Builder	`arn:aws:states:::aws-sdk:imagebuilder:` *`[apiAction]`*	2021 年 9 月 30 日	Imagebuilder
AWS Elastic Disaster Recovery	`arn:aws:states:::aws-sdk:drs:` *`[apiAction]`*	2022 年 4 月 19 日	Drs
Elastic Load Balancing	`arn:aws:states:::aws-sdk:elasticloadbalancing:` *`[apiAction]`*	2021 年 9 月 30 日	ElasticLoadBalancing
Elastic Load Balancing V2	`arn:aws:states:::aws-sdk:elasticloadbalancingv2:` *`[apiAction]`*	2021 年 9 月 30 日	ElasticLoadBalancingV2
MediaConnect	`arn:aws:states:::aws-sdk:mediaconnect:` *`[apiAction]`*	2021 年 9 月 30 日	MediaConnect
Amazon S3 Control	`arn:aws:states:::aws-sdk:s3control:` *`[apiAction]`* ***	2021 年 9 月 30 日	S3Control
Recycle Bin for Amazon EBS	`arn:aws:states:::aws-sdk:rbin:` *`[apiAction]`*	2022 年 4 月 19 日	Rbin

サービス名	**Task** ステートリソース	サポート日	例外プレフィックス
Savings Plans	`arn:aws:states:::aws-sdk:savingsplans: [apiAction]`	2021 年 9 月 30 日	Savingsplans
Amazon EventBridge Schema Registry	`arn:aws:states:::aws-sdk:schemas: [apiAction]`	2021 年 9 月 30 日	Schemas
Service Quotas	`arn:aws:states:::aws-sdk:servicequotas: [apiAction]`	2021 年 9 月 30 日	ServiceQuotas
AWS Snowball	`arn:aws:states:::aws-sdk:snowball: [apiAction]`	2021 年 9 月 30 日	Snowball

サポートされているサービスでサポートされていない API アクション

次の表は、 AWS SDK サービスインテグレーションでサポートされていない API アクションの一覧です。右側の列には、左側の列にリストされているサービスで現在サポートされていない API アクションが記載されています。

サービス名	サポートされていない API アクション
AWS Application Discovery Service	• `DescribeExportConfigurations` • `ExportConfigurations`
Amazon Bedrock	• `InvokeModelWithResponseStream`
Amazon Bedrock ランタイム用エージェント	• `InvokeAgent`
AWS CodeDeploy	• `BatchGetDeploymentInstances` • `GetDeploymentInstance` • `ListDeploymentInstances`

サービス名	サポートされていない API アクション
	• SkipWaitTimeForInstanceTermination
Amazon Comprehend Medical	• DetectEntities
AWS Direct Connect	• AllocateConnectionOnInterconnect • DescribeConnectionLoa • DescribeConnectionsOnInterconnect • DescribeInterconnectLoa
Amazon Elastic File System	• CreateTags
Amazon Elastic Transcoder	• TestRole
Amazon EMR	• DescribeJobFlows
AWS IoT	• AttachPrincipalPolicy • ListPrincipalPolicies • DetachPrincipalPolicy • ListPolicyPrincipals • DetachPrincipalPolicy
AWS IoT コアデバイスアドバイザー	• ListTestCases
Amazon Kinesis	• SubscribeToShard
AWS Lambda	• InvokeAsync • InvokeWithResponseStream
Amazon Lex Runtime V2	• StartConversation
AWS Elemental MediaPackage	• RotateChannelCredentials
Amazon Relational Database Service	• ExecuteSql

サービス名	サポートされていない API アクション
Amazon Simple Storage Service	• `SelectObjectContent`
Amazon S3 コントロール	• `SelectObjectContent`
AWS Shield	• `DeleteSubscription`
AWS Security Token Service	• `AssumeRole` • `AssumeRoleWithSAML` • `AssumeRoleWithWebIdentity`

非推奨の AWS SDK サービス統合

次の AWS SDK サービス統合は廃止されました。

- AWS モバイル
- Amazon Macie

Step Functions 用統合最適化

以下のトピックには、他のサービスを調整するための Amazon States Language でサポートされている API、パラメータ、リクエスト/レスポンス構文が含まれています。 AWS トピックでは、コード例も示します。`Task`状態の `Resource` フィールドで、最適化された統合サービスを Amazon States Language から直接呼び出すことができます。

3 つのサービス統合パターンを使用できます。

- レスポンスをリクエスト (デフォルト)-HTTP レスポンスを待ってから、次の状態に進みます。
- Job ブの実行 (`.sync`)-ジョブが完了するのを待つ
- Callback (`.waitForTaskToken`) を待つ-タスクトークンが返されるまでワークフローを一時停止します

標準ワークフローとエクスプレスワークフローは同じ統合をサポートしていますが、同じ統合パターンはサポートしていません。

- 最適化された統合パターンのサポートは、各統合ごとに異なります。
- エクスプレスワークフローは、 Job 実行 (.sync) またはコールバックを待機 (. waitForTaskトークン)。
- 詳細については、「標準ワークフロー対 Express ワークフロー」を参照してください。

Standard Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS SDK インテグレーション	200 以上	✓		✓

Express Workflows

サポートされているサービス統合

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
最適化された統合	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	Amazon SageMaker	✓		
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		

	サービス	レスポンスのリクエスト	ジョブの実行 (.sync)	コールバックまで待機 (.waitForTaskToken)
AWS SDK インテグレーション	200 以上	✓		

Step Functions を使用して API Gateway を呼び出し

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> 最適化された API ゲートウェイ統合と API Gateway AWS SDK 統合との違い
>
> - `apigateway:invoke`: AWS SDK サービス統合には同等の機能はありません。代わりに、最適化 API Gateway サービスは API Gateway エンドポイントを直接呼び出します。

Amazon API Gateway を使用して、HTTP と REST API の作成、発行、管理、モニタリングが可能です。API Gateway と統合するには、コードを記述したり、他のインフラストラクチャに依存したりすることなく、API Gateway HTTP または API Gateway REST エンドポイントを直接呼び出す Step Functions 内の `Task` 状態を定義します。`Task` 状態定義には、API コールに必要なすべての情報が含まれます。別の認可方法を選択することもできます。

> Note
>
> Step Functions は、API Gateway を介して HTTP エンドポイントを呼び出す機能をサポートしていますが、現在、汎用 HTTP エンドポイントを呼び出す機能はサポートしていません。

API Gateway 特徴のサポート

Step Functions API Gateway 統合は、一部の API Gateway 機能をサポートしますが、すべてはサポートしていません。サポートされている特徴の詳細なリストについては、以下を参照してください。

- 以下は、Step Functions API Gateway REST API と API Gateway HTTP API 統合の両方でサポートされています。
  - オーソライザー: IAM (署名バージョン 4を使用)、認証なし、Lambda Authorizers (カスタムヘッダーを使用したリクエストパラメータベースおよびトークンベース)
  - API タイプ: リージョン
  - API 管理: API Gateway API ドメイン名、API ステージ、パス、クエリパラメータ、リクエストボディ
- Step Functions API Gateway HTTP API 統合でサポートされています。エッジ最適化 API のオプションを提供する Step Functions API Gateway REST API 統合はサポートされていません。
- Step Functions API Gateway 統合ではサポートされていません。
  - オーソライザー: Amazon Cognito、ネイティブオープン ID Connect/OAuth 2.0、トークンベースの Lambda オーソライザーの認可ヘッダー
  - API タイプ: プライベート
  - API 管理: カスタムドメイン名

API Gateway と HTTP および REST API の詳細については、以下を参照してください。

- Amazon API Gateway の概念ページ。
- API Gateway デベロッパーガイドの HTTP API と REST API 間で選択します。

リクエストの形式

`Task` 状態定義を作成するとき、Step Functions はパラメータを検証し、呼び出しを実行するのに必要な URL を構築し、API を呼び出します。レスポンスには、HTTP ステータスコード、ヘッダー、およびレスポンス本文が含まれます。リクエスト形式には、必須およびオプションのパラメータがあります。

必須リクエストパラメータ

- `ApiEndpoint`

- タイプ: String
- API Gateway URL のホスト名。形式は *<API ID>*.execute-api.*<region>*.amazonaws.com です。

  API ID には、次の英数字の組み合わせのみを使用できます: 0123456789abcdefghijklmnopqrstuvwxyz

- Method
  - タイプ: Enum
  - HTTP メソッド。次のいずれかとなる必要があります。
    - GET
    - POST
    - PUT
    - DELETE
    - PATCH
    - HEAD
    - OPTIONS

オプションのリクエストパラメータ

- Headers
  - タイプ: JSON
  - HTTP ヘッダーは、同じキーに関連付けられた値のリストを許可します。
- Stage
  - タイプ: String
  - API Gateway で API がデプロイされるステージの名前。$default ステージを使用する HTTP API のオプションとなっています。
- Path
  - タイプ: String
  - API エンドポイントの後に追加されるパスパラメータ。
- QueryParameters
  - タイプ: JSON
  - クエリ文字列では、同じキーに関連付けられた値のリストのみが許可されます。

- RequestBody
  - タイプ: JSON または String
  - HTTP リクエストボディ。そのタイプは、JSON オブジェクトまたは String です。RequestBody は、PATCH、POST、および PUT HTTP メソッドに対してのみサポートされています。
- AllowNullValues
  - タイプ: BOOLEAN
  - AllowNullValues を true に設定すると、次のように、ヌル値を渡す許可がでます。

```
{
    "NewPet": {
        "type": "turtle",
        "price": 123,
        "name": null
    }
}
```

- AuthType
  - タイプ: JSON
  - 認証方法。デフォルトの方法は NO_AUTH です。指定できる値は次のとおりです。
    - NO_AUTH
    - IAM_ROLE
    - RESOURCE_POLICY

    詳細は、認証と認可を参照してください。

> **Note**
>
> セキュリティを考慮して、以下の HTTP ヘッダーキーは現在、許可されていません。
>
> - X-Forwarded、X-Amz、または X-Amzn のプレフィックスが付いているキー。
> - Authorization
> - Connection
> - Content-md5
> - Expect

- Host
- Max-Forwards
- Proxy-Authenticate
- Server
- TE
- Transfer-Encoding
- Trailer
- Upgrade
- Via
- Www-Authenticate

次のコード例は、Step Functions を使用して API Gateway を呼び出す方法を示しています。

```
{
    "Type": "Task",
    "Resource":"arn:aws:states:::apigateway:invoke",
    "Parameters": {
        "ApiEndpoint": "example.execute-api.us-east-1.amazonaws.com",
        "Method": "GET",
        "Headers": {
            "key": ["value1", "value2"]
        },
        "Stage": "prod",
        "Path": "bills",
        "QueryParameters": {
            "billId": ["123456"]
        },
        "RequestBody": {},
        "AuthType": "NO_AUTH"
    }
}
```

認証と認可

次の認証方法を使用できます。

- 認可なし: 認可メソッドなしで API を直接呼び出します。
- IAM ロール: この方法では、Step Functions はステートマシンのロールを引き受けて、署名バージョン 4 (SigV4) を使ってリクエストに署名した後、API を呼び出します。
- リソースポリシー: Step Functions はリクエストを認証し、API を呼び出します。API に以下を指定するリソースポリシーをアタッチする必要があります。

  1. API Gateway を呼び出すステートマシン。

     > **Important**
     >
     > アクセスを制限するため、ステートマシンを指定する必要があります。そうしない場合は、API へのリソースポリシー認証を使って API Gateway リクエストを認証するステートマシンにアクセスが付与されます。

  2. そのStep Functions は、API Gateway を呼び出すサービスです: `"Service": "states.amazonaws.com"`。
  3. アクセス対象のリソースには、以下が含まれます。
     - *region*。
     - 指定されたリージョンの *account-id*。
     - *api-id*。
     - *stage-name*。
     - *HTTP-VERB* (メソッド)。
     - は、*resource-path-specifier*.

  リソースポリシーの例については、Step Functions と API Gateway 用 IAM ポリシーを参照してください。

  リソース形式の詳細については、API Gateway デベロッパーガイドの API Gateway における API 実行許可のリソース形式を参照してください。

  > **Note**
  >
  > リソースポリシーは REST API の場合に限り、サポートされます。

サービス統合パターン

API Gateway 統合では、次の 2 つのサービス統合パターンがサポートされています。

- レスポンスのリクエスト。デフォルトの統合パターンです。このおかげで、TTP レスポンスを受信した直後に Step Functions が H次のステップに進むことができます。
- タスクトークンのコールバックまで待機する (.waitForTaskToken) は、タスクトークンがペイロードとともに返されるまで待機します。.waitForTaskTokenこのパターンを使用するには、を追加します。 waitForTask次の例のように、タスク定義の Resource フィールドの末尾にあるトークン。

```
{
    "Type": "Task",
    "Resource":"arn:aws:states:::apigateway:invoke.waitForTaskToken",
    "Parameters": {
        "ApiEndpoint": "example.execute-api.us-east-1.amazonaws.com",
        "Method": "POST",
        "Headers": {
            "TaskToken.$": "States.Array($$.Task.Token)"
        },
        "Stage": "prod",
        "Path": "bills/add",
        "QueryParameters": {},
        "RequestBody": {
            "billId": "my-new-bill"
        },
        "AuthType": "IAM_ROLE"
    }
}
```

出力形式

次の出力パラメータが提供されます。

名前	型	説明
ResponseBody	JSON または String	API コールのレスポンスの本文。

名前	型	説明
Headers	JSON	レスポンスヘッダー
StatusCode	Integer	レスポンスの HTTP ステータスコード。
StatusText	String	レスポンスのステータステキスト。

レスポンスの例:

```
{
    "ResponseBody": {
        "myBills": []
    },
    "Headers": {
        "key": ["value1", "value2"]
    },
    "StatusCode": 200,
    "StatusText": "OK"
}
```

エラー処理

エラーが発生した場合、error そして cause は次のように返されます。

- HTTP ステータスコードが使用可能な場合、エラーは ApiGateway.*<HTTP Status Code>* 形式で返されます。
- HTTP ステータスコードが使用できない場合、エラーは ApiGateway.*<Exception>* 形式で返されます。

いずれの場合も、cause は文字列として返されます。

以下は、エラーが発生したレスポンスの例です。

```
{
    "error": "ApiGateway.403",
    "cause": "{\"message\":\"Missing Authentication Token\"}"
```

```
}
```

> **Note**
>
> 2XX のステータスコードは成功を示し、エラーは返されません。他のすべてのステータスコードまたはスローされた例外は、エラーになります。

詳細については、以下を参照してください。

- API Gateway デベロッパーガイドの Amazon API Gateway の概念。
- Amazon API Gateway IAM ポリシー
- API Gateway を呼び出す の方法を示すサンプルプロジェクト

API Gateway デベロッパーガイドの Amazon API Gateway の概念。

Step Functions で Athena を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> **最適化された Athena インテグレーションと AWS Athena SDK インテグレーションの違い**
>
> - ジョブの実行 (.sync) 統合パターンがサポートされています。
> - レスポンスのリクエスト 統合パターンの最適化はありません。
> - タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。

Amazon Athena AWS Step Functions とのサービス統合により、Step Functions を使用してクエリの実行を開始および停止し、クエリ結果を取得できます。Step Functions を使用して、アドホックまたはスケジュールされたデータクエリを実行し、S3 データレイクを対象とした結果を取得できます。Athena はサーバーレスであるため、インフラストラクチャの設定や管理は不要です。また、実行したクエリにのみ課金されます。

Amazon Athena AWS Step Functions と統合するには、提供されている Athena サービス統合 API を使用します。

サービス統合 API は、対応する Athena API と同じです。次の表に示すとおり、すべての API がすべての統合パターンをサポートしているわけではありません。

API	レスポンスのリクエスト	ジョブの実行 (.sync)
StartQueryExecution	✓	✓
StopQueryExecution	✓	
GetQueryExecution	✓	
GetQueryResults	✓	

サポートされている Amazon Athena API:

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

- StartQueryExecution
  - リクエストの構文
  - サポートされているパラメータ:
    - ClientRequestToken
    - ExecutionParameters
    - QueryExecutionContext
    - QueryString
    - ResultConfiguration
    - WorkGroup
  - Response syntax
- StopQueryExecution
  - リクエストの構文

    - サポートされているパラメータ:
        - QueryExecutionId
- GetQueryExecution
    - リクエストの構文
    - サポートされているパラメータ:
        - QueryExecutionId
    - Response syntax
- GetQueryResults
    - リクエストの構文
    - サポートされているパラメータ:
        - MaxResults
        - NextToken
        - QueryExecutionId
    - Response syntax

以下には、Athena クエリをスタートするタスク状態が含まれます。

```
"Start an Athena query": {
  "Type": "Task",
  "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
  "Parameters": {
    "QueryString": "SELECT * FROM \"myDatabase\".\"myTable\" limit 1",
    "WorkGroup": "primary",
    "ResultConfiguration": {
       "OutputLocation": "s3://athenaQueryResult"
    }
  },
  "Next": "Get results of the query"
}
```

IAMStep Functions AWS 他のサービスと併用する場合のアクセス権限の設定方法については、「」を参照してください。統合サービスの IAM ポリシー

AWS Batch Step Functions による管理

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> AWS Batch 最適化インテグレーションは AWS BatchAWS SDK インテグレーションとどう違うのか
>
> - ジョブの実行 (.sync) 統合パターンが利用可能です。
>
> レスポンスのリクエスト または タスクトークンのコールバックまで待機する 統合パターンの最適化はないことに注意してください。

サポートされている AWS Batch API:

- SubmitJob
  - リクエストの構文
  - サポートされているパラメータ:
    - ArrayProperties
    - ContainerOverrides
    - DependsOn
    - JobDefinition
    - JobName
    - JobQueue
    - Parameters
    - RetryStrategy
    - Timeout
    - Tags
  - レスポンスの構文

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

以下には、ジョブをサブミットし、Task AWS Batch ジョブが完了するのを待つ状態が含まれます。

```
{
  "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": {
          "ResourceRequirements": [
            {
              "Type": "VCPU",
              "Value": "4"
            }
          ]
        }
      },
      "End": true
    }
  }
}
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください。[統合サービスの IAM ポリシー]

Step Functions を使用して Amazon Bedrock を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

トピック

- Amazon Bedrock サービス統合 API
- Amazon Bedrock 統合のタスクステート定義

Amazon Bedrock サービス統合 API

AWS Step Functions を Amazon Bedrock と統合するために、以下の API を使用できます。これらの API は、対応する Amazon Bedrock API に似ていますが、渡されるリクエストにいくつかの違いがあります。

各サービス統合 API と対応する Amazon Bedrock API の違いを次の表に示します。

Amazon Bedrock サービス統合 API と、対応する Amazon Bedrock API

Amazon Bedrock サービス統合 API	対応する Amazon Bedrock API	差異
InvokeModel リクエスト本文に指定された入力を使って推論を実行するために、指定された Amazon Bedrock モデルを呼び出します。テキストモデル、画像モデル、埋め込みモデルの推論を実行するために、`InvokeModel` を使用します。	InvokeModel	Amazon Bedrock サービス統合 API のリクエスト本文には、以下の追加パラメータが含まれます。 • Body — コンテンツタイプのリクエストヘッダーで指定された形式で入力データを指定します。Body には、ターゲットモデル固有のパラメータが含まれます。 `InvokeModel` API を使用する場合は、Body パラメータを指定する必要があります。Step Functions で

Amazon Bedrock サービス統合 API	対応する Amazon Bedrock API	差異
		は、Body で入力した内容は検証されません。 最適化された Amazon Bedrock 統合を使用して Body を指定する場合、最大 256 KB のペイロードを指定できます。ペイロードが 256 KB を超える場合は、Input を使用することをお勧めします。 • Input — 入力データを取得するソースを指定します。このオプションフィールドは、Step Functions に最適化された Amazon Bedrock 統合に固有のものです。このフィールドでは、S3Uri を指定できます。 パラメータ または Input に Body を指定できますが、両方に指定することはできません。 ContentType を指定せずに Input を指定すると、入力データソースのコンテンツタイプが ContentType の値になります。 • Output — API レスポンスの記述先を指定します。このオプションフィールドは、Step Functions に最適

Amazon Bedrock サービス統合 API	対応する Amazon Bedrock API	差異
		化された Amazon Bedrock 統合に固有のものです。このフィールドでは、S3Uri を指定できます。 このフィールドを指定すると、API レスポンス本文は元の出力の Amazon S3 ロケーションへの参照に置き換えられます。 次の例は、Amazon Bedrock統合用 InvokeModel API の構文を示しています。 `{ "ModelId": String, // required "Accept": String, // default: application/json "ContentType": String, // default: application/json "Input": { // not from Bedrock API "S3Uri": String }, "Output": { // not from Bedrock API "S3Uri": String } }`

Amazon Bedrock サービス統合 API	対応する Amazon Bedrock API	差異
CreateModelCustomizationJob ベースモデルをカスタマイズするためのファインチューニングジョブを作成します。	CreateModelCustomizationJob	なし
CreateModelCustomizationJob.sync ベースモデルをカスタマイズするためのファインチューニングジョブを作成します。	CreateModelCustomizationJob	なし

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Amazon Bedrock 統合のタスクステート定義

以下のタスクステート定義は、ステートマシンで Amazon Bedrock とどのように統合できるかを示しています。この例は、パス (`result_one`) で指定されたモデル呼び出しの結果をすべて抽出するタスクステートを示しています。これは基盤モデルの推論パラメータに基づいています。この例では、Cohere Command 大規模言語モデル (LLM) を使用しています。

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::bedrock:invokeModel",
  "Parameters": {
    "ModelId": "cohere.command-text-v14",
    "Body": {
      "prompt.$": "$.prompt_one",
      "max_tokens": 250
    },
    "ContentType": "application/json",
    "Accept": "*/*"
  },
  "ResultPath": "$.result_one",
  "ResultSelector": {
```

```
    "result_one.$": "$.Body.generations[0].text"
  },
  "End": true
}
```

> **Tip**
>
> Amazon Bedrockと統合するステートマシンの例をにデプロイするには AWS アカウント、を参照してくださいAmazon Bedrock で AI プロンプトチェーンを実行する。

Step AWS CodeBuild Functions による呼び出し

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> **CodeBuild 最適化インテグレーションは CodeBuild AWS SDK インテグレーションとどう違うのか**
>
> - ジョブの実行 (.sync) 統合パターンがサポートされています。
> - StopBuildまたはを呼び出してもStopBuildBatch、 CodeBuild ビルドやビルドの状態を確定するための内部作業が完了するまで、ビルドまたはビルドバッチはすぐには削除できません。この期間中に BatchDeleteBuilds または DeleteBuildBatch を使おうとすると、構築または構築バッチは削除されない場合があります。BatchDeleteBuilds と DeleteBuildBatch のために最適化されたサービス統合には、内部再試行が含まれており、停止直後に削除するというユースケースを簡素化します。

AWS Step Functions AWS CodeBuild とのサービス統合により、Step Functions を使用してビルドのトリガー、停止、管理を行ったり、ビルドレポートを共有したりできます。Step Functions を使用すると、アプリケーションのソフトウェア変更を検証するための継続的統合パイプラインを設計および実行できます。

次の表に示すとおり、すべての API がすべての統合パターンをサポートしているわけではありません。

API	レスポンスのリクエスト	ジョブの実行 (.sync)
StartBuild	✓	✓
StopBuild	✓	
BatchDeleteBuilds	✓	
BatchGetReports	✓	
StartBuildBatch	✓	✓
StopBuildBatch	✓	
RetryBuildBatch	✓	✓
DeleteBuildBatch	✓	

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

サポートされている CodeBuild API と構文:

- `StartBuild`
  - リクエストの構文
  - サポートされているパラメータ:
    - `ProjectName`
    - `ArtifactsOverride`
    - `BuildspecOverride`
    - `CacheOverride`
    - `CertificateOverride`
    - `ComputeTypeOverride`

    - EncryptionKeyOverride
    - EnvironmentTypeOverride
    - EnvironmentVariablesOverride
    - GitCloneDepthOverride
    - GitSubmodulesConfigOverride
    - IdempotencyToken
    - ImageOverride
    - ImagePullCredentialsTypeOverride
    - InsecureSslOverride
    - LogsConfigOverride
    - PrivilegedModeOverride
    - QueuedTimeoutInMinutesOverride
    - RegistryCredentialOverride
    - ReportBuildStatusOverride
    - SecondaryArtifactsOverride
    - SecondarySourcesOverride
    - SecondarySourcesVersionOverride
    - ServiceRoleOverride
    - SourceAuthOverride
    - SourceLocationOverride
    - SourceTypeOverride
    - SourceVersion
    - TimeoutInMinutesOverride
  - レスポンスの構文
- StopBuild
  - リクエストの構文
  - サポートされているパラメータ:
    - Id
  - レスポンスの構文
- BatchDeleteBuilds

- リクエストの構文
- サポートされているパラメータ:
  - Ids
- レスポンスの構文

- BatchGetReports
  - リクエストの構文
  - サポートされているパラメータ:
    - ReportArns
  - レスポンスの構文
- StartBuildBatch
  - リクエストの構文
  - サポートされているパラメータ:
    - ProjectName
    - ArtifactsOverride
    - BuildBatchConfigOverride
    - BuildspecOverride
    - BuildTimeoutInMinutesOverride
    - CacheOverride
    - CertificateOverride
    - ComputeTypeOverride
    - DebugSessionEnabled
    - EncryptionKeyOverride
    - EnvironmentTypeOverride
    - EnvironmentVariablesOverride
    - GitCloneDepthOverride
    - GitSubmodulesConfigOverride
    - IdempotencyToken
    - ImageOverride
    - ImagePullCredentialsTypeOverride
    - InsecureSslOverride

    - LogsConfigOverride
    - PrivilegedModeOverride
    - QueuedTimeoutInMinutesOverride
    - RegistryCredentialOverride
    - ReportBuildBatchStatusOverride
    - SecondaryArtifactsOverride
    - SecondarySourcesOverride
    - SecondarySourcesVersionOverride
    - ServiceRoleOverride
    - SourceAuthOverride
    - SourceLocationOverride
    - SourceTypeOverride
    - SourceVersion
  - レスポンスの構文
- StopBuildBatch
  - リクエストの構文
  - サポートされているパラメータ:
    - Id
  - レスポンスの構文
- RetryBuildBatch
  - リクエストの構文
  - サポートされているパラメータ:
    - Id
    - IdempotencyToken
    - RetryType
  - レスポンスの構文
- DeleteBuildBatch
  - リクエストの構文
  - サポートされているパラメータ:
    - Id

- レスポンスの構文

> **Note**
>
> BatchDeleteBuilds には、JSONPath 再帰的降下 (..) 演算子を使用できます。これは配列を返し、次の例に示すように、Arn フィールドを StartBuild から複数形の Ids パラメータに変換できます。
>
> ```
> "BatchDeleteBuilds": {
>     "Type": "Task",
>     "Resource": "arn:aws:states:::codebuild:batchDeleteBuilds",
>     "Parameters": {
>         "Ids.$": "$.Build..Arn"
>     },
>     "Next": "MyNextState"
> },
> ```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions を使用した DynamoDB API の呼び出し

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

最適化された DynamoDB インテグレーションと DynamoDB AWS SDK インテグレーションの違い

- レスポンスのリクエスト 統合パターンの最適化はありません。
- タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。
- 最適化された統合では、GetItem、PutItem、UpdateItem、および DeleteItem API アクションのみを使用できます。などその他の API CreateTableアクションは、DynamoDB AWS SDK インテグレーションを使用して利用できます。

サポートされている Amazon DynamoDB API および構文:

- GetItem
  - リクエストの構文
  - サポートされているパラメータ:
    - Key
    - TableName
    - AttributesToGet
    - ConsistentRead
    - ExpressionAttributeNames
    - ProjectionExpression
    - ReturnConsumedCapacity
  - レスポンスの構文
- PutItem
  - リクエストの構文
  - サポートされているパラメータ:
    - Item
    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames

    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - レスポンスの構文
- DeleteItem
  - リクエストの構文
  - サポートされているパラメータ:
    - Key
    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - レスポンスの構文
- UpdateItem
  - リクエストの構文
  - サポートされているパラメータ:
    - Key
    - TableName
    - AttributeUpdates
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues

- ReturnConsumedCapacity
- ReturnItemCollectionMetrics
- ReturnValues
- UpdateExpression

- レスポンスの構文

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

DynamoDB からメッセージを取得する `Task` 状態を以下に示します。

```
"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"
},
```

この状態を実例で確認するには、データレコードの転送 (Lambda、DynamoDB、Amazon SQS) サンプルプロジェクトを参照してください。

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください。統合サービスの IAM ポリシー

Step Functions で Amazon ECS または Fargate タスクを管理する

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

最適化された Amazon ECS と Fargate のインテグレーションが Amazon ECS や Fargate SDK のインテグレーションとどう違うのか AWS

- ジョブの実行 (.sync) 統合パターンがサポートされています。
- `ecs:runTask` は、HTTP 200 のレスポンスを返すことができますが、次のように、空でない `Failures` フィールドがあります。
  - [リクエストレスポンス]: レスポンスを返し、タスクは失敗しません。これは、最適化なしと同じです。
  - [ジョブを実行する]: 空でない `Failures` フィールドが検出された場合、`AmazonECS.Unknown` エラーが発生し、タスクは失敗します。

サポートされている Amazon ECS/Fargate API と構文:

Note

ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

- `RunTask` は、指定されたタスク定義を使用して新しいタスクを開始します。
  - リクエストの構文
  - サポートされているパラメータ:
    - `Cluster`
    - `Group`
    - `LaunchType`
    - `NetworkConfiguration`
    - `Overrides`

- PlacementConstraints
- PlacementStrategy
- PlatformVersion
- PropagateTags
- TaskDefinition
- EnableExecuteCommand

- レスポンスの構文

Amazon ECS タスクにデータを渡す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

overrides を使用することで、コンテナのデフォルトコマンドを上書きし、入力を Amazon ECS タスクに渡すことができます。ContainerOverride を参照してください。この例では、TaskTask入力からステートに値を渡していました JsonPath 。

以下には、Amazon ECS タスクを実行し、完了するまで待機する Task 状態が含まれます。

```
{
 "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
```

```
    }
  }
}
```

ContainerOverrides の "Command.$": "$.commands" 行は状態の入力からコンテナにコマンドを渡します。

前の例では、実行への入力が次の場合、各コマンドはコンテナのオーバーライドとして渡されます。

```
{
  "commands": [
    "test command 1",
    "test command 2",
    "test command 3"
  ]
}
```

以下には Amazon ECS タスクを実行する Task 状態が含まれ、その後タスクトークンが返されるまで待機します。タスクトークンのコールバックまで待機する を参照してください。

```
{
   "StartAt":"Manage ECS task",
   "States":{
      "Manage ECS task":{
         "Type":"Task",
         "Resource":"arn:aws:states:::ecs:runTask.waitForTaskToken",
         "Parameters":{
            "LaunchType":"FARGATE",
            "Cluster":"cluster-arn",
            "TaskDefinition":"job-id",
            "Overrides":{
               "ContainerOverrides":[
                  {
                     "Name":"container-name",
                     "Environment":[
                        {
                           "Name":"TASK_TOKEN_ENV_VARIABLE",
                           "Value.$":"$$.Task.Token"
                        }
                     ]
                  }
               ]
```

```
            }
         },
         "End":true
      }
   }
}
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions を使用して Amazon EMR を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> **最適化された Amazon EMR インテグレーションと Amazon EMR AWS SDK インテグレーションの違い**
>
> 最適化された Amazon EMR サービス統合には、以下で説明するような基になる Amazon EMR API をラップするカスタマイズされた一連の API があります。このため、Amazon EMR AWS SDK サービスの統合とは大きく異なります。また、ジョブの実行 (.sync) 統合パターンがサポートされています。

Amazon EMR AWS Step Functions と統合するには、提供されている Amazon EMR サービス統合 API を使用します。サービス統合 API は対応する Amazon EMR API に似ていますが、渡されるフィールドと返される応答にいくつかの違いがあります。

Step Functions は、実行が停止しても Amazon EMR クラスターを自動的に終了しません。Amazon EMR クラスターが終了する前にステートマシンが停止した場合、クラスターは無期限に実行され、追加料金が発生する可能性があります。これを回避するには、作成した Amazon EMR クラスターが正しく終了していることを確認してください。詳細については、以下を参照してください。

- Amazon EMR ユーザーガイド のクラスター終了コントロール。
- サービス統合パターン ジョブの実行 (.sync) セクション。

> **Note**
>
> `emr-5.28.0` の時点で、クラスターの作成時に `StepConcurrencyLevel` パラメータを指定して、単一のクラスターで複数のステップを並行して実行することを許可します。Step Functions `Map` および `Parallel` 状態を使用して、並行して作業をクラスターに送信できます。

Amazon EMR サービス統合の可用性は、Amazon EMR API の可用性により決定します。特殊リージョンにおける制限については、Amazon EMR をチェックしてください。

> **Note**
>
> Amazon EMR との統合のため、Step Functions は最初の 10 分とその後の 300 秒間、ジョブポーリング頻度をハードコーディングして 60 秒に設定しています。

各サービス統合 API と対応する Amazon EMR API の違いを次の表に示します。

Amazon EMR サービス統合 API および対応する Amazon EMR API

Amazon EMR サービス統合 API	対応する EMR API	差異
createCluster 新しいクラスター (ジョブフロー) を作成して実行を開始します。 Amazon EMR はサービスリンクロールとして知られる IAM ロールの一意のタイプに直接リンクされています。`createCluster` と `createCluster.sync` が機能するには、サービスリンクロール `AWSServiceRoleForEMRCleanup`	runJobFlow	`createCluster` 以下を除いてと同じリクエスト構文を使用します。runJobFlow • このフィールド `Instances.KeepJobFlowAliveWhenNoSteps` は必須で、ブール値 `TRUE` である必要があります。 • フィールド `Steps` は許可されていません。 • フィールド `Instances.InstanceFleets[index].Name` は指定する

Amazon EMR サービス統合 API	対応する EMR API	差異

を作成するために必要なアクセス許可が設定されている必要があります。IAM 許可ポリシーに追加できるステートメントなど、この詳細については、Amazon EMR のサービスリンクロールを使用するを参照してください。		必要があり、オプションの `modifyInstanceFleetByName` コネクタ API を使用する場合は一意である必要があります。
		• フィールド `Instances.InstanceGroups[index].Name` は指定する必要があり、オプションの `modifyInstanceGroupByName` API を使用する場合は一意である必要があります。
		
		レスポンスは次のとおりです。
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR は以下を使用します。
		
		`{ "JobFlowId": "string" }`

createCluster.sync 新しいクラスター (ジョブフロー) を作成して実行を開始します。	runJobFlow	`createCluster` と同じですが、クラスターが `WAITING` 状態になるまで待機します。

Amazon EMR サービス統合 API	対応する EMR API	差異

setClusterTermination保護
 	
クラスター (ジョブフロー) をロックして、クラスター内の EC2 インスタンスをユーザーの介入、API コール、またはジョブフローエラーが発生した場合に終了できないようにします。
	setTerminationProtection	リクエストは以下を使用します。
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR は以下を使用します。
		
		`{ "JobFlowIds": ["string"] }`

terminateCluster
 	
クラスター (ジョブフロー) をシャットダウンします。
	terminateJobFlows	リクエストは以下を使用します。
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR は以下を使用します。
		
		`{ "JobFlowIds": ["string"] }`

terminateCluster.sync クラスター (ジョブフロー) をシャットダウンします。	terminateJobFlows	`terminateCluster` と同じですが、クラスターが終了するまで待機します。

Amazon EMR サービス統合 API	対応する EMR API	差異

addStep
 	
実行中のクラスターに新しいステップを追加します。
 	
オプションで、この API `ExecutionRoleArn` を使用する際に パラメータを指定することもできます。
	addJobFlow手順	リクエストはキー `"ClusterId"` を使用します。Amazon EMR は `"JobFlowId"` を使用します。リクエストは 1 つのステップを使用します。
		
		`{ "Step": <"StepConfig object"> }`
		
		Amazon EMR は以下を使用します。
		
		`{ "Steps": [<StepConfig objects>] }`
		
		レスポンスは次のとおりです。
		
		`{ "StepId": "string" }`
		
		Amazon EMR はこれを返します。
		
		`{ "StepIds": [<strings>] }`

Amazon EMR サービス統合 API	対応する EMR API	差異
addStep.sync 実行中のクラスターに新しいステップを追加します。 オプションで、この API `ExecutionRoleArn` を使用する際に パラメータを指定することもできます。	addJobFlowステップ	`addStep` と同じですが、ステップが完了するまで待機します。

Amazon EMR サービス統合 API	対応する EMR API	差異

cancelStep
 	
実行中のクラスターで保留中のステップを取り消します。
	cancelSteps	リクエストは以下を使用します。
		
		`{ "StepId": "string" }`
		
		Amazon EMR は以下を使用します。
		
		`{ "StepIds": [<strings >] }`
		
		レスポンスは次のとおりです。
		
		`{ "CancelStepsInfo": <CancelStepsInfo object> }`
		
		Amazon EMR は以下を使用します。
		
		`{ "CancelStepsInfoList": [<CancelStepsInfo objects>] }`

Amazon EMR サービス統合 API	対応する EMR API	差異
modifyInstanceFleetByName 指定された InstanceFleetName を使用して、インスタンスフリートのターゲットオンデマンドおよびターゲットスポット容量を変更します。	modifyInstanceFleet	リクエストは modifyInstanceFleet の場合と同じですが、以下が異なります。 • フィールド Instance.InstanceFleetId は許可されていません。 • 実行時に、InstanceFleetId は ListInstanceFleets を呼び出して結果を解析することにより、サービス統合によって自動的に決定されます。

Amazon EMR サービス統合 API	対応する EMR API	差異

modifyInstanceGroupByName
 	
インスタンスグループのノード数と構成設定を変更します。
	modifyInstanceGroups	リクエストは次のとおりです。
		
		`{ "ClusterId": "string", "InstanceGroup": <InstanceGroupModifyConfig object> }`
		
		Amazon EMR は以下のリストを使用します。
		
		`{ "ClusterId": ["string"], "InstanceGroups": [<InstanceGroupModifyConfig objects>] }`
		
		InstanceGroupModifyConfig オブジェクト内では、フィールド InstanceGroupId は使用できません。
		
		新しいフィールド InstanceGroupName が追加されました。実行時に、InstanceGroupId は ListInstanceGroups を呼び出して結果を解析することにより、サービス統合によって自動的に決定されます。

以下にはクラスターを作成する Task 状態が含まれています。

```
"Create_Cluster": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:createCluster.sync",
    "Parameters": {
        "Name": "MyWorkflowCluster",
        "VisibleToAllUsers": true,
        "ReleaseLabel": "emr-5.28.0",
        "Applications": [
            {
                "Name": "Hive"
            }
        ],
        "ServiceRole": "EMR_DefaultRole",
        "JobFlowRole": "EMR_EC2_DefaultRole",
        "LogUri": "s3n://aws-logs-123456789012-us-east-1/elasticmapreduce/",
        "Instances": {
            "KeepJobFlowAliveWhenNoSteps": true,
            "InstanceFleets": [
                {
                    "InstanceFleetType": "MASTER",
                    "Name": "MASTER",
                    "TargetOnDemandCapacity": 1,
                    "InstanceTypeConfigs": [
                        {
                            "InstanceType": "m4.xlarge"
                        }
                    ]
                },
                {
                    "InstanceFleetType": "CORE",
                    "Name": "CORE",
                    "TargetOnDemandCapacity": 1,
                    "InstanceTypeConfigs": [
                        {
                            "InstanceType": "m4.xlarge"
                        }
                    ]
                }
            ]
        }
    },
    "End": true
```

```
}
```

以下には終了保護を有効にする Task 状態が含まれています。

```
"Enable_Termination_Protection": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:setClusterTerminationProtection",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "TerminationProtected": true
    },
    "End": true
}
```

以下にはクラスターにステップを送信する Task 状態が含まれています。

```
"Step_One": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:addStep.sync",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/myEMR-execution-role",
        "Step": {
            "Name": "The first step",
            "ActionOnFailure": "CONTINUE",
            "HadoopJarStep": {
                "Jar": "command-runner.jar",
                "Args": [
                    "hive-script",
                    "--run-hive-script",
                    "--args",
                    "-f",
                    "s3://<region>.elasticmapreduce.samples/cloudfront/code/
Hive_CloudFront.q",
                    "-d",
                    "INPUT=s3://<region>.elasticmapreduce.samples",
                    "-d",
                    "OUTPUT=s3://<mybucket>/MyHiveQueryResults/"
                ]
            }
        }
    },
    "End": true
```

```
}
```

以下には、ステップをキャンセルする Task 状態が含まれます。

```
"Cancel_Step_One": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:cancelStep",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "StepId.$": "$.AddStepsResult.StepId"
    },
    "End": true
}
```

以下には、クラスターを終了する Task 状態を示します。

```
"Terminate_Cluster": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:terminateCluster.sync",
    "Parameters": {
        "ClusterId.$": "$.ClusterId"
    },
    "End": true
}
```

以下には、インスタンスグループに合わせてクラスターをスケールアップまたはスケールダウンする Task 状態が含まれています。

```

"ModifyInstanceGroupByName": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceGroupByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceGroupName": "MyCoreGroup",
        "InstanceGroup": {
            "InstanceCount": 8
        }
    },
    "End": true
}
```

以下には、インスタンスフリートに合わせてクラスターをスケールアップまたはスケールダウンする Task 状態が含まれています。

```
"ModifyInstanceFleetByName": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceFleetByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceFleetName": "MyCoreFleet",
        "InstanceFleet": {
            "TargetOnDemandCapacity": 8,
            "TargetSpotCapacity": 0
        }
    },
    "End": true
}
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

EKS のAmazon EMR に電話をかけるには AWS Step Functions

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

EKS での最適化された Amazon EMR と EKS SDK の統合での Amazon EMR との相違点 AWS

- ジョブの実行 (.sync) 統合パターンがサポートされています。
- レスポンスのリクエスト 統合パターンの最適化はありません。
- タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。

Note

Amazon EMR との統合のため、Step Functions は最初の 10 分とその後の 300 秒間、ジョブポーリング頻度をハードコーディングして 60 秒に設定しています。

EKS 上の Amazon EMR AWS Step Functions と統合するには、EKS 上の Amazon EMR サービス統合 API を使用してください。サービス統合 API は EKS API の対応する Amazon EMR と同じですが、次の表で示すとおり、すべての API があらゆる統合パターンをサポートしているわけではありません。

API	リクエストレスポンス	ジョブの実行 (.sync)
CreateVirtualCluster	✓	
DeleteVirtualCluster	✓	✓
StartJobRun	✓	✓

EKS API 上でサポートされる Amazon EMR は次のとおりです。

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

- `CreateVirtualCluster`
  - リクエストの構文
  - サポートされているパラメータ
  - レスポンスの構文
- `DeleteVirtualCluster`
  - リクエストの構文
  - サポートされているパラメータ
  - レスポンスの構文
- `StartJobRun`
  - リクエストの構文
  - サポートされているパラメータ

- レスポンスの構文

以下には仮想クラスターを作成する Task 状態が含まれています。

```
"Create_Virtual_Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-containers:createVirtualCluster",
  "Parameters": {
    "Name": "MyVirtualCluster",
    "ContainerProvider": {
      "Id": "EKSClusterName",
      "Type": "EKS",
      "Info": {
        "EksInfo": {
          "Namespace": "Namespace"
        }
      }
    }
  },
  "End": true
}
```

以下には、 ジョブを仮想クラスターに送信し、完了するまで待機する Task 状態が含まれます。

```
"Submit_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-containers:startJobRun.sync",
    "Parameters": {
      "Name": "MyJobName",
      "VirtualClusterId.$": "$.VirtualClusterId",
      "ExecutionRoleArn": "arn:aws:iam::<accountId>:role/job-execution-role",
      "ReleaseLabel": "emr-6.2.0-latest",
      "JobDriver": {
        "SparkSubmitJobDriver": {
          "EntryPoint": "s3://<mybucket>/jobs/trip-count.py",
          "EntryPointArguments": [
            "60"
          ],
          "SparkSubmitParameters": "--conf spark.driver.cores=2 --conf
 spark.executor.instances=10 --conf spark.kubernetes.pyspark.pythonVersion=3 --conf
 spark.executor.memory=10G --conf spark.driver.memory=10G --conf spark.executor.cores=1
 --conf spark.dynamicAllocation.enabled=false"
```

```
        }
      },
      "ConfigurationOverrides": {
        "ApplicationConfiguration": [
          {
            "Classification": "spark-defaults",
            "Properties": {
              "spark.executor.instances": "2",
              "spark.executor.memory": "2G"
            }
          }
        ],
        "MonitoringConfiguration": {
          "PersistentAppUI": "ENABLED",
          "CloudWatchMonitoringConfiguration": {
            "LogGroupName": "MyLogGroupName",
            "LogStreamNamePrefix": "MyLogStreamNamePrefix"
          },
          "S3MonitoringConfiguration": {
            "LogUri": "s3://<mylogsbucket>"
          }
        }
      },
      "Tags": {
        "taskType": "jobName"
      }
    },
    "End": true
}
```

以下には、仮想クラスターを削除し、削除が完了するまで待機する Task 状態が含まれます。

```
"Delete_Virtual_Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-containers:deleteVirtualCluster.sync",
  "Parameters": {
    "Id.$": "$.VirtualClusterId"
  },
  "End": true
}
```

IAMStep Functions AWS 他のサービスと併用する場合のアクセス権限の設定方法については、を参照してください。[統合サービスの IAM ポリシー]

Step Functions で Amazon EKS を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> 最適化された Amazon EKS インテグレーションと Amazon EKS AWS SDK インテグレーションの違い
>
> - ジョブの実行 (.sync) 統合パターンがサポートされています。
> - レスポンスのリクエスト 統合パターンの最適化はありません。
> - タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。

IAMStep Functions AWS 他のサービスと併用する場合のアクセス権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions は、Amazon Elastic Kubernetes Service と統合するため、2 種類のサービス統合 API を提供します。1 種類では、Amazon EKS API を使用して Amazon EKS クラスターを作成および管理できます。もう一方では、Kubernetes API を使用してクラスターと対話し、アプリケーションのワークフローの一部としてジョブを実行できるようにします。eksctl ツールまたは Amazon EKS コンソール、または同様の方法によって作成した Amazon EKS クラスターで Step Functions を使って作成した Amazon EKS クラスターとの Kubernetes API 統合を使用できます。詳細については、Amazon EKS ユーザーガイドの Amazon EKS クラスターを作成を参照してください。

> Note
>
> Step Functions EKS 統合は、公開エンドポイントにアクセスできる Kubernetes API だけをサポートします。デフォルトでは、EKS クラスター API サーバーエンドポイントはパブリックアクセスが可能です。詳細については、「Amazon EKS ユーザーガイド」の 「Amazon EKS クラスターエンドポイントアクセスコントロール」を参照してください。

Step Functions は、実行が停止しても、Amazon EKS クラスターを自動的に終了しません。Amazon EKS クラスターが終了する前にステートマシンが停止した場合、クラスターは無期限に実行され、追加料金が発生する可能性があります。これを回避するには、作成した Amazon EKS クラスターが正しく終了するようにしてください。詳細については、以下を参照してください。

- Amazon EKS ユーザーガイドのクラスターを削除。
- サービス統合パターンの ジョブの実行 (.sync)。

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

Kubernetes API 統合

Step Functions は、次の Kubernetes API をサポートしています。

RunJob

`eks:runJob` サービス統合によって、Amazon EKS クラスターでのジョブ実行が許可されます。`eks:runJob.sync` バリアントによって、ジョブの完了までの待機とオプションでの口グ取得が許可されます。

Kubernetes API サーバーは、ステートマシンで使用される IAM ロールにアクセス権限を付与する必要があります。詳細については、「アクセス許可」を参照してください。

ジョブを実行する (`.sync`) パターンの場合、ジョブの状態はポーリングによって決定されます。Step Functions は、最初は 1 分あたり約 1 ポーリングの割合でポーリングします。このレートは低速になり、最終的には 5 分ごとに約 1 ポーリングとなります。ポーリングの頻度を増やす必要がある場合、またはこれまでよりもポーリング戦略の制御を行う必要がある場合は、`eks:call` 統合を使って、ジョブの状態のクエリを出すことができます。

`eks:runJob` 統合は、`batch/v1` Kubernetes ジョブ固有のものです。詳細については、Kubernetes ドキュメントのジョブを参照してください。カスタムリソースを含む他の Kubernetes リソースを管理する場合は、`eks:call` サービス統合を使用します。the section called “Job ステータスの投票 (Lambda、) AWS Batch” サンプルプロジェクトで示すように、Step Functions を使用してポーリングループを構築できます。

サポートされるパラメータには次のものが含まれます。

- `ClusterName`: コールしたい Amazon EKS クラスター名。

  - Type: String
  - 必須: はい
- CertificateAuthority: ご自分のクラスターとの通信に必要な Base64 でエンコードされた証明書データ。この値は、Amazon EKS コンソールから、または Amazon EKS DescribeClusterAPI を使用して取得できます。
  - Type: String
  - 必須: はい
- Endpoint: Kubernetes API サーバーのエンドポイント。この値は、Amazon EKS コンソールから、または Amazon EKS DescribeClusterAPI を使用して取得できます。
  - Type: String
  - 必須: はい
- Namespace: ジョブを実行する名前空間。提供されない場合は、名前空間 default が用いられる。
  - Type: String
  - 必須: いいえ
- Job: Kubernetes Job の定義。Kubernetes ドキュメントのジョブを参照してください。
  - Type: JSON または String
  - 必須: はい
- LogOptions: ログのオプションの取得を制御するオプションのセット。[Run a Job] (ジョブの実行) (.sync) サービス統合パターンを使用してジョブの完了を待機するために使われるのが唯一適切です。
  - Type: JSON
  - 必須: いいえ
  - ログはキー logs の下のレスポンスに含まれます。ジョブ内には複数のポッドがあり、それぞれに複数のコンテナがあります。

```
{
  ...
  "logs": {
    "pods": {
      "pod1": {
        "containers": {
          "container1": {
            "log": <log>
```

```
          },
          ...
        }
      },
      ...
    }
  }
```

  - ログの取得は、ベストエフォートベースで実行されます。ログの取得エラーが発生した場合は、log フィールドの代わりに、error および cause フィールドが存在します。
- LogOptions.RetrieveLogs: ジョブの完了後にログの取得を有効にします。デフォルトでは、ログは取得されません。
  - Type: Boolean
  - 必須: いいえ
- LogOptions.RawLogs: RawLogs が true に設定されている場合、ログは JSON に解析しようとせずに生文字列として返されます。デフォルトでは、可能であれば、ログは JSON に逆シリアル化されます。場合によっては、このような解析によって、多くの桁数を含む数値の精度を制限するなど、望ましくない変更を導入しかねません。
  - Type: Boolean
  - 必須: いいえ
- LogOptions.LogParameters: Kubernetes API のログ読み取り API は、ログの取得を制御するクエリパラメータをサポートしています。例えば、tailLines または limitBytes を使用して、取得したログのサイズを制限し、Step Functions のデータサイズクォータ内に留まります。詳細については、SDK for Go API リファレンスのログを読み取りセクションを参照してください。
  - List of Strings への String の Type: マッピング
  - 必須: いいえ
  - 例：

```
"LogParameters": {
  "tailLines": [ "6" ]
}
```

次の例には、ジョブを実行し、完了を待ってから、ジョブのログを取得する Task 状態が含まれます。

```
{
```

```
  "StartAt": "Run a job on EKS",
  "States": {
    "Run a job on EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:runJob.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://AKIAIOSFODNN7EXAMPLE.yl4.us-east-1.eks.amazonaws.com",
        "LogOptions": {
          "RetrieveLogs": true
        },
        "Job": {
          "apiVersion": "batch/v1",
          "kind": "Job",
          "metadata": {
            "name": "example-job"
          },
          "spec": {
            "backoffLimit": 0,
            "template": {
              "metadata": {
                "name": "example-job"
              },
              "spec": {
                "containers": [
                  {
                    "name": "pi-2000",
                    "image": "perl",
                    "command": [ "perl" ],
                    "args": [
                      "-Mbignum=bpi",
                      "-wle",
                      "print bpi(2000)"
                    ]
                  }
                ],
                "restartPolicy": "Never"
              }
            }
          }
        }
      },
      "End": true
```

```
    }
  }
}
```

Call

eks:call サービス統合は、Kubernetes API の使用を許可し、Kubernetes API エンドポイントを経由して Kubernetes リソースオブジェクトを読み取り書き込みます。

Kubernetes API サーバーは、ステートマシンで使用される IAM ロールにアクセス権限を付与する必要があります。詳細については、「アクセス許可」を参照してください。

使用できる操作の詳細については、Kubernetes API リファレンスを参照してください。

Call 用にサポートされているパラメータには以下が含まれます。

- ClusterName: コールしたい Amazon EKS クラスター名。
  - Type: 文字列
  - 必須: はい
- CertificateAuthority: ご自分のクラスターとの通信に必要な Base64 でエンコードされた証明書データ。この値は、Amazon EKS コンソールから、または Amazon EKS DescribeClusterAPI を使用して取得できます。
  - Type: String
  - 必須: はい
- Endpoint: Kubernetes API サーバーのエンドポイント。この値は、Amazon EKS コンソールまたは Amazon EKS の DescribeCluster API を使用して確認できます。
  - Type: String
  - 必須: はい
- Method: リクエストの HTTP メソッド。GET、POST、PUT、DELETE、HEAD、PATCH のいずれかです。
  - Type: String
  - 必須: はい
- Path: Kubernetes REST API オペレーションの HTTP パス。
  - Type: String
  - 必須: はい
- QueryParameters: Kubernetes REST API オペレーションの HTTP クエリパラメーター。

- List of Strings への String の Type: マッピング
- 必須: いいえ
- 例 :

```
"QueryParameters": {
  "labelSelector": [ "job-name=example-job" ]
}
```

- RequestBody: Kubernetes REST API オペレーションの HTTP メッセージ本文。
  - Type: JSON または String
  - 必須: いいえ

以下には、eks:call を使用して、ジョブ example-job に所属するポッドをリストする Task 状態が含まれます。

```
{
  "StartAt": "Call EKS",
  "States": {
    "Call EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "GET",
        "Path": "/api/v1/namespaces/default/pods",
        "QueryParameters": {
          "labelSelector": [
            "job-name=example-job"
          ]
        }
      },
      "End": true
    }
  }
}
```

以下には、eks:call を使用して、ジョブ example-job を削除する Task 状態が含まれ、propagationPolicy をジョブのポッドも確実に削除されるよう設定します。

```
{
  "StartAt": "Call EKS",
  "States": {
    "Call EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "DELETE",
        "Path": "/apis/batch/v1/namespaces/default/jobs/example-job",
        "QueryParameters": {
          "propagationPolicy": [
            "Foreground"
          ]
        }
      },
      "End": true
    }
  }
}
```

サポートされる Amazon EKS API

サポートされている Amazon EKS API と構文には次のものが含まれます。

- CreateCluster
  - リクエストの構文
  - レスポンスの構文

    eks:createCluster サービス統合を使って Amazon EKS クラスターが作成され時点で、IAM ロールが Kubernetes RBAC 認可テーブルに (システム: マスター許可のある) 管理者として追加されます。初期状態では、その IAM エンティティのみが、Kubernetes API サーバーを呼び出すことができます。詳細については、以下を参照してください。

    - Amazon EKS ユーザーガイドのクラスターのユーザーまたは IAM ロールの管理
    - アクセス許可 セクション

    Amazon EKS は、ユーザーに代わって他のサービスを呼び出すために Amazon EKS が必要な許可を含むサービスにリンクされたロールを使用します。これらのサービスにリンクされ

たロールがアカウントにまだ存在しない場合は、Step Functions で使用される IAM ロールに対する `iam:CreateServiceLinkedRole` 許可を追加する必要があります。詳細については、Amazon EKS ユーザーガイドのサーバーがリンクしたロールを使用を参照してください。

Step Functions が使用する IAM ロールには、クラスタ IAM ロールを Amazon EKS に渡す `iam:PassRole` 許可がおりている必要があります。詳細については、Amazon EKS ユーザーガイドの Amazon EKS クラスターの IAM ロールを参照してください。

- `DeleteCluster`
  - リクエストの構文
  - レスポンスの構文

  クラスターを削除する前に、Fargate プロファイルまたはノードグループを削除する必要があります。
- `CreateFargateProfile`
  - リクエストの構文
  - レスポンスの構文

  Amazon EKS は、ユーザーに代わって他のサービスを呼び出すために Amazon EKS が必要な許可を含むサービスにリンクされたロールを使用します。これらのサービスにリンクされたロールがアカウントにまだ存在しない場合は、Step Functions で使用される IAM ロールに対する `iam:CreateServiceLinkedRole` 許可を追加する必要があります。詳細については、Amazon EKS ユーザーガイドのサーバーがリンクしたロールを使用を参照してください。

  Fargate 上の Amazon EKS はすべてのリージョンで利用可能なわけではありません。リージョンの可用性の詳細については、Fargate の Amazon EKS ユーザーガイドの Fargate に関するセクションを参照してください。

  Step Functions で使用されている IAM ロールには、ポッド実行 IAM ロールを Amazon EKS に渡す `iam:PassRole` 許可が必要です。詳細については、Amazon EKS ユーザーガイドのポッド実行ロールを参照してください。
- `DeleteFargateProfile`
  - リクエストの構文
  - レスポンスの構文
- `CreateNodegroup`
  - リクエストの構文
  - レスポンスの構文

Amazon ECS は、お客様に代わって Amazon EKS がその他の サービスを呼び出すために必要な許可を含むサービスにリンクされたロールを使用します。これらのサービスにリンクされたロールがアカウントにまだ存在しない場合は、Step Functions で使用される IAM ロールに対する `iam:CreateServiceLinkedRole` 許可を追加する必要があります。詳細については、Amazon EKS ユーザーガイドのサーバーがリンクしたロールを使用を参照してください。

Step Functions で使用される IAM ロールには、ノード IAM ロールを Amazon EKS に渡すため `iam:PassRole` 許可が必要です。詳細については、Amazon EKS ユーザーガイドのサーバーがリンクしたロールを使用を参照してください。

- DeleteNodegroup
  - リクエストの構文
  - レスポンスの構文

以下には Amazon EKS クラスターを作成する `Task` が含まれています。

```
{
  "StartAt": "CreateCluster.sync",
  "States": {
    "CreateCluster.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createCluster.sync",
      "Parameters": {
        "Name": "MyCluster",
        "ResourcesVpcConfig": {
          "SubnetIds": [
            "subnet-053e7c47012341234",
            "subnet-027cfea4b12341234"
          ]
        },
        "RoleArn": "arn:aws:iam::123456789012:role/MyEKSClusterRole"
      },
      "End": true
    }
  }
}
```

以下には Amazon EKS クラスターを削除する `Task` 状態が含まれています。

```
{
```

```
  "StartAt": "DeleteCluster.sync",
  "States": {
    "DeleteCluster.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteCluster.sync",
      "Parameters": {
        "Name": "MyCluster"
      },
      "End": true
    }
  }
}
```

以下には Fargate プロファイルを作成する Task 状態が含まれています。

```
{
  "StartAt": "CreateFargateProfile.sync",
  "States": {
    "CreateFargateProfile.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createFargateProfile.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "FargateProfileName": "MyFargateProfile",
        "PodExecutionRoleArn": "arn:aws:iam::123456789012:role/
MyFargatePodExecutionRole",
        "Selectors": [{
            "Namespace": "my-namespace",
            "Labels": { "my-label": "my-value" }
          }]
      },
      "End": true
    }
  }
}
```

以下には Fargate プロファイルを削除する Task 状態が含まれています。

```
{
  "StartAt": "DeleteFargateProfile.sync",
  "States": {
    "DeleteFargateProfile.sync": {
      "Type": "Task",
```

```
      "Resource": "arn:aws:states:::eks:deleteFargateProfile.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "FargateProfileName": "MyFargateProfile"
      },
      "End": true
    }
  }
}
```

以下にはノードグループを作成する Task 状態が含まれています。

```
{
  "StartAt": "CreateNodegroup.sync",
  "States": {
    "CreateNodegroup.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createNodegroup.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "NodegroupName": "MyNodegroup",
        "NodeRole": "arn:aws:iam::123456789012:role/MyNodeInstanceRole",
        "Subnets": ["subnet-09fb51df01234", "subnet-027cfea4b1234"]
      },
      "End": true
    }
  }
}
```

以下にはノードグループを削除する Task 状態が含まれています。

```
{
  "StartAt": "DeleteNodegroup.sync",
  "States": {
    "DeleteNodegroup.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteNodegroup.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "NodegroupName": "MyNodegroup"
      },
      "End": true
    }
```

```
  }
}
```

アクセス許可

eks:createCluster サービス統合を使って Amazon EKS クラスターが作成され時点で、IAM ロールを system:masters 許可を受けた管理者として、Kubernetes RBAC 認可テーブルに追加されています。初期状態では、その IAM エンティティのみが、Kubernetes API サーバーを呼び出すことができます。例えば、Step Functions ステートマシンと同じロールを引き受けるのでなければ、または追加の IAM エンティティに許可を付与するように Kubernetes を構成している場合は、kubectl を使用して、Kubernetes API と対話することはできません。詳細については、Amazon EKS ユーザーガイドのクラスター用ユーザーまたは IAM ロールを管理を参照してください。

kube-system 名前空間の中の aws-auth ConfigMap に追加して、ユーザーまたはロールなど、追加 IAM エンティティの許可を追加できます。Step Functions からクラスターを作成する場合は、eks:call サービス統合を使用します。

以下には、aws-auth ConfigMap を作成し、ユーザー arn:aws:iam::123456789012:user/my-user と IAM ロール arn:aws:iam::123456789012:role/my-role に system:masters 許可を付与する Task 状態が含まれます。

```
{
  "StartAt": "Add authorized user",
  "States": {
    "Add authorized user": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "LS0tLS1CRUd...UtLS0tLQo=",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "POST",
        "Path": "/api/v1/namespaces/kube-system/configmaps",
        "RequestBody": {
           "apiVersion": "v1",
           "kind": "ConfigMap",
           "metadata": {
              "name": "aws-auth",
              "namespace": "kube-system"
           },
           "data": {
```

```
            "mapUsers": "[{ \"userarn\": \"arn:aws:iam::123456789012:user/my-user\",
\"username\": \"my-user\", \"groups\": [ \"system:masters\" ] } ]",
            "mapRoles": "[{ \"rolearn\": \"arn:aws:iam::123456789012:role/my-role\",
\"username\": \"my-role\", \"groups\": [ \"system:masters\" ] } ]"
          }
        }
      },
      "End": true
    }
  }
```

> **Note**
>
> IAM ロールの ARN が、`arn:aws:iam::123456789012:role/service-role/my-role` などのパス /service-role/ を含む形式で表示されるのをご覧になるかもしれません。`aws-auth` でロールを一覧表示するときは、この service-role パストークンを含めないでください。

クラスターが最初に作成されるときは、`aws-auth ConfigMap` は存在しませんが、Fargate プロファイルを作成すると、自動的に追加されます。`aws-auth` の現在の値を取得し、追加許可を追加し、新しいバージョンを PUT します。通常、Fargate プロファイルの前に `aws-auth` を作成するほうが簡単です。

Step Functions の外部でクラスターが作成された場合は、kubectl を設定して、Kubernetes API サーバーと通信します。次に、`kubectl apply -f aws-auth.yaml` を使って新しい `aws-auth ConfigMap` を作成します。または、`kubectl edit -n kube-system configmap/aws-auth` を使用して既に存在するものを編集します。詳細については、以下を参照してください。

- Amazon EKS ユーザーガイドの Amazon EKS 用 kubeconfig を作成します。
- Amazon EKS ユーザーガイドのクラスターのユーザーまたは IAM ロールを管理。

Kubernetes で IAM ロールに十分な許可がない場合、`eks:call` または `eks:runJob` のサービス統合は、次のエラーで失敗します。

```
Error:
EKS.401

Cause:
```

```
{
  "ResponseBody": {
    "kind": "Status",
    "apiVersion": "v1",
    "metadata": {},
    "status": "Failure",
    "message": "Unauthorized",
    "reason": "Unauthorized",
    "code": 401
  },
  "StatusCode": 401,
  "StatusText": "Unauthorized"
}
```

Step Functions を使用して Amazon EMR Serverless を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

最適化した EMR Serverless 統合と EMR Serverless AWS SDK 統合との違い

- 最適化された EMR Serverless サービス統合には、基になる EMR Serverless API をラップする API のカスタマイズされたセットがあります。このカスタマイズにより、EMR Serverless最適化された統合は EMR Serverless AWS SDK サービスの統合とは大きく異なります。さらに、最適化された EMR Serverless 統合は ジョブの実行 (.sync) 統合パターンをサポートします。
- タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。

このトピックの内容

- EMR Serverless サービス統合 API
- EMR サーバーレス統合のユースケース

EMR Serverless サービス統合 API

AWS Step Functions を EMR Serverless と統合するために、以下の 6 つの EMR Serverless サービス統合 API を使用できます。これらのサービス統合 API は、対応する EMR Serverless API に似ていますが、渡されるフィールドと返される応答にいくつかの違いがあります。

各サービス統合 API と対応する EMR Serverless API の違いを次の表に示します。

EMR Serverless サービス統合 API と、対応する EMR Serverless API

EMR Serverless サービス統合 API	対応する EMR Serverless API	差異
createApplication アプリケーションを作成します。 EMR Serverless はサービスリンクロールとして知られる IAM ロールの一意のタイプにリンクされています。`createApplication` と `createApplication.sync` が機能するには、サービスリンクロール `AWSServiceRoleForAmazonEMRServerless` を作成するために必要なアクセス許可が設定されている必要があります。IAM アクセス許可ポリシーに追加できるステートメントなど、詳細については、「EMR Serverless のサービスにリンクされたロールの使用」を参照してください。	CreateApplication	なし
createApplication.sync	CreateApplication	EMR Serverless API と EMR Serverless サービス統合 API

EMR Serverless サービス統合 API	対応する EMR Serverless API	差異
アプリケーションを作成します。		のリクエストとレスポンスには違いはありません。ただし、createApplication.sync はアプリケーションが CREATED 状態に達するまで待機します。
startApplication 指定されたアプリケーションを起動し、設定されている場合はアプリケーションの初期容量を初期化します。	StartApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }`
startApplication.sync 指定されたアプリケーションを起動し、設定されている場合は初期容量を初期化します。	StartApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }` また、startApplication.sync はアプリケーションが STARTED 状態に達するまで待機します。

EMR Serverless サービス統合 API	対応する EMR Serverless API	差異
stopApplication 指定されたアプリケーションを停止し、設定されている場合は初期容量を解放します。アプリケーションを停止する前に、スケジュールされたジョブと実行中のジョブをすべて完了またはキャンセルする必要があります。	StopApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }`
stopApplication.sync 指定されたアプリケーションを停止し、設定されている場合は初期容量を解放します。アプリケーションを停止する前に、スケジュールされたジョブと実行中のジョブをすべて完了またはキャンセルする必要があります。	StopApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }` また、stopApplication.sync はアプリケーションが STOPPED 状態に達するまで待機します。

EMR Serverless サービス統合 API	対応する EMR Serverless API	差異
DeleteApplication アプリケーションを削除します アプリケーションを削除するには、そのアプリケーションが STOPPED または CREATED 状態になっている必要があります。	DeleteApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }`
deleteApplication.sync アプリケーションを削除します アプリケーションを削除するには、そのアプリケーションが STOPPED または CREATED 状態になっている必要があります。	DeleteApplication	EMR Serverless API レスポンスにはデータは含まれませんが、EMR Serverless サービス統合 API レスポンスには以下のデータが含まれます。 `{ "ApplicationId": "string" }` また、stopApplication.sync はアプリケーションが TERMINATED 状態に達するまで待機します。
startJobRun ジョブ実行を開始します。	StartJobRun	なし

EMR Serverless サービス統合 API	対応する EMR Serverless API	差異
startJobRun.sync ジョブ実行を開始します。	StartJobRun	EMR Serverless API と EMR Serverless サービス統合 API のリクエストとレスポンスには違いはありません。ただし、startJobRun.sync はアプリケーションがその状態になるまで待機します。SUCCESS
cancelJobRun ジョブ実行をキャンセルします。	CancelJobRun	なし
cancelJobRun.sync ジョブ実行をキャンセルします。	CancelJobRun	EMR Serverless API と EMR Serverless サービス統合 API のリクエストとレスポンスには違いはありません。ただし、cancelJobRun.sync はアプリケーションがその状態になるまで待機します。CANCELLED

EMR サーバーレス統合のユースケース

最適化された EMR Serverless サービス統合では、アプリケーションを 1 つ作成して、そのアプリケーションを使用して複数のジョブを実行することをお勧めします。たとえば、1 つのステートマシンに、startJobRun同じアプリケーションを使用する複数のリクエストを含めることができます。以下の タスク 状態の例は、API EMR Serverless を Step Functions と統合するユースケースを示しています。EMR Serverless のその他のユースケースの情報については、「Amazon EMR Serverless とは」を参照してください。

> **Tip**
>
> EMR Serverlessと統合して複数のジョブを実行するステートマシンの例をにデプロイするには AWS アカウント、を参照してくださいEMR Serverless ジョブを実行する。

- アプリケーションの作成
- アプリケーションの起動
- アプリケーションの停止
- アプリケーションの削除
- アプリケーションでのジョブの開始
- アプリケーションでのジョブのキャンセル

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

以下のユースケースの例で*#########*テキストを、リソース固有の情報に置き換えてください。たとえば、EMR Serverlessをアプリケーションの ID (など) *yourApplicationId*に置き換えてください`00yv7iv71inak893`。

アプリケーションの作成

次のタスクステートの例では、createApplication.sync サービス統合 API を使用してアプリケーションを作成しています。

```
"Create_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:createApplication.sync",
    "Parameters": {
        "Name": "MyApplication",
        "ReleaseLabel": "emr-6.9.0",
        "Type": "SPARK"
    },
    "End": true
}
```

アプリケーションの起動

次のタスク状態の例では、startApplication.sync サービス統合 API を使用してアプリケーションを起動します。

```
"Start_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:startApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

アプリケーションの停止

次のタスク状態の例では、stopApplication.sync サービス統合 API を使用してアプリケーションを停止します。

```
"Stop_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:stopApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

アプリケーションの削除

次のタスク状態の例では、deleteApplication.sync サービス統合 API を使用してアプリケーションを削除します。

```
"Delete_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:deleteApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

アプリケーションでのジョブの開始

次のタスクステートの例では、startJobRun.sync サービス統合 API を使用してアプリケーションでジョブを開始します。

```
"Start_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:startJobRun.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId",
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/myEMRServerless-execution-
role",
        "JobDriver": {
            "SparkSubmit": {
                "EntryPoint": "s3://<mybucket>/sample.py",
                "EntryPointArguments": ["1"],
                "SparkSubmitParameters": "--conf spark.executor.cores=4 --conf
 spark.executor.memory=4g --conf spark.driver.cores=2 --conf spark.driver.memory=4g --
conf spark.executor.instances=1"
            }
        }
    },
    "End": true
}
```

アプリケーションでのジョブのキャンセル

次のタスクステートの例では、cancelJobRun.sync サービス統合 API を使用してアプリケーション内のジョブをキャンセルします。

```
"Cancel_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:cancelJobRun.sync",
    "Parameters": {
        "ApplicationId.$": "$.ApplicationId",
        "JobRunId.$": "$.JobRunId"
    },
    "End": true
}
```

Step EventBridge Functions による呼び出し

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

EventBridge 最適化インテグレーションは EventBridge AWS SDK インテグレーションとどう違うのか

- 実行 ARN とステートマシン ARN は、各 `PutEventsRequestEntry` の `Resources` フィールドに自動的に追加されます。
- `PutEvents` からのレスポンスにゼロ以外の `FailedEntryCount` が含まれるのであれば、`Task` 状態はエラー `EventBridge.FailedEntry` で失敗します。

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions は Amazon EventBridge と統合するためのサービス統合 API を提供します。これにより、Step Functions ワークフローから直接カスタムイベントを送信して、イベント駆動型アプリケーションを構築できます。

`PutEvents`この API を使用するには、 EventBridge 送信するイベントの特定のパターンに一致するルールをアカウント内に作成する必要があります。例えば、次のことができます。

- ルールに一致するイベントを受信して出力する Lambda 関数をアカウント内に作成します。 EventBridge
- アカウントのデフォルトイベントバスで、特定のイベントパターンに一致し、Lambda EventBridge 関数を対象とするルールを作成します。

詳細については、以下を参照してください。

- 『 EventBridge ユーザーガイド』 PutEventsの Amazon EventBridge イベントの追加
- サービス統合パターンの タスクトークンのコールバックまで待機する。

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

サポートされている EventBridge API

サポートされている EventBridge API と構文には以下が含まれます。

- PutEvents
  - リクエストの構文
  - サポートされているパラメータ:
    - Entries
  - レスポンスの構文

以下には、カスタムイベントを送信する Task が含まれます。

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::events:putEvents",
  "Parameters": {
    "Entries": [
      {
        "Detail": {
          "Message": "MyMessage"
        },
        "DetailType": "MyDetailType",
        "EventBusName": "MyEventBus",
        "Source": "my.source"
      }
    ]
  },
  "End": true
}
```

エラー処理

PutEvents API はエントリの配列を入力として受け取り、結果エントリの配列を返します。PutEvents アクションが成功する限り、PutEvents は、1 つ以上のエントリが失敗した場合でも、HTTP 200 レスポンスを返します。PutEvents は FailedEntryCount フィールドの失敗したエントリの数を返します。

Step Functions、FailedEntryCount がゼロより大きいかどうかをチェックします。0 より大きい場合、Step Functions はエラー EventBridge.FailedEntry を使って状態を失敗させます。こうして、エントリに失敗した場合、キャッチまたは再試行のため、タスクの状態で Step Functions の組み込みエラー処理を使用できます。レスポンスから FailedEntryCount を分析する追加状態を使う必要はありません。

> **Note**
>
> べき等を実装し、すべてのエントリで安全に再試行できる場合は、Step Functions の再試行ロジックを使用できます。Step Functions は、再試行する前に、PutEvents 入力配列から成功したエントリを削除しません。代わりに、元のエントリの配列を使って再試行します。

Step Functions AWS Glue によるジョブの管理

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> **AWS Glue 最適化インテグレーションと AWS GlueAWS SDK インテグレーションの違い**
>
> - ジョブの実行 (.sync) 統合パターンが利用可能です。
> - JobName フィールドがリクエストから抽出され、レスポンスに挿入されます。通常は JobRunID のみが含まれます。

サポートされている AWS Glue API:

- StartJobRun

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

`Task` AWS Glue 以下にはジョブを開始する状態が含まれます。

```
"Glue StartJobRun": {
      "Type": "Task",
      "Resource": "arn:aws:states:::glue:startJobRun.sync",
      "Parameters": {
        "JobName": "GlueJob-JTrRO5l98qMG"
      },
      "Next": "ValidateOutput"
    },
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions AWS Glue DataBrew によるジョブの管理

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

DataBrew このインテグレーションを使用して、分析と機械学習のワークフローにデータクリーニングとデータ正規化のステップを追加できます。

サポートされている DataBrew API:

- `StartJobRun`

以下には、`Task` DataBrewリクエスト/レスポンスジョブを開始する状態が含まれます。

```
"DataBrew StartJobRun": {
            "Type": "Task",
            "Resource": "arn:aws:states:::databrew:startJobRun",
            "Parameters": {
```

```
            "Name": "sample-proj-job-1"
          },
          "Next": "NEXT_STATE"
        },
```

以下には、Task同期ジョブを開始する状態が含まれます。 DataBrew

```
"DataBrew StartJobRun": {
          "Type": "Task",
          "Resource": "arn:aws:states:::databrew:startJobRun.sync",
          "Parameters": {
             "Name": "sample-proj-job-1"
          },
          "Next": "NEXT_STATE"
        },
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions で Lambda を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> 最適化された Lambda インテグレーションと Lambda AWS SDK インテグレーションの違い
>
> - レスポンスの Payload フィールドは、エスケープされた Json から Json に解析されます。
> - レスポンスにフィールド FunctionError が含まれている場合、または Lambda 関数内で例外が発生した場合、タスクは失敗します。

状態の出入力および結果の管理については、Step Functions の入出力処理 を参照してください。

AWS Lambda サポートされている API:

- Invoke

- リクエストの構文
- サポートされているパラメータ
  - ClientContext
  - FunctionName
  - InvocationType
  - Qualifier
  - Payload
- レスポンスの構文

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

以下には、Lambda 関数を呼び出す Task 状態が含まれます。

```
{
   "StartAt":"CallLambda",
   "States":{
      "CallLambda":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-1:123456789012:function:MyFunction"
         },
         "End":true
      }
   }
}
```

以下には、コールバックサービス統合パターンを実行する Task の状態が含まれます。

```
{
   "StartAt":"GetManualReview",
   "States":{
      "GetManualReview":{
```

```
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-1:123456789012:function:get-model-
review-decision",
            "Payload":{
               "model.$":"$.new_model",
               "token.$":"$$.Task.Token"
            },
            "Qualifier":"prod-v1"
         },
         "End":true
      }
   }
}
```

Lambda 関数を呼び出すと、実行は関数の完了を待ちます。コールバックタスクで Lambda 関数を呼び出す場合、Lambda 関数の実行が完了して結果を返すまで、ハートビートタイムアウトのカウントは開始されません。Lambda 関数が実行されている限り、ハートビートタイムアウトは適用されません。

ただし、次の例に示されているとおり、InvocationType パラメータを使って非同期で Lambda を呼び出すこともできます。

> **Note**
>
> Lambda 関数の非同期呼び出しでは、ハートビートタイムアウト期間がすぐに開始されます。

```
{

  "Comment": "A Hello World example of the Amazon States Language using Pass states",
  "StartAt": "Hello",
  "States": {
    "Hello": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-1:123456789012:function:echo",
        "InvocationType": "Event"
      },
```

```
      "End": true
    }
  }
}
```

ときに Task の結果が返されると、関数の出力はメタデータのディクショナリ内にネストされます。例:

```
{

   "ExecutedVersion":"$LATEST",
   "Payload":"FUNCTION OUTPUT",
   "SdkHttpMetadata":{
      "HttpHeaders":{
         "Connection":"keep-alive",
         "Content-Length":"4",
         "Content-Type":"application/json",
         "Date":"Fri, 26 Mar 2021 07:42:02 GMT",
         "X-Amz-Executed-Version":"$LATEST",
         "x-amzn-Remapped-Content-Length":"0",
         "x-amzn-RequestId":"0101aa0101-1111-111a-aa55-1010aaa1010",
         "X-Amzn-Trace-Id":"root=1-1a1a000a2a2-fe0101aa10ab;sampled=0"
      },
      "HttpStatusCode":200
   },
   "SdkResponseMetadata":{
      "RequestId":"6b3bebdb-9251-453a-ae45-512d9e2bf4d3"
   },
   "StatusCode":200
}
```

または、「リソース」フィールドに関数 ARN を直接指定して Lambda 関数を呼び出せます。この方法で Lambda 関数を呼び出す場合、.waitForTaskToken を指定することはできず、タスク結果には関数の出力のみが含まれます。

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
         "Resource":"arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
         "End": true
```

```
        }
    }
}
```

Lambda 関数の特定バージョンまたはエイリアスを呼び出すには、`Resource` フィールドの ARN でそれらのオプションを指定します。Lambda ドキュメントで以下を参照してください。

- AWS Lambda バージョニング
- AWS Lambda エイリアス

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

SageMaker Step Functions による管理

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> SageMaker 最適化インテグレーションは SageMaker AWS SDK インテグレーションとどう違うのか
>
> - ジョブの実行 (.sync) 統合パターンがサポートされています。
> - レスポンスのリクエスト 統合パターンの最適化はありません。
> - タスクトークンのコールバックまで待機する 統合パターンはサポートされていません。

サポートされている SageMaker API と構文:

- `CreateEndpoint`
  - リクエストの構文
  - サポートされているパラメータ:
    - `EndpointConfigName`
    - `EndpointName`
    - `Tags`
  - レスポンスの構文

- CreateEndpointConfig
  - リクエストの構文
  - サポートされているパラメータ:
    - EndpointConfigName
    - KmsKeyId
    - ProductionVariants
    - Tags
  - レスポンスの構文
- CreateHyperParameterTuningJob

> **Note**
>
> この API アクションは .sync 統合パターンをサポートします。

  - リクエストの構文
  - サポートされているパラメータ:
    - HyperParameterTuningJobConfig
    - HyperParameterTuningJobName
    - Tags
    - TrainingJobDefinition
    - WarmStartConfig
  - レスポンスの構文
- CreateLabelingJob

> **Note**
>
> この API アクションは .sync 統合パターンをサポートします。

  - リクエストの構文
  - サポートされているパラメータ:
    - HumanTaskConfig

    - InputConfig
    - LabelAttributeName
    - LabelCategoryConfigS3Uri
    - LabelingJobAlgorithmsConfig
    - LabelingJobName
    - OutputConfig
    - RoleArn
    - StoppingConditions
    - Tags
  - レスポンスの構文
- CreateModel
  - リクエストの構文
  - サポートされているパラメータ:
    - Containers
    - EnableNetworkIsolation
    - ExecutionRoleArn
    - ModelName
    - PrimaryContainer
    - Tags
    - VpcConfig
- CreateProcessingJob

> **Note**
>
> この API アクションは .sync 統合パターンをサポートします。

  - リクエストの構文
  - サポートされているパラメータ:
    - AppSpecification
    - Environment
    - ExperimentConfig

    - NetworkConfig
    - ProcessingInputs
    - ProcessingJobName
    - ProcessingOutputConfig
    - ProcessingResources
    - RoleArn
    - StoppingCondition
    - Tags
  - レスポンスの構文
- CreateTrainingJob

> **Note**
>
> この API アクションは .sync 統合パターンをサポートします。

  - リクエストの構文
  - サポートされているパラメータ:
    - AlgorithmSpecification
    - HyperParameters
    - InputDataConfig
    - OutputDataConfig
    - ResourceConfig
    - RoleArn
    - StoppingCondition
    - Tags
    - TrainingJobName
    - VpcConfig
  - レスポンスの構文
- CreateTransformJob

> **Note**
>
> この API アクションは .sync 統合パターンをサポートします。

> **Note**
>
> AWS Step Functions のポリシーは自動的に作成されませんCreateTransformJob。インラインポリシーは、作成したロールにアタッチする必要があります。詳細については、次の IAM ポリシー例: CreateTrainingJob を参照してください。

- リクエストの構文
- サポートされているパラメータ:
  - BatchStrategy
  - Environment
  - MaxConcurrentTransforms
  - MaxPayloadInMB
  - ModelName
  - Tags
  - TransformInput
  - TransformJobName
  - TransformOutput
  - TransformResources
- レスポンスの構文

- UpdateEndpoint
  - リクエストの構文
  - サポートされているパラメータ:
    - EndpointConfigName
    - EndpointName
  - レスポンスの構文

SageMaker トランスフォームJob 例

以下には、Amazon Task SageMaker トランスフォームジョブを作成するステートが含まれており、DataSourceとの Amazon S3 TransformOutput ロケーションを指定しています。

```
{
"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"
},
```

SageMaker トレーニングJob 例

以下には、Amazon Task SageMaker トレーニングジョブを作成するステートが含まれます。

```
{
   "SageMaker CreateTrainingJob":{
      "Type":"Task",
      "Resource":"arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters":{
         "TrainingJobName":"search-model",
```

```
         "ResourceConfig":{
            "InstanceCount":4,
            "InstanceType":"ml.c4.8xlarge",
            "VolumeSizeInGB":20
         },
         "HyperParameters":{
            "mode":"batch_skipgram",
            "epochs":"5",
            "min_count":"5",
            "sampling_threshold":"0.0001",
            "learning_rate":"0.025",
            "window_size":"5",
            "vector_dim":"300",
            "negative_samples":"5",
            "batch_size":"11"
         },
         "AlgorithmSpecification":{
            "TrainingImage":"...",
            "TrainingInputMode":"File"
         },
         "OutputDataConfig":{
            "S3OutputPath":"s3://bucket-name/doc-search/model"
         },
         "StoppingCondition":{
            "MaxRuntimeInSeconds":100000
         },
         "RoleArn":"arn:aws:iam::123456789012:role/docsearch-stepfunction-iam-role",
         "InputDataConfig":[
            {
               "ChannelName":"train",
               "DataSource":{
                  "S3DataSource":{
                     "S3DataType":"S3Prefix",
                     "S3Uri":"s3://bucket-name/doc-search/interim-data/training-data/",
                     "S3DataDistributionType":"FullyReplicated"
                  }
               }
            }
         ]
      },
      "Retry":[
         {
            "ErrorEquals":[
               "SageMaker.AmazonSageMakerException"
```

```
            ],
            "IntervalSeconds":1,
            "MaxAttempts":100,
            "BackoffRate":1.1
         },
         {
            "ErrorEquals":[
               "SageMaker.ResourceLimitExceededException"
            ],
            "IntervalSeconds":60,
            "MaxAttempts":5000,
            "BackoffRate":1
         },
         {
            "ErrorEquals":[
               "States.Timeout"
            ],
            "IntervalSeconds":1,
            "MaxAttempts":5,
            "BackoffRate":1
         }
      ],
      "Catch":[
         {
            "ErrorEquals":[
               "States.ALL"
            ],
            "ResultPath":"$.cause",
            "Next":"Sagemaker Training Job Error"
         }
      ],
      "Next":"Delete Interim Data Job"
   }
}
```

SageMaker ラベル作成Job 例

以下には、Amazon Task SageMaker ラベリングジョブを作成するステートが含まれます。

```
{
  "StartAt": "SageMaker CreateLabelingJob",
```

```
  "TimeoutSeconds": 3600,
  "States": {
    "SageMaker CreateLabelingJob": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createLabelingJob.sync",
      "Parameters": {
        "HumanTaskConfig": {
          "AnnotationConsolidationConfig": {
            "AnnotationConsolidationLambdaArn": "arn:aws:lambda:us-
west-2:123456789012:function:ACS-TextMultiClass"
          },
          "NumberOfHumanWorkersPerDataObject": 1,
          "PreHumanTaskLambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:PRE-
TextMultiClass",
          "TaskDescription": "Classify the following text",
          "TaskKeywords": [
            "tc",
            "Labeling"
          ],
          "TaskTimeLimitInSeconds": 300,
          "TaskTitle": "Classify short bits of text",
          "UiConfig": {
            "UiTemplateS3Uri": "s3://s3bucket-example/TextClassification.template"
          },
          "WorkteamArn": "arn:aws:sagemaker:us-west-2:123456789012:workteam/private-
crowd/ExampleTesting"
        },
        "InputConfig": {
          "DataAttributes": {
            "ContentClassifiers": [
              "FreeOfPersonallyIdentifiableInformation",
              "FreeOfAdultContent"
            ]
          },
          "DataSource": {
            "S3DataSource": {
              "ManifestS3Uri": "s3://s3bucket-example/manifest.json"
            }
          }
        },
        "LabelAttributeName": "Categories",
        "LabelCategoryConfigS3Uri": "s3://s3bucket-example/labelcategories.json",
        "LabelingJobName": "example-job-name",
        "OutputConfig": {
```

```
          "S3OutputPath": "s3://s3bucket-example/output"
        },
        "RoleArn": "arn:aws:iam::123456789012:role/service-role/AmazonSageMaker-
ExecutionRole",
        "StoppingConditions": {
          "MaxHumanLabeledObjectCount": 10000,
          "MaxPercentageOfInputDatasetLabeled": 100
        }
      },
      "Next": "ValidateOutput"
    },
    "ValidateOutput": {
      "Type": "Choice",
      "Choices": [
        {
          "Not": {
            "Variable": "$.LabelingJobArn",
            "StringEquals": ""
          },
          "Next": "Succeed"
        }
      ],
      "Default": "Fail"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail",
      "Error": "InvalidOutput",
      "Cause": "Output is not what was expected. This could be due to a service outage
 or a misconfigured service integration."
    }
  }
}
```

SageMaker 処理Job 例

以下には、Amazon Task SageMaker 処理ジョブを作成するステートが含まれます。

```
{
  "StartAt": "SageMaker CreateProcessingJob Sync",
  "TimeoutSeconds": 3600,
```

```
  "States": {
    "SageMaker CreateProcessingJob Sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createProcessingJob.sync",
      "Parameters": {
        "AppSpecification": {
          "ImageUri": "737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-
learn:0.20.0-cpu-py3"
        },
        "ProcessingResources": {
          "ClusterConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.t3.medium",
            "VolumeSizeInGB": 10
          }
        },
        "RoleArn": "arn:aws:iam::123456789012:role/SM-003-
CreateProcessingJobAPIExecutionRole",
        "ProcessingJobName.$": "$.id"
      },
      "Next": "ValidateOutput"
    },
    "ValidateOutput": {
      "Type": "Choice",
      "Choices": [
        {
          "Not": {
            "Variable": "$.ProcessingJobArn",
            "StringEquals": ""
          },
          "Next": "Succeed"
        }
      ],
      "Default": "Fail"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail",
      "Error": "InvalidConnectorOutput",
      "Cause": "Connector output is not what was expected. This could be due to a
 service outage or a misconfigured connector."
    }
```

```
  }
}
```

IAMStep Functions AWS 他のサービスと併用する場合のアクセス権限の設定方法については、を参照してください統合サービスの IAM ポリシー。

Step Functions で Amazon SNS を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> 最適化された Amazon SNS インテグレーションは Amazon SNS SDK インテグレーションとどう違うのか AWS
>
> レスポンスのリクエスト または タスクトークンのコールバックまで待機する 統合パターンの最適化はありません。

サポートされている Amazon SNS API:

> **Note**
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

- `Publish`
  - リクエストの構文
  - サポートされているパラメータ
    - `Message`
    - `MessageAttributes`
    - `MessageStructure`
    - `PhoneNumber`
    - `Subject`

- TargetArn
- TopicArn

- レスポンスの構文

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

以下には、Amazon Simple Notification Service (Amazon SNS) トピックに発行する `Task` 状態が含まれます。

```
{
 "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",
       "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
    }
  }
}
```

動的な値を渡す。上記の例を変更して、この JSON ペイロードから属性を動的に渡すことができます。

```
{
"input": {
  "message": "Hello world"
},
"SNSDetails": {
  "attribute1": "some value",
  "attribute2": "some other value",
}
}
```

.$ を StringValue フィールドに追加:

```
"MessageAttributes": {
  "my_attribute_no_1": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute1"
  },
  "my_attribute_no_2": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute2"
  }
```

以下には、Amazon SNS トピックに発行され、その後タスクトークンが返されるまで待機する Task 状態が含まれます。タスクトークンのコールバックまで待機する を参照してください。

```
{
   "StartAt":"Send message to SNS",
   "States":{
      "Send message to SNS":{
         "Type":"Task",
         "Resource":"arn:aws:states:::sns:publish.waitForTaskToken",
         "Parameters":{
            "TopicArn":"arn:aws:sns:us-east-1:123456789012:myTopic",
            "Message":{
               "Input.$":"$",
               "TaskToken.$":"$$.Task.Token"
            }
         },
         "End":true
```

```
        }
    }
}
```

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください。統合サービスの IAM ポリシー

Step Functions で Amazon SQS を呼び出す

Step Functions AWS は特定のサービスを Amazon ステートメント言語 (ASL) から直接制御できます。詳細については、「他の サービスでの使用」および「サービス API にパラメータを渡す」を参照してください。

> 最適化された Amazon SQS インテグレーションが Amazon SQS SDK インテグレーションとどう違うのか AWS
>
> レスポンスのリクエスト または タスクトークンのコールバックまで待機する 統合パターンの最適化はありません。

サポートされているAmazon SQS API:

> Note
>
> Step Functions のタスクには入力データまたは結果データの最大サイズにはクォータがあります。これにより、別のサービスとの間でデータを送受信するときに、UTF-8 でエンコードされた文字列として 256 KB のデータに制限されます。ステートマシンの実行に関連するクォータ を参照してください。

- SendMessage

  サポートされているパラメータ:

  - DelaySeconds
  - MessageAttribute
  - MessageBody
  - MessageDeduplicationId
  - MessageGroupId

  - QueueUrl
- Response syntax

> **Note**
>
> ネイティブサービス API Step Functions が camelCase の場合でも PascalCase、のパラメータはで表現されます。例えば、Step Functions API アクション `startSyncExecution` を使用し、そのパラメータとして `StateMachineArn` を指定できます。

以下には、Amazon Simple Queue Service (Amazon SQS) メッセージを送信する `Task` 状態が含まれます。

```
{
 "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": "attribute1"
         },
         "my_attribute_no_2": {
           "DataType": "String",
           "StringValue": "attribute2"
         }
       }
     },
     "End": true
    }
  }
}
```

以下には Amazon SQS キューに発行してから、その後タスクトークンが返されるまで待機する `Task` 状態が含まれます。タスクトークンのコールバックまで待機する を参照してください。

```
{
   "StartAt":"Send message to SQS",
   "States":{
      "Send message to SQS":{
         "Type":"Task",
         "Resource":"arn:aws:states:::sqs:sendMessage.waitForTaskToken",
         "Parameters":{
            "QueueUrl":"https://sqs.us-east-1.amazonaws.com/123456789012/myQueue",
            "MessageBody":{
               "Input.$":"$",
               "TaskToken.$":"$$.Task.Token"
            }
         },
         "End":true
      }
   }
}
```

Amazon SQS におけるメッセージの受信の詳細については、Amazon Simple Queue Service デベロッパーガイドのメッセージの受信および削除を参照してください。

IAMStep Functions AWS 他のサービスと併用する場合の権限の設定方法については、を参照してください。統合サービスの IAM ポリシー

AWS Step Functions 実行を統合サービスとして管理

Step Functions は、サービス統合として独自の API と統合します。こうすることで、Step Functions は実行中のタスク状態から直接ステートマシンの新しい実行をスタートする許可を受けます。新しいワークフローを構築するときに、ネストされたワークフロー実行を使用して、メインワークフローの複雑さを軽減し、一般的なプロセスを再利用します。

最適化されたStep Functions の統合とStep Functions AWS SDK の統合との違い

- ジョブの実行 (.sync) 統合パターンが利用可能です。

  レスポンスのリクエスト または タスクトークンのコールバックまで待機する 統合パターンの最適化はないことに注意してください。

詳細については、以下を参照してください。

- タスクからの実行の開始
- 他の サービスでの使用
- サービス API にパラメータを渡す

サポートされている Step Functions API および構文:

- StartExecution
  - リクエストの構文
  - サポートされているパラメータ
    - Input
    - Name
    - StateMachineArn
  - レスポンスの構文

以下には、別のステートマシンの実行を開始し、その完了を待機する Task 状態が含まれています。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.sync:2",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!"
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

以下には、別のステートマシンの実行を開始する Task 状態が含まれています。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!"
```

```
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

以下には、コールバックサービス統合パターンを実行する Task の状態が含まれます。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.waitForTaskToken",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!",
        "token.$": "$$.Task.Token"
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

ネストされたワークフロー実行を、それを開始した親実行に関連付けるには、コンテキストオブジェクトからプルされた実行 ID を含む特別な名前のパラメータを渡します。ネストされた実行を開始するときは、AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID という名前のパラメータを使用します。パラメータ名に .$ を追加し、$$.Execution.Id を使用してコンテキストオブジェクトの ID を参照することで、実行 ID を渡します。詳細については、「コンテキストオブジェクトへのアクセス」を参照してください。

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.sync",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!",
        "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
       },
```

```
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

ネストされたステートマシンは、以下を返します。

リソース	出力
startExecution.sync	文字列
startExecution.sync:2	JSON

どちらもネストされたステートマシンが完了するのを待機しますが、異なる Output 形式を返します。例えば、オブジェクト { "MyKey": "MyValue" } を返す Lambda 関数を作成すると、次のレスポンスが得られます。

startExecution.sync の場合:

```
{
   <other fields>
   "Output": "{ \"MyKey\": \"MyValue\" }"
}
```

startExecution.sync:2 の場合:

```
{
   <other fields>
   "Output": {
      "MyKey": "MyValue"
   }
}
```

ネストされたステートマシンの IAM アクセス許可の設定

親ステートマシンは、ポーリングとイベントを使用して子ステートマシンが実行を完了したかどうかを判断します。states:DescribeExecutionポーリン

グには権限が必要ですが、Step Functions EventBridge に送信されるイベントに はevents:PutTargets、events:PutRule、events:DescribeRuleの権限が必要です。IAM ロールにこれらのアクセス許可がない場合、親ステートマシンが子ステートマシンの実行完了を認識するまで、遅延が発生する可能性があります。

1 つのネストされたワークフロー実行のために StartExecution を呼び出すステートマシンの場合は、そのステートマシンへの許可を制限する IIAM ポリシーを使用します。

詳細については、「IAM permissions for Step Functions」を参照してください。

サードパーティーの API を呼び出す

HTTP タスクは タスク ステートの一種で、Salesforce や Stripe などのパブリックなサードパーティー API をワークフローで呼び出すことができます。サードパーティー API を呼び出すには、arn:aws:states:::http:invoke リソースでタスクステートを使用します。次に、API URL、使用するメソッド、認証の詳細など、API エンドポイント構成の詳細を指定します。

Workflow Studio を使用して HTTP タスクを含むステートマシンを構築すると、Workflow Studio は HTTP タスクの IAM ポリシーを含む実行ロールを自動的に生成します。詳細については、「Workflow Studio で HTTP タスクをテストするためのロール」を参照してください。

トピック

- HTTP タスク定義
- HTTP タスクフィールド
- HTTP タスクの認証
- EventBridge 接続データと HTTP タスク定義データをマージする
- リクエスト本文に URL エンコーディングを適用します。
- HTTP タスクを実行するための IAM アクセス許可
- HTTP タスク例
- HTTP タスクのテスト
- サポートされていない HTTP タスクレスポンス

HTTP タスク定義

ASL 定義は、http:invoke リソースを含む HTTP タスクを表します。次の HTTP タスク定義は、すべての顧客のリストを返す Stripe API を呼び出します。

```
"Call third-party API": {
  "Type": "Task",
  "Resource": "arn:aws:states:::http:invoke",
  "Parameters": {
    "ApiEndpoint": "https://api.stripe.com/v1/customers",
    "Authentication": {
      "ConnectionArn": "arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
    },
    "Method": "GET"
  },
  "End": true
}
```

HTTP タスクフィールド

HTTP タスクの定義には以下のフィールドが含まれます。

`Resource` (必須)

タスクタイプを指定するには、`Resource` フィールドにで ARN を指定します。HTTP タスクの場合、`Resource` フィールドを次のように指定します。

```
"Resource": "arn:aws:states:::http:invoke"
```

`Parameters` (必須)

呼び出したいサードパーティー API に関する情報を提供する `ApiEndpoint`、`Method`、および `ConnectionArn` フィールドが含まれます。`Parameters` には `Headers` や `QueryParameters` などのオプションフィールドも含まれています。

`Parameters`フィールドのように`Parameters`、静的 JSON JsonPathと構文の組み合わせを指定できます。詳細については、「サービス API にパラメータを渡す」を参照してください。

`ApiEndpoint` (必須)

呼び出すサードパーティー API の URL を指定します。URL にクエリパラメータを追加するには、`QueryParameters` フィールドを使用します。次の例は、Stripe API を呼び出して、すべての顧客のリストを取得する方法を示しています。

```
"ApiEndpoint":"https://api.stripe.com/v1/customers"
```

JsonPath構文を使用して参照パスを指定し、サードパーティ API URL を含む JSON ノードを選択することもできます。例えば、特定の顧客 ID を使用して Stripe の API のいずれかを呼び出す場合を想定します。次のようなステートの入力を指定したとします。

```
{
    "customer_id": "1234567890",
    "name": "John Doe"
}
```

Stripe API を使用してこの顧客 ID の詳細を取得するには、次の例に示されているように、ApiEndpoint を指定します。この例では、組み込み関数と参照パスを使用しています。

```
"ApiEndpoint.$":"States.Format('https://api.stripe.com/v1/customers/{}',
 $.customer_id)"
```

実行時に、Step Functions は以下のように ApiEndpoint の値を解決します。

```
https://api.stripe.com/v1/customers/1234567890
```

Method (必須)

サードパーティー API の呼び出しに使用する HTTP メソッドを指定します。HTTP タスクでは、以下のいずれかのメソッドを指定できます。GET、POST、PUT、DELETE、PATCH、OPTIONS、または HEAD。

例えば、GET メソッドを使用するには、Method フィールドを次のように指定します。

```
"Method": "GET"
```

参照パスを使用して実行時にメソッドを指定することもできます。例えば **"Method.$": "$.myHTTPMethod"** です。

Authentication (必須)

サードパーティー API コールの認証方法を指定する ConnectionArn フィールドが含まれます。Step Functions は、Amazon EventBridge の接続リソースを使用して、指定された ApiEndpoint の認証をサポートします。

ConnectionArn (必須)

EventBridge 接続 ARN を指定します。

HTTP タスクには、API EventBridge プロバイダーの認証情報を安全に管理する接続が必要です。接続には、サードパーティー API の認証に使用する認証タイプと認証情報を指定します。接続を使用すると、API キーなどのシークレットをステートマシン定義にハードコーディングすることを避けることができます。接続では、Headers、QueryParameters、RequestBody パラメータを指定することもできます。

EventBridge 接続を作成するときは、認証の詳細を指定します。HTTP タスクでの認証の仕組みの詳細については、「HTTP タスクの認証」を参照してください。

次の例は、HTTP タスク定義で Authentication フィールドを指定する方法を示しています。

```
"Authentication": {
  "ConnectionArn": "arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
}
```

Headers (オプション)

API エンドポイントに追加のコンテキストとメタデータを提供します。ヘッダーは文字列または JSON 配列として指定できます。

ヘッダーは、EventBridge 接続と HTTP タスクの Headers フィールドで指定できます。API プロバイダへの認証情報を Headers フィールドに入力しないことをお勧めします。これらの情報は、EventBridge 接続に含めることをお勧めします。

Step Functions は EventBridge 接続で指定したヘッダーを HTTP タスク定義で指定したヘッダーに追加します。定義と接続に同じヘッダーキーが存在する場合、Step Functions はそれらのヘッダーに EventBridge 接続で指定された対応する値を使用します。Step Functions がデータマージを実行する方法の詳細については、「EventBridge 接続データと HTTP タスク定義データをマージする」を参照してください。

次の例では、以下のサードパーティー API コールに含まれるヘッダーを指定しています。content-type

```
"Headers": {
  "content-type": "application/json"
}
```

参照パスを使用して実行時にヘッダーを指定することもできます。例えば **"Headers.$": "$.myHTTPHeaders"** です。

Step Functions は、User-Agent、Range、Host ヘッダーを設定します。Step Functions は呼び出している API に基づいて Host ヘッダーの値を設定します。以下は、ヘッダーの例です。

```
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1,
Range: bytes=0-262144,
Host: api.stripe.com
```

HTTP タスク定義では以下のヘッダーは使用できません。これらのヘッダーを使用すると、HTTP タスクは States.Runtime エラーで失敗します。

- A-IM
- Accept-Charset
- Accept-Datetime
- Accept-Encoding
- Cache-Control
- Connection
- Content-Encoding
- Content−MD5
- 日付
- Expect
- Forwarded
- From
- ホスト
- HTTP2-Settings
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards

- オリジン
- Pragma
- Proxy-Authorization
- リファラー
- サーバー
- TE
- Trailer
- Transfer-Encoding
- Upgrade
- Via
- 警告
- x-forwarded-*
- x-amz-*
- x-amzn-*

`QueryParameters` (オプション)

API URL の末尾にキーと値のペアを挿入します。クエリパラメータは、文字列、JSON 配列、または JSON オブジェクトとして指定できます。Step Functions はサードパーティー API を呼び出すとき、クエリパラメータを自動的に URL エンコードします。

例えば、Stripe API を呼び出して、取引を米ドル (USD) で行っている顧客を検索したい場合を想定します。ステート入力として以下の `QueryParameters` を指定したとします。

```
"QueryParameters": {
  "currency": "usd"
}
```

実行時に、Step Functions は次のように `QueryParameters` を API URL に追加します。

```
https://api.stripe.com/v1/customers/search?currency=usd
```

参照パスを使用して実行時にクエリパラメータを指定することもできます。例えば **`"QueryParameters.$": "$.myQueryParameters"`** です。

EventBridge 接続でクエリパラメータを指定した場合、Step Functions はこれらのクエリパラメータを HTTP タスク定義で指定するクエリパラメータに追加します。定義と接続に同じク

エリパラメータキーが存在する場合、Step Functions はそれらのヘッダーに EventBridge 接続で指定されている対応する値を使用します。Step Functions がデータマージを実行する方法の詳細については、「EventBridge 接続データと HTTP タスク定義データをマージする」を参照してください。

`Transform` (オプション)

`RequestBodyEncoding` および `RequestEncodingOptions` フィールドが含まれます。デフォルトでは、Step Functions はリクエスト本文を JSON データとして API エンドポイントに送信します。

API プロバイダーが `form-urlencoded` リクエスト本文を受け入れる場合は、`Transform` フィールドを使用してリクエスト本文の URL エンコーディングを指定します。また、`content-type` ヘッダーを `application/x-www-form-urlencoded` として指定する必要があります。Step Functions は、リクエスト本文を自動的に URL エンコードします。

`RequestBodyEncoding`

リクエスト本文の URL エンコーディングを指定します。値は `NONE` または `URL_ENCODED` のいずれかを指定できます。

- `NONE` — HTTP リクエスト本文は、`RequestBody` フィールドのシリアル化された JSON になります。これは、デフォルト値です。
- `URL_ENCODED` — HTTP リクエスト本文は、`RequestBody` フィールドの URL エンコードされたフォームデータになります。

`RequestEncodingOptions`

`RequestBodyEncoding` を `URL_ENCODED` に設定した場合、リクエスト本文の配列に使用するエンコーディングオプションを決定します。

Step Functions では、次の配列エンコーディングオプションをサポートします。これらのオプションと例の詳細については、「リクエスト本文に URL エンコーディングを適用します。」を参照してください。

- `INDICES` — 配列要素のインデックス値を使用して配列をエンコードします。デフォルトでは、Step Functions はこのエンコーディングオプションを使用します。
- `REPEAT` — 配列内の各項目に対してキーを繰り返します。
- `COMMAS` — キー内のすべての値を、カンマで区切られた値のリストとしてエンコードします。
- `BRACKETS` — 配列内の各項目についてキーを繰り返し、そのキーに括弧 [] を追加して配列であることを示します。

次の例では、リクエスト本文データに URL エンコーディングを設定します。また、リクエスト本文の配列に COMMAS エンコーディングオプションを使用するように指定します。

```
"Transform": {
  "RequestBodyEncoding": "URL_ENCODED",
  "RequestEncodingOptions": {
    "ArrayFormat": "COMMAS"
  }
}
```

RequestBody (オプション)

ステート入力で指定した JSON データを受け入れます。ではRequestBody、静的 JSON JsonPathと構文の組み合わせを指定できます。例えば、以下の入力を使用するとします。

```
{
    "CardNumber": "1234567890",
    "ExpiryDate": "09/25"
}
```

実行時にこれらの CardNumber と ExpiryDate の値をリクエスト本文で使用するには、リクエスト本文に次の JSON データを指定できます。

```
"RequestBody": {
  "Card": {
    "Number.$": "$.CardNumber",
    "Expiry.$": "$.ExpiryDate",
    "Name": "John Doe",
    "Address": "123 Any Street, Any Town, USA"
  }
}
```

呼び出したいサードパーティー API に form-urlencoded リクエスト本文が必要な場合は、リクエスト本文データに URL エンコーディングを指定する必要があります。詳細については、「リクエスト本文に URL エンコーディングを適用します。」を参照してください。

HTTP タスクの認証

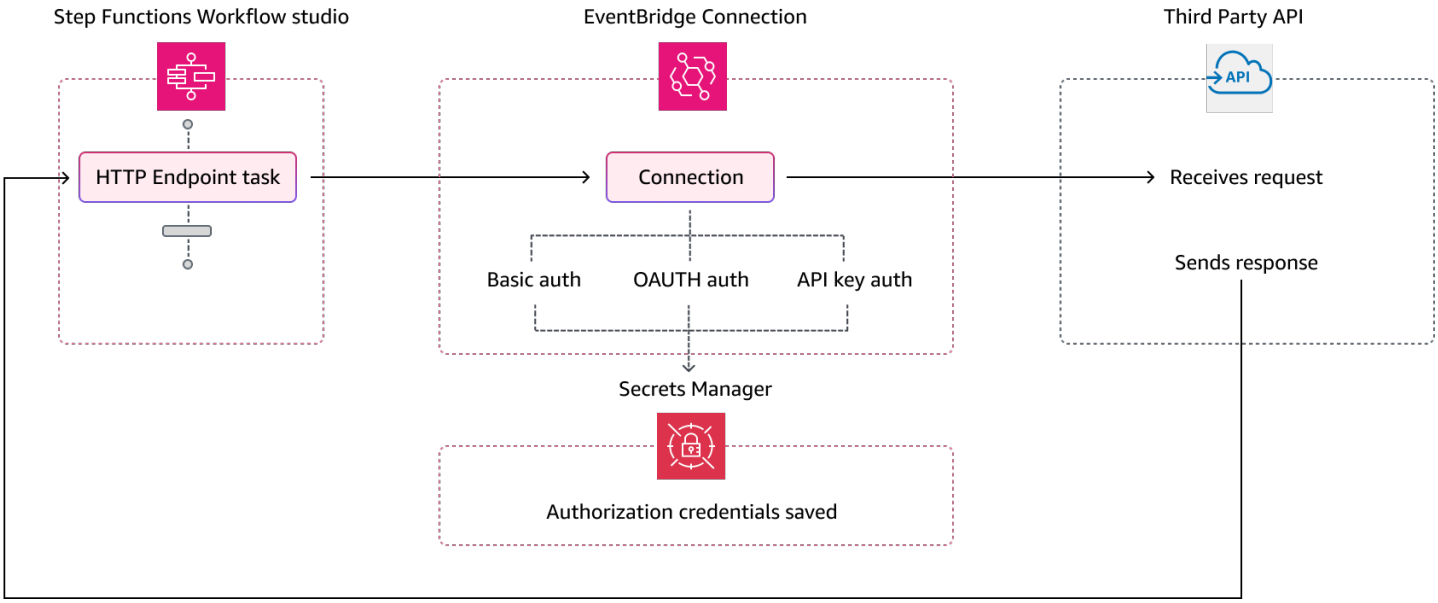
HTTP タスクには、API EventBridge プロバイダの認証情報を安全に管理する接続が必要です。接続には、サードパーティー API の認証に使用する認証タイプと認証情報を指定します。接続を使用

すると、API キーなどのシークレットをステートマシン定義にハードコーディングすることを避けることができます。 EventBridge 接続は、ベーシック、OAuth、および API キー認証スキームをサポートします。

EventBridge 接続を作成するときは、認証情報を入力します。API での認証に必要なヘッダー、本文、およびクエリパラメータを含めることもできます。サードパーティー API を呼び出す HTTP タスクには、接続 ARN を含める必要があります。

接続を作成して認証パラメータを追加すると、 EventBridge にシークレットが作成されます AWS Secrets Manager。 EventBridge このシークレットには、接続パラメータと認証パラメータが暗号化された形式で格納されます。接続を正常に作成または更新するには、Secrets Manager を使用する権限を持つを使用する必要があります。 AWS アカウント IAM EventBridge ステートマシンが接続にアクセスするために必要な権限の詳細については、を参照してくださいHTTP タスクを実行するための IAM アクセス許可。

次の図は、Step Functions が EventBridge 接続を使用してサードパーティー API 呼び出しの認証を処理する方法を示しています。



EventBridge 接続データと HTTP タスク定義データをマージする

HTTP タスクを呼び出すと、EventBridge 接続と HTTP タスク定義のデータを指定できます。このデータには `Headers`、`QueryParameters`、および `RequestBody` パラメータが含まれます。Step Functions は、リクエスト本文が文字列で接続本文のパラメータが空でない場合を除き、サードパーティー API を呼び出す前に、リクエスト本文と接続本文パラメータをマージします。この場合、HTTP タスクは `States.Runtime` エラーで失敗します。

HTTP タスク定義と EventBridge 接続に重複するキーが指定されている場合、Step Functions は HTTP タスクの値を接続の値で上書きします。

次のリストは、サードパーティー API Step Functions を呼び出す前にデータをマージする方法を示しています。

- ヘッダー — Step Functions は接続で指定したヘッダーを HTTP タスクの `Headers` フィールドのヘッダーに追加します。ヘッダーキーが競合する場合、Step Functions はそのヘッダーに接続で指定された値を使用します。例えば、HTTP タスク定義と EventBridge 接続の両方で `content-type` ヘッダーを指定した場合、Step Functions は接続で指定された `content-type` ヘッダー値を使用します。
- クエリパラメータ — Step Functions は接続で指定したクエリパラメータを HTTP タスクの `QueryParameters` フィールドのクエリパラメータに追加します。クエリパラメータキーが競合する場合は、Step Functions は接続で指定された値をクエリパラメータに使用します。例えば、HTTP タスク定義と EventBridge 接続の両方で `maxItems` クエリパラメータを指定した場合、Step Functions は接続で指定された `maxItems` クエリパラメータ値を使用します。
- Body パラメータ
  - Step Functions は接続で指定されたすべてのリクエスト本文値を HTTP タスクの `RequestBody` フィールドのリクエスト本文に追加します。リクエスト本文のキーが競合する場合は、Step Functions は接続で指定された値をリクエスト本文として使用します。例えば、HTTP タスク定義と EventBridge 接続の両方で `RequestBody` の `Mode` フィールドを指定したとします。Step Functions は接続で指定した `Mode` フィールド値を使用します。
  - リクエスト本文を JSON オブジェクトではなく文字列として指定し、EventBridge 接続にリクエスト本文も含まれている場合、Step Functions はこの 2 つの場所で指定されたリクエスト本文をマージすることはできません。HTTP タスクは `States.Runtime` エラーで失敗します。

  リクエスト本文のマージが完了したら、Step Functions はすべての変換を適用してリクエスト本文をシリアル化します。

次の例では、HTTP タスクと EventBridge 接続の両方に `Headers`、`QueryParameters`、および `RequestBody` フィールドを設定します。

HTTP タスク定義

```
{
  "Comment": "Data merging example for HTTP Task and EventBridge connection",
  "StartAt": "ListCustomers",
  "States": {
```

```
    "ListCustomers": {
      "Type": "Task",
      "Resource": "arn:aws:states:::http:invoke",
      "Parameters": {
        "Authentication": {
          "ConnectionArn": "arn:aws:events:us-
east-1:123456789012:connection/Example/81210c42-8af1-456b-9c4a-6ff02fc664ac"
        },
        "ApiEndpoint": "https:/example.com/path",
        "Method": "GET",
        "Headers": {
          "Request-Id": "my_request_id",
          "Header-Param": "state_machine_header_param"
        },
        "RequestBody": {
          "Job": "Software Engineer",
          "Company": "AnyCompany",
          "BodyParam": "state_machine_body_param"
        },
        "QueryParameters": {
          "QueryParam": "state_machine_query_param"
        }
      }
    }
  }
}
```

EventBridge 接続

```
{
  "AuthorizationType": "API_KEY",
  "AuthParameters": {
    "ApiKeyAuthParameters": {
      "ApiKeyName": "ApiKey",
      "ApiKeyValue": "key_value"
    },
    "InvocationHttpParameters": {
      "BodyParameters": [
        {
          "Key": "BodyParam",
          "Value": "connection_body_param"
        }
      ],
```

```
      "HeaderParameters": [
        {
          "Key": "Header-Param",
          "Value": "connection_header_param"
        }
      ],
      "QueryStringParameters": [
        {
          "Key": "QueryParam",
          "Value": "connection_query_param"
        }
      ]
    }
  }
}
```

この例では、HTTP タスクと EventBridge 接続に重複するキーが指定されています。そのため、Step Functions は HTTP タスクの値を接続の値で上書きします。次のコードスニペットは、Step Functions がサードパーティー API に送信する HTTP リクエストを示しています。

```
POST /path?QueryParam=connection_query_param HTTP/1.1
Apikey: key_value
Content-Length: 79
Content-Type: application/json; charset=UTF-8
Header-Param: connection_header_param
Host: example.com
Range: bytes=0-262144
Request-Id: my_request_id
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1

{"Job":"Software Engineer","Company":"AnyCompany","BodyParam":"connection_body_param"}
```

リクエスト本文に URL エンコーディングを適用します。

デフォルトでは、Step Functions はリクエスト本文を JSON データとして API エンドポイントに送信します。サードパーティー API プロバイダーが `form-urlencoded` リクエスト本文を必要とする場合は、リクエスト本文に URL エンコードを指定する必要があります。Step Functions は、選択した URL エンコードオプションに基づいてリクエスト本文を自動的に URL エンコードします。

URL エンコードは `Transform` を使用してフィールドを指定します。このフィールドには、リクエスト本文に URL エンコードを適用するかどうかを指定する `RequestBodyEncoding` フィールド

が含まれます。RequestBodyEncoding フィールドを指定すると、Step Functions はサードパーティー API を呼び出す前に JSON リクエスト本文を form-urlencoded リクエスト本文に変換します。URL でエンコードされたデータを受け入れる API は content-type ヘッダーを必要とするため、content-type ヘッダーも application/x-www-form-urlencoded として指定する必要があります。

リクエスト本文の配列をエンコードするために、Step Functions では以下の配列エンコードオプションが用意されています。

- INDICES — 配列内の各項目についてキーを繰り返し、そのキーに括弧 [] を追加して配列であることを示します。この括弧には配列要素のインデックスが含まれます。インデックスを追加すると、配列要素の順序を指定しやすくなります。デフォルトでは、Step Functions はこのエンコーディングオプションを使用します。

  例えば、リクエスト本文に次の配列が含まれているとします。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions はこの配列を次の文字列にエンコードします。

  ```
  array[0]=a&array[1]=b&array[2]=c&array[3]=d
  ```

- REPEAT — 配列内の各項目に対してキーを繰り返します。

  例えば、リクエスト本文に次の配列が含まれているとします。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions はこの配列を次の文字列にエンコードします。

  ```
  array=a&array=b&array=c&array=d
  ```

- COMMAS — キー内のすべての値を、カンマで区切られた値のリストとしてエンコードします。

  例えば、リクエスト本文に次の配列が含まれているとします。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions はこの配列を次の文字列にエンコードします。

```
array=a,b,c,d
```

- BRACKETS — 配列内の各項目についてキーを繰り返し、そのキーに括弧 [] を追加して配列であることを示します。

  例えば、リクエスト本文に次の配列が含まれているとします。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions はこの配列を次の文字列にエンコードします。

  ```
  array[]=a&array[]=b&array[]=c&array[]=d
  ```

HTTP タスクを実行するための IAM アクセス許可

ステートマシンの実行ロールには、HTTP タスクがサードパーティー API を呼び出すための `states:InvokeHTTPEndpoint`、`events:RetrieveConnectionCredentials`、`secretsmanager:G` および `secretsmanager:DescribeSecret` アクセス許可が必要です。次のIAMポリシーの例では、Stripe APIを呼び出すために必要な最小特権をステートマシンのロールに付与しています。このIAMポリシーは、SecSecrets Manager EventBridge に保存されているこの接続のシークレットを含め、特定の接続にアクセスする権限をステートマシンロールに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": "states:InvokeHTTPEndpoint",
            "Resource": "arn:aws:states:us-
east-2:123456789012:stateMachine:myStateMachine",
            "Condition": {
                "StringEquals": {
                    "states:HTTPMethod": "GET"
                },
                "StringLike": {
                    "states:HTTPEndpoint": "https://api.stripe.com/*"
                }
            }
```

```
        },
        {
            "Sid": "Statement2",
            "Effect": "Allow",
            "Action": [
                "events:RetrieveConnectionCredentials",
            ],
            "Resource": "arn:aws:events:us-
east-2:123456789012:connection/oauth_connection/aeabd89e-d39c-4181-9486-9fe03e6f286a"
        },
        {
            "Sid": "Statement3",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:events!connection/*"
        }
    ]
}
```

HTTP タスク例

次のステートマシン定義は Headers、QueryParameters、Transform、および RequestBody パラメータを含む HTTP タスクを示しています。HTTP タスクは Stripe API (https://api.stripe.com/v1/invoices) を呼び出して請求書を生成します。HTTP タスクでは、INDICES エンコーディングオプションを使用してリクエスト本文の URL エンコーディングも指定します。

EventBridge 接続が作成されていることを確認します。次の例は、BASIC 認証タイプを使用して作成された接続を示しています。

```
{
    "Type": "BASIC",
    "AuthParameters": {
        "BasicAuthParameters": {
            "Password": "myPassword",
            "Username": "myUsername"
         },
    }
}
```

######のテキストを、リソース固有の情報に必ず置き換えてください。

```
{
  "Comment": "A state machine that uses HTTP Task",
  "StartAt": "CreateInvoiceAPI",
  "States": {
    "CreateInvoiceAPI": {
      "Type": "Task",
      "Resource": "arn:aws:states:::http:invoke",
      "Parameters": {
        "ApiEndpoint": "https://api.stripe.com/v1/invoices",
        "Method": "POST",
        "Authentication": {
          "ConnectionArn": ""arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
        },
        "Headers": {
          "Content-Type": "application/x-www-form-urlencoded"
        },
        "RequestBody": {
          "customer.$": "$.customer_id",
          "description": "Monthly subscription",
          "metadata": {
            "order_details": "monthly report data"
          }
        },
        "Transform": {
          "RequestBodyEncoding": "URL_ENCODED",
          "RequestEncodingOptions": {
            "ArrayFormat": "INDICES"
          }
        }
      },
      "Retry": [
        {
          "ErrorEquals": [
            "States.Http.StatusCode.429",
            "States.Http.StatusCode.503",
            "States.Http.StatusCode.504",
            "States.Http.StatusCode.502"
          ],
          "BackoffRate": 2,
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
```

```
          "JitterStrategy": "FULL"
        }
      ],
      "Catch": [
        {
          "ErrorEquals": [
            "States.Http.StatusCode.404",
            "States.Http.StatusCode.400",
            "States.Http.StatusCode.401",
            "States.Http.StatusCode.409",
            "States.Http.StatusCode.500"
          ],
          "Comment": "Handle all non 200 ",
          "Next": "HandleInvoiceFailure"
        }
      ],
      "End": true
    }
  }
}
```

このステートマシンを実行するには、次の例のように顧客 ID を入力として指定します。

```
{
    "customer_id": "1234567890"
}
```

次の例は、Stripe API Step Functions に送信する HTTP リクエストを示しています。

```
POST /v1/invoices HTTP/1.1
Authorization: Basic <base64 of username and password>
Content-Type: application/x-www-form-urlencoded
Host: api.stripe.com
Range: bytes=0-262144
Transfer-Encoding: chunked
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1

description=Monthly%20subscription&metadata%5Border_details%5D=monthly%20report
%20data&customer=1234567890
```

HTTP タスクのテスト

コンソール、SDK、またはを通じて TestStateAPI を使用して HTTP タスクをテストできます。AWS CLI 以下の手順では、Step Functionsコンソールで TestState API を使用する方法について説明します。HTTP タスクが期待どおりに動作するまで、API リクエスト、応答、認証の詳細を繰り返しテストできます。

HTTP タスクステートを Step Functions コンソールでテストする

1. Step Functions コンソールを開きます。
2. [ステートマシンの作成] を選択してステートマシンの作成を開始するか、HTTP タスクを含む既存のステートマシンを選択します。

   既存のステートマシンでタスクをテストする場合は、ステップ 4 を参照してください。
3. デザインモード の Workflow Studio で、HTTP タスクを視覚的に設定します。または、コードモードを選択して、ローカル開発環境からステートマシン定義をコピーして貼り付けます。
4. デザインモードで、Workflow Studio の Inspector パネルで [ステートをテスト] を選択します。
5. [ステートをテスト] ダイアログボックスで、次の操作を行います。

   a. [実行ロール] では、ステートをテストする実行ロールを選択します。HTTP タスクに必要なアクセス許可を持つロールがない場合は、ロールを作成するために「Workflow Studio で HTTP タスクをテストするためのロール」を参照してください。

   b. (オプション) 選択したステートでテストに必要な JSON 入力を入力します。

   c. [検査レベル] は、デフォルトの [INFO] のままにします。このレベルには、API 呼び出しのステータスと状態出力が表示されます。API レスポンスをすばやく確認するのに便利です。

   d. [テストを開始] を選択します。

   e. テストが成功すると、[ステートをテスト] ダイアログボックスの右側にステートの出力が表示されます。テストに失敗すると、エラーが表示されます。

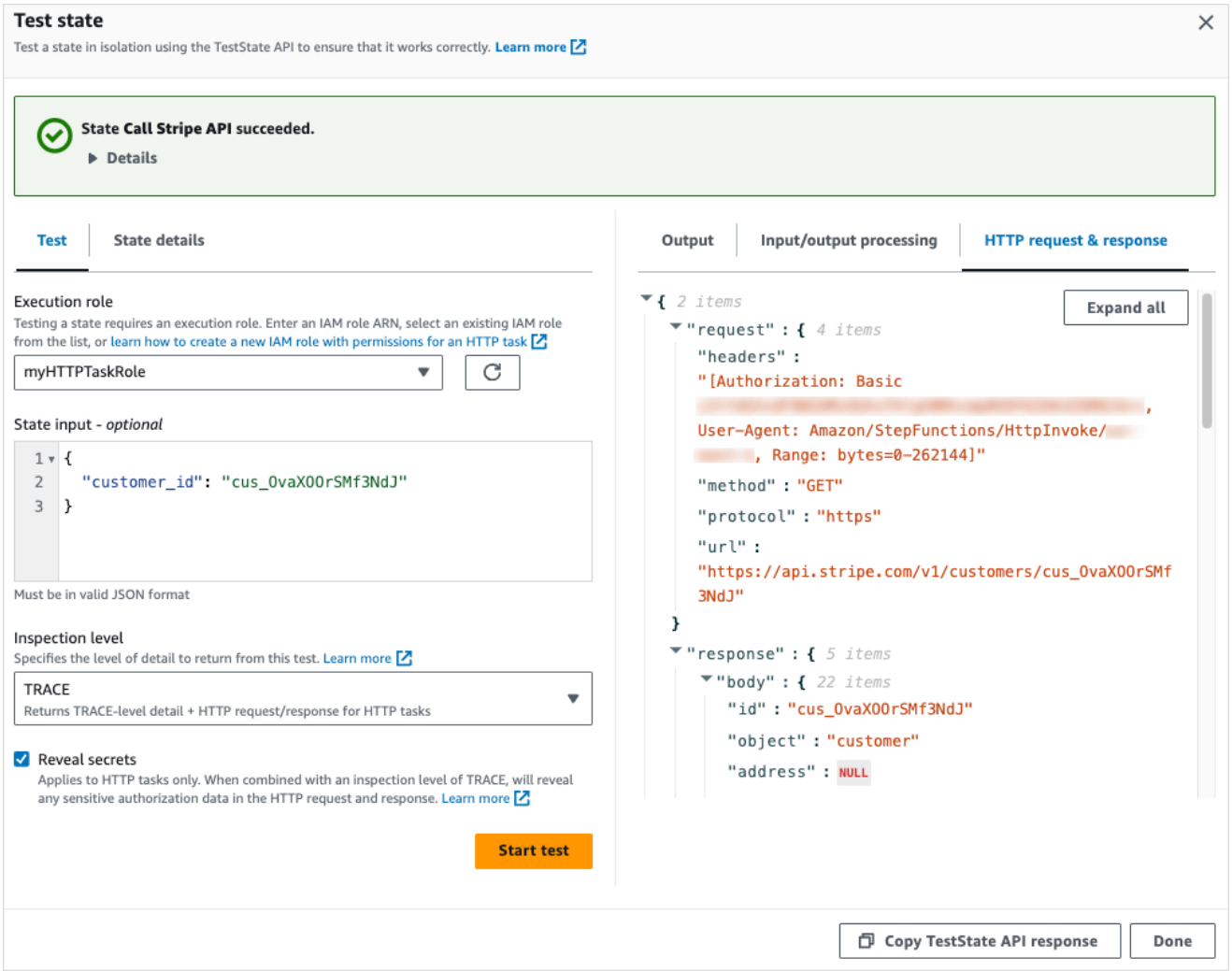
      ダイアログボックスの [ステートの詳細] タブには、ステート定義と EventBridge 接続へのリンクが表示されます。

   f. [検査レベル] を [TRACE] に変更します。このレベルは未加工の HTTP リクエストと応答を表示し、ヘッダー、クエリパラメータ、その他の API 固有の詳細を検証するのに役立ちます。

   g. [シークレットを公開] チェックボックスを選択します。この設定を [TRACE] と組み合わせると、API キーなど、EventBridge 接続によって挿入される機密データを確認できます。

コンソールへのアクセスに使用する IAM ユーザー ID には、`states:RevealSecrets` アクションを実行するアクセス許可が必要です。このアクセス許可がない場合、Step Functions はテストの開始時にアクセス拒否エラーをスローします。これらの `states:RevealSecrets` アクセス許可を付与する IAM ポリシーの例については、「IAM TestState API を使用するための アクセス許可」を参照してください。

次の図は、HTTP タスクが成功するかどうかのテストを示しています。このステートの [検査レベル] は [TRACE] に設定されています。以下の画像の [HTTP リクエストおよびレスポンス] タブには、サードパーティー API コールの結果が表示されます。



h. [テストを開始] を選択します。

i. テストが成功すると、[HTTP リクエストおよびレスポンス] タブに HTTP の詳細が表示されます。

サポートされていない HTTP タスクレスポンス

返された応答が次のいずれかの条件に当てはまる場合、HTTP タスクは `States.Runtime` エラーで失敗します。

- レスポンスには、`application/octet-stream`、`image/*`、`video/*`、または `audio/*` というコンテンツタイプヘッダーが含まれています。
- 応答は有効な文字列として読み取ることができません。例えば、バイナリデータや画像データなどです。

サービス統合パターン

AWS Step Functions Amazon ステート言語でサービスと直接統合されます。3 つの異なるサービス統合パターンを使用して、これらの AWS のサービスを制御できます。

- サービスを呼び出し、Step Functions が HTTP レスポンスを取得した直後に次の状態に進むことができるようにします。
- サービスを呼び出し、ジョブが完了するまで Step Functions が待機するようにします。
- タスクトークンでサービスを呼び出し、トークンがペイロードとともに返されるまで Step Functions が待機するようにします。

これらの各サービス統合パターンは、[task definition] (タスク定義) の `"Resource"` フィールドで URI を作成する方法によって制御されます。

統合されたサービスを呼び出す方法

- レスポンスのリクエスト
- ジョブの実行 (.sync)
- タスクトークンのコールバックまで待機する

統合サービスの設定 AWS Identity and Access Management (IAM) については、を参照してください。統合サービスの IAM ポリシー

レスポンスのリクエスト

タスク状態の "Resource" 文字列でサービスを指定する時にリソースを提供するだけで、Step Functions は HTTP レスポンスがあるまで待機した後、次の状態に進みます。Step Functions は、ジョブが完了するまで待機しません。

次の例は Amazon SNS トピックを発行する方法を示しています。

```
"Send message to SNS":{
   "Type":"Task",
   "Resource":"arn:aws:states:::sns:publish",
   "Parameters":{
      "TopicArn":"arn:aws:sns:us-east-1:123456789012:myTopic",
      "Message":"Hello from Step Functions!"
   },
   "Next":"NEXT_STATE"
}
```

この例では、Amazon SNS の Publish API を参照しています。Publish API の呼び出し後、ワークフローは次の状態に進みます。

> **Tip**
>
> Request Response サービスの統合パターンを使用するサンプルワークフローをにデプロイするには AWS アカウント、「モジュール 2- AWS Step Functions ワークショップのリクエストレスポンス」を参照してください。

ジョブの実行 (.sync)

AWS Batch や Amazon ECS などの統合サービスの場合、Step Functions はリクエストが完了するのを待ってから次の状態に進むことができます。Step Functions を待機させるには、タスクの状態定義で "Resource" フィールドを指定し、リソース URI の後に .sync サフィックスを追加します。

たとえば、 AWS Batch ジョブを送信するときは、"Resource"この例のようにステートマシン定義のフィールドを使用します。

```
"Manage Batch task": {
  "Type": "Task",
```

```
  "Resource": "arn:aws:states:::batch:submitJob.sync",
  "Parameters": {
    "JobDefinition": "arn:aws:batch:us-east-2:123456789012:job-definition/
testJobDefinition",
    "JobName": "testJob",
    "JobQueue": "arn:aws:batch:us-east-2:123456789012:job-queue/testQueue"
  },
  "Next": "NEXT_STATE"
}
```

リソース Amazon リソースネーム (ARN) に追加される .sync 部分があるため、ジョブが完了するまで Step Functions が待機します。 AWS Batch submitJob の呼び出し後、ワークフローは停止します。ジョブが完了すると、Step Functions は次の状態に進みます。詳細については、 AWS Batch サンプルプロジェクトを参照してください。バッチジョブの管理 (AWS Batch、Amazon SNS)

この (.sync) サービス統合パターンを使用するタスクが中止となり、Step Functions がタスクをキャンセルできない場合、統合サービスから追加料金が発生する可能性があります。タスクは次の場合に中断できます。

- ステートマシンの実行を停止するには
- 並行状態の別のブランチは、キャッチされないエラーで失敗します。
- マップ状態の反復は、キャッチされないエラーで失敗します。

Step Functions は、ベストエフォートでタスクをキャンセルしようとします。例えば、Step Functions states:startExecution.sync タスクが中断されれば、Step Functions StopExecution API アクションを呼び出します。ただし、Step Functions がタスクをキャンセルできない可能性があります。考えられる理由を次に示します (ただし、これらに限定されるものではありません)。

- IAM 実行ロールには、対応する API コールを行う許可がありません。
- 一時的なサービス停止が発生しました。

.sync サービス統合パターンを使用すると、Step Functions は割り当てられたクォータとイベントを消費するポーリングを使用してジョブのステータスをモニタリングします。.sync同じアカウント内での呼び出しの場合、Step Functions EventBridge はイベントを使用して、ステートで指定した API をポーリングします。Taskクロスアカウント .sync 呼び出しの場合、Step Functions はポーリングのみを使用します。たとえば、の場合states:StartExecution.sync、Step Functions は DescribeExecutionAPI に対してポーリングを実行し、割り当てられたクォータを使用します。

> **Tip**
>
> Run a Job (.sync) サービス統合パターンを使用するサンプルワークフローをにデプロイするには AWS アカウント、「The Workshop」の「モジュール 3-Job ブの実行 (.sync)」を参照してください。 AWS Step Functions

どの統合されたサービスでジョブの完了の待機 (`.sync`) がサポートされているかのリストについては、Step Functions 用統合最適化 を参照してください。

> **Note**
>
> `.sync` を使用するサービス統合には追加の IAM 許可が必要です。詳細については、「統合サービスの IAM ポリシー」を参照してください。

場合によっては、ジョブが完全に完了する前に Step Functions でワークフローを続行させたい場合もあります。これは、タスクトークンのコールバックまで待機する サービス統合パターンを使用する場合と同じ方法で実現できます。そのためには、タスクトークンをジョブに渡し、`SendTaskSuccess` または `SendTaskFailure` API 呼び出しを使用してそれを返します。Step Functions は、その呼び出しで指定したデータを使用してタスクを完了し、ジョブのモニタリングを停止し、ワークフローを続行します。

タスクトークンのコールバックまで待機する

コールバックタスクは、タスクトークンが返されるまでワークフローを待機させる方法を提供します。タスクには、人間による承認、サードパーティーとの統合、あるいはレガシーシステムの呼び出しまで待機することが必要になる場合があります。このようなタスクでは、ワークフローの実行が 1 年間のサービスクォータに達するまで Step Functions を一時停止して (状態のスロットリングに関連するクォータ を参照)、外部のプロセスあるいはワークフローが完了するまで待機させることができます このような場合、Step Functions では AWS SDK サービスインテグレーションと一部の最適化サービスインテグレーションにタスクトークンを渡すことができます。`SendTaskSuccess` あるいは `SendTaskFailure` 呼び出しでタスクがタスクトークンを受け取るまで、このタスクは停止します。

コールバックタスクトークンを使用する `Task` ステートがタイムアウトになると、新しいランダムトークンが生成されます。タスクトークンには、コンテキストオブジェクトからアクセスできます

> **Note**
>
> タスクトークンには少なくとも 1 文字が含まれている必要があり、1024 文字を超えることはできません。

AWS SDK `.waitForTaskToken` インテグレーションで使用するには、使用する API にタスクトークンを配置するパラメータフィールドが必要です。

> **Note**
>
> AWS タスクトークンは同じアカウント内のプリンシパルから渡す必要があります。 AWS トークンを別のアカウントのプリンシパルから送信した場合、トークンは機能しません。

> **Tip**
>
> コールバックタスクトークンサービス統合パターンを使用するサンプルワークフローをデプロイするには AWS アカウント、「モジュール 4-ワークショップのタスクトークンでコールバックを待つ」を参照してください。 AWS Step Functions

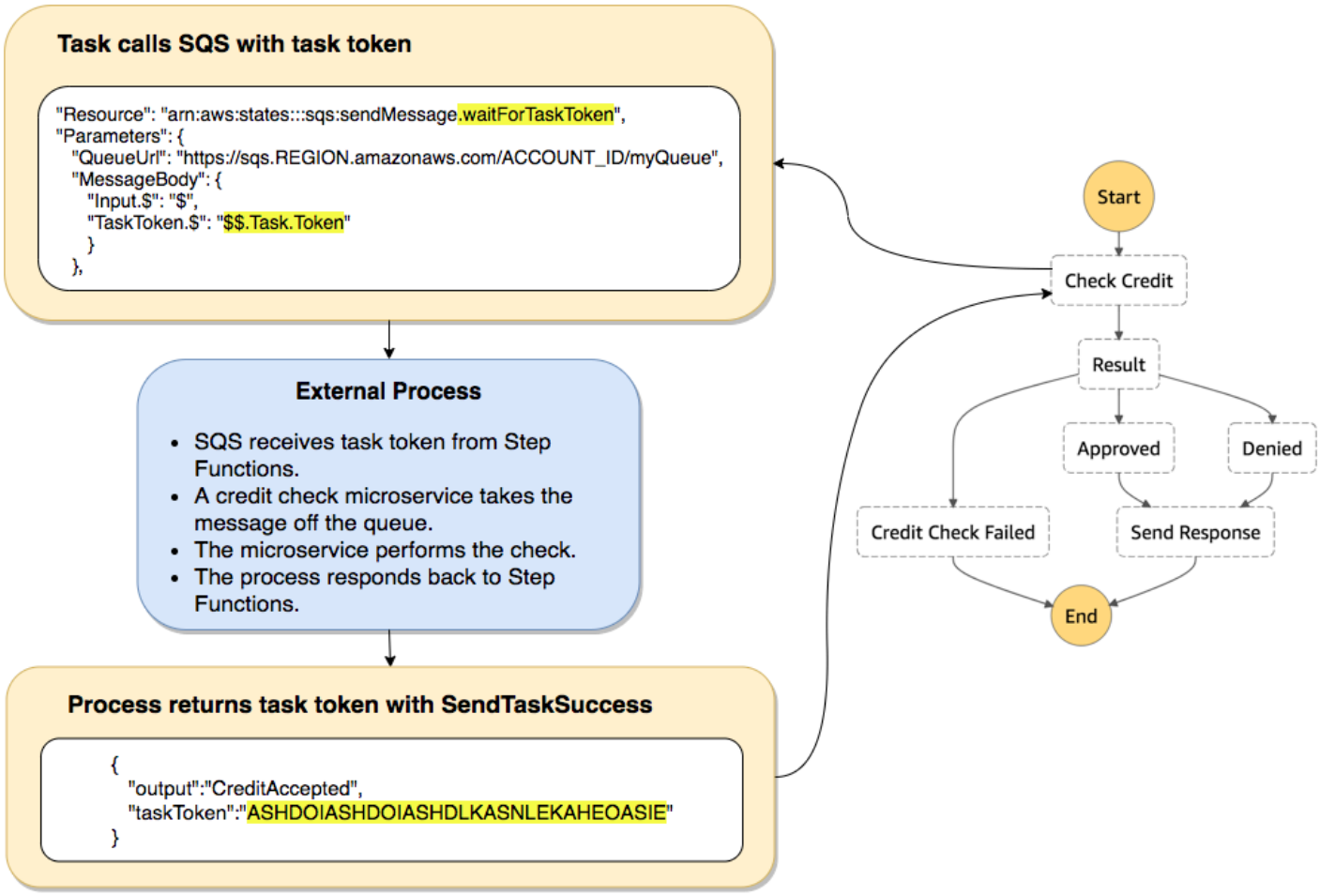
どの統合されたサービスでタスクトークンの待機 (`.waitForTaskToken`) がサポートされているかのリストについては、Step Functions 用統合最適化 を参照してください。

トピック

- タスクトークンの例
- コンテキストオブジェクトからトークンを取得する
- 待機中のタスクにハートビートタイムアウトを設定する

タスクトークンの例

この例では、承認ワークフローの一部としてクレジットチェックを実行するために、外部のマイクロサービスを Step Functions ワークフローに統合する必要があります。Step Functions は、タスクトークンを含む Amazon SQS メッセージをメッセージの一部として公開します。1 つの外部システムが Amazon SQS に統合され、キューからメッセージを引き出します。これが完了すると、結果および元のタスクトークンが返されます。それから、Step Functions は ワークフローを続けます。



Amazon SQS を参照するタスク定義の "Resource" フィールドには、末尾に追加された .waitForTaskToken が含まれています。

```
"Send message to SQS": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
  "Parameters": {
    "QueueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/myQueue",
    "MessageBody": {
        "Message": "Hello from Step Functions!",
        "TaskToken.$": "$$.Task.Token"
     }
  },
  "Next": "NEXT_STATE"
}
```

これにより、Step Functions にタスクトークンを一時停止して待機するようにという指示がでます。.waitForTaskToken を使用してリソースを指定する場合、状態定義の "Parameters" フィールドで特別なパス指定 ($$.Task.Token) を使用して、タスクトークンにアクセスすることができます。最初の $$. は、そのパスアクセスにコンテキストオブジェクトを指定し、現在実行中のタスクのタスクトークンを取得します。

完了すると、外部サービスが taskToken が含まれる SendTaskSuccess または SendTaskFailure を呼び出します。この時点でのみ、ワークフローが次の状態へと続行します。

> **Note**
>
> プロセスが SendTaskSuccess または SendTaskFailure とともにタスクトークンを送信することに失敗した場合に無期限に待機することを回避するには、待機中のタスクにハートビートタイムアウトを設定する を参照してください。

コンテキストオブジェクトからトークンを取得する

コンテキストオブジェクトは、実行に関する情報が含まれた内部の JSON オブジェクトです。状態の入力と同様に、実行中に "Parameters" フィールドからパスを使用してこれにアクセスできます。タスクの定義内からアクセスすると、これにはタスクトークンを含めた特定の実行に関する情報が含まれます。

```
{
    "Execution": {
        "Id": "arn:aws:states:us-
east-1:123456789012:execution:stateMachineName:executionName",
        "Input": {
           "key": "value"
        },
        "Name": "executionName",
        "RoleArn": "arn:aws:iam::123456789012:role...",
        "StartTime": "2019-03-26T20:14:13.192Z"
    },
    "State": {
        "EnteredTime": "2019-03-26T20:14:13.192Z",
        "Name": "Test",
        "RetryCount": 3
    },
    "StateMachine": {
        "Id": "arn:aws:states:us-east-1:123456789012:stateMachine:stateMachineName",
```

```
        "Name": "name"
    },
    "Task": {
        "Token": "h7XRiCdLtd/83p1E0dMccoxlzFhglsdkzpK9mBVKZsp7d9yrT1W"
    }
}
```

タスク定義の "Parameters" フィールド内から特別なパスを使用して、タスクトークンにアクセスできます。入力あるいはコンテキストオブジェクトにアクセスするには、.$ をパラメータ名に追加してパラメータがパスになるようにまず指定します。以下では、"Parameters" 仕様内で入力およびコンテキストオブジェクトからのノードを指定します。

```
"Parameters": {
   "Input.$": "$",
   "TaskToken.$": "$$.Task.Token"
 },
```

いずれの場合でも、.$ をパラメータ名に追加することで、Step Functions がパスを予期するように指示します。最初のケースでは、"$" は入力全体を含むパスです。後者のケースでは、$$. はパスがコンテキストオブジェクトにアクセスすることを指定し、また、$$.Task.Token は稼働している実行のコンテキストオブジェクト内のタスクトークンの値に対してパラメータを設定します。

Amazon SQS の例では、"Resource" フィールド内の .waitForTaskToken によって、タスクトークンが返されるまで待機するように指示されます。"TaskToken.$": "**$$.Task.Token**" パラメータは、このトークンを Amazon SQS メッセージの一部として渡します。

```
"Send message to SQS": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
  "Parameters": {
    "QueueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/myQueue",
    "MessageBody": {
        "Message": "Hello from Step Functions!",
        "TaskToken.$": "$$.Task.Token"
     }
  },
  "Next": "NEXT_STATE"
}
```

コンテキストオブジェクトの詳細については、このガイドの 入力および出力処理 セクションの コンテキストオブジェクト を参照してください。

待機中のタスクにハートビートタイムアウトを設定する

タスクトークンを待っているタスクは、実行が 1 年間のサービスクォータに達するまで待機します (状態のスロットリングに関連するクォータ を参照)。実行のスタックを回避するには、ステートマシン定義でハートビートタイムアウト間隔を設定できます。HeartbeatSeconds フィールドを使用して、タイムアウト間隔を指定します。

```
{
  "StartAt": "Push to SQS",
  "States": {
    "Push to SQS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
      "HeartbeatSeconds": 600,
      "Parameters": {
        "MessageBody": { "myTaskToken.$": "$$.Task.Token" },
        "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/push-based-queue"
      },
      "ResultPath": "$.SQS",
      "End": true
    }
  }
}
```

このステートマシン定義では、1 つのタスクがメッセージを Amazon SQS にプッシュし、外部プロセスが指定したタスクトークンとともにコールバックするまで待機します。"HeartbeatSeconds": 600 フィールドは、ハートビートタイムアウト間隔を 10 分に設定しています。タスクは、タスクトークンが次のいずれかの API アクションとともに返されるまで待機します。

- SendTaskSuccess
- SendTaskFailure
- SendTaskHeartbeat

待機しているタスクが 10 分以内に有効なタスクトークンを受信しない場合、タスクは States.Timeout エラー名で失敗します。

詳細については、コールバックタスクサンプルプロジェクト コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda) を参照してください。

サービス API にパラメータを渡す

サービス API に渡されるパラメータを管理するには、Parameters フィールドを Task 状態で使用します。

Parameters フィールド内では、API アクションで複数形の配列パラメータを使用する必要があります。例えば、Amazon EC2 との統合に DescribeSnapshots API アクションの [フィルター] フィールドを使用する場合、Filtersフィールドをとして定義する必要があります。複数形を使用しない場合、Step Functions は次のエラーを返します。

```
The field Filter is not supported by Step Functions.
```

静的 JSON をパラメータとして渡す

JSON オブジェクトをパラメータとしてリソースに渡すには、ステートマシン定義に直接含めます。

例えば、RetryStrategy API の SubmitJob パラメータを AWS Batch 用に設定するには、次のようにパラメータに含めます。

```
"RetryStrategy": {
  "attempts": 5
}
```

また、静的 JSON を使用して複数のパラメータを渡すこともできます。より複雑な例として、*myTopic* という名前の Amazon SNS トピックを発行するタスクの仕様の Resource フィールドおよび Parameters フィールドを以下に示します。

```
"Resource": "arn:aws:states:::sns:publish",
  "Parameters": {
     "TopicArn": "arn:aws:sns:us-east-2:123456789012:myTopic",
     "Message": "test 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"
       }
     }
```

```
  },
```

パスを使用して状態入力をパラメータとして渡す

状態の入力の一部をパラメータに渡すには、パスを使用します。パスは $ で始まる文字列です。これを使用して JSON テキスト内でコンポーネントを識別できます。Step Functions パスは JsonPath 構文を使用します。

パラメータがパスを使用するように指定するには、パラメータ名の末尾を .$ で終了します。例えば、状態の入力に message という名前のノード内にテキストが含まれている場合、そのテキストをパスを使用してパラメータとして渡すことができます。

次の状態入力を考えてみます。

```
{
  "comment": "A message in the state input",
  "input": {
    "message": "foo",
    "otherInfo": "bar"
  },
  "data": "example"
}
```

message という名前の指定されたノードの値をパラメータとして渡すには、次の構文を指定します。

```
"Parameters": {"myMessage.$": "$.input.message"},
```

次に、Step Functions は値 foo をパラメータとして渡します。

Step Functions のパラメータの使用の詳細については、以下を参照してください。

- 入力および出力処理
- InputPath、パラメータ、 ResultSelector

コンテキストオブジェクトノードをパラメータとして渡す

静的コンテンツおよび状態の入力からのノードに加えて、コンテキストオブジェクトからノードをパラメータとして渡すことができます。コンテキストオブジェクトは、ステートマシンの実行中に存在する動的な JSON データです。これにはステートマシンと現在の実行に関する情報が含まれます。

コンテキストオブジェクトにアクセスするには、状態の定義の "Parameters" フィールド内のパスを使用できます。

コンテキストオブジェクトに関する詳細と "Parameters" フィールドからデータにアクセスする方法については、次を参照してください。

- コンテキストオブジェクト
- コンテキストオブジェクトへのアクセス
- コンテキストオブジェクトからトークンを取得する

サポートされている AWS SDK インテグレーションの変更ログ

次の表は、サービスが最初に Step Functions と統合された時期と、その統合 API が最近更新された時期をまとめたものです。インテグレーションの使用方法の詳細については、を参照してください。AWS SDK サービスインテグレーション

> **Important**
>
> API アクションサポートは四半期ごとにリリースされます。新しいパラメータなど、すでにサポートされているアクションの更新は、すぐには利用できない場合があります。

サービス	初期サポート	更新	
AWS AppFabric	2024 年 1 月 18 日		
B2B Data Interchange	2024 年 1 月 18 日		
AWS Data Exports	2024 年 1 月 18 日		
Amazon Bedrock	2024 年 1 月 18 日		
Amazon Bedrock Agents	2024 年 1 月 18 日		
Amazon Bedrock Runtime Agents	2024 年 1 月 18 日		

サービス	初期サポート	更新	
Amazon Bedrock Runtime	2024 年 1 月 18 日		
Amazon CloudFront KeyValueStore	2024 年 1 月 18 日		
Amazon CodeGuru Security	2024 年 1 月 18 日		
AWS Cost Optimization Hub	2024 年 1 月 18 日		
Amazon DataZone	2024 年 1 月 18 日		
Amazon EKS Auth	2024 年 1 月 18 日		
AWS Entity Resolution	2024 年 1 月 18 日		
AWS 無料利用枠	2024 年 1 月 18 日		
Amazon Inspector Scan	2024 年 1 月 18 日		
AWS Launch Wizard	2024 年 1 月 18 日		
Amazon Managed Blockchain Query	2024 年 1 月 18 日		
AWS Elemental MediaPackage V2	2024 年 1 月 18 日		
AWS HealthImaging	2024 年 1 月 18 日		
Network Manager	2024 年 1 月 18 日		
AWS Payment Cryptography	2024 年 1 月 18 日		

サービス	初期サポート	更新	
AWS Payment Cryptography Data	2024 年 1 月 18 日		
AWS Private CA Connector for Active Directory	2024 年 1 月 18 日		
Amazon Q Business	2024 年 1 月 18 日		
Amazon Q Connect	2024 年 1 月 18 日		
AWS re:Post	2024 年 1 月 18 日		
Amazon Timestream Query	2024 年 1 月 18 日		
Amazon Timestream Write	2024 年 1 月 18 日		
Trusted Advisor	2024 年 1 月 18 日		
Verified Permissions	2024 年 1 月 18 日		
Amazon WorkSpaces Thin Client	2024 年 1 月 18 日		
AWS CloudTrail Data	2023 年 6 月 16 日		
Amazon CloudWatch Internet Monitor	2023 年 6 月 16 日		
Amazon Interacti ve Video Service RealTime	2023 年 6 月 16 日		
AWS IoT TwinMaker	2023 年 6 月 16 日		

サービス	初期サポート	更新	
Amazon OpenSearch Ingestion	2023 年 6 月 16 日		
AWS Telco Network Builder	2023 年 6 月 16 日		
Amazon VPC Lattice	2023 年 6 月 16 日		
AWS Backup Storage	2023 年 2 月 17 日		
Amazon Chime Media Pipelines	2023 年 2 月 17 日		
Amazon Chime Voice	2023 年 2 月 17 日		
AWS Clean Rooms	2023 年 2 月 17 日	2024 年 1 月 18 日	
Amazon CodeCatalyst	2023 年 2 月 17 日		
Amazon Connect Cases	2023 年 2 月 17 日		
AWS Control Tower	2023 年 2 月 17 日		
Amazon DocumentDB Elastic Clusters	2023 年 2 月 17 日		
Amazon EMR Serverless	2023 年 2 月 17 日		
Amazon IVS Chat	2023 年 2 月 17 日		
Amazon Kendra Intelligent Ranking	2023 年 2 月 17 日		
AWS HealthOmics	2023 年 2 月 17 日		
Amazon Redshift Serverless	2023 年 2 月 17 日		

サービス	初期サポート	更新	
Amazon Security Lake	2023 年 2 月 17 日		
AWS Health	2023 年 2 月 17 日		
AWS IoT FleetWise	2023 年 2 月 17 日		
AWS IoT RoboRunner	2023 年 2 月 17 日		
AWS Mainframe Modernization	2023 年 2 月 17 日		
AWS Migration Hub Orchestrator	2023 年 2 月 17 日		
AWS Private 5G	2023 年 2 月 17 日		
AWS Resource Explorer	2023 年 2 月 17 日		
AWS SimSpace Weaver	2023 年 2 月 17 日		
AWS Support App	2023 年 2 月 17 日		
CloudWatch Observability Access Manager	2023 年 2 月 17 日		
EventBridge Pipes	2023 年 2 月 17 日		
EventBridge Scheduler	2023 年 2 月 17 日		
IAM Roles Anywhere	2023 年 2 月 17 日		
Kinesis Video WebRTC Storage	2023 年 2 月 17 日		

サービス	初期サポート	更新	
License Manager Linux Subscriptions	2023 年 2 月 17 日		
License Manager User Subscriptions	2023 年 2 月 17 日		
OpenSearch Serverless	2023 年 2 月 17 日		
Route 53 ARC Zonal Shift	2023 年 2 月 17 日		
SageMaker Geospatial	2023 年 2 月 17 日		
SageMaker Metrics	2023 年 2 月 17 日		
Systems Manager for SAP	2023 年 2 月 17 日		
AWS Account Management	2022 年 4 月 19 日		
AWS Amplify	2021 年 9 月 30 日		
AWS App Mesh	2021 年 9 月 30 日		
AWS App Runner	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS AppConfig	2021 年 9 月 30 日		
AWS AppConfig Data	2022 年 4 月 19 日		
AWS AppSync	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Application Discovery Service	2021 年 9 月 30 日		

サービス	初期サポート	更新	
AWS Application Migration Service	2021 年 9 月 30 日		
AWS Audit Manager	2021 年 9 月 30 日		
AWS Auto Scaling Plans	2021 年 9 月 30 日		
AWS Backup	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Backup gateway	2022 年 4 月 19 日	2023 年 2 月 17 日	
AWS Batch	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Billing Conductor	2022 年 7 月 26 日	2023 年 2 月 17 日	
AWS Budgets	2021 年 9 月 30 日		
AWS Certificate Manager	2021 年 9 月 30 日		
AWS Private Certificate Authority	2021 年 9 月 30 日		
AWS Cloud Map	2021 年 9 月 30 日		
AWS Cloud9	2021 年 9 月 30 日		
AWS CloudFormation	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS CloudHSM	2021 年 9 月 30 日		
AWS CloudHSM	2021 年 9 月 30 日		
AWS CloudTrail	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Cloud Control	2022 年 4 月 19 日		

サービス	初期サポート	更新	
AWS CodeBuild	2021 年 9 月 30 日		
AWS CodeCommit	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS CodeDeploy	2021 年 9 月 30 日		
AWS CodePipeline	2021 年 9 月 30 日		
AWS CodeStar	2021 年 9 月 30 日		
AWS CodeStar	2021 年 9 月 30 日		
AWS CodeStar	2021 年 9 月 30 日		
AWS Compute Optimizer	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Config	2021 年 9 月 30 日	2022 年 7 月 26 日	
AWS Cost Explorer Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Cost and Usage Report	2021 年 9 月 30 日		
AWS Data Exchange	2021 年 9 月 30 日	2022 年 7 月 26 日	
AWS Data Pipeline	2021 年 9 月 30 日		
AWS DataSync	2021 年 9 月 30 日	2022 年 7 月 26 日	
AWS Database Migration Service	2021 年 9 月 30 日		
AWS Device Farm	2021 年 9 月 30 日		
AWS Direct Connect	2021 年 9 月 30 日		

サービス	初期サポート	更新	
AWS Directory Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS EC2 Instance Connect	2021 年 9 月 30 日		
AWS Elastic Beanstalk	2021 年 9 月 30 日		
AWS Elemental MediaLive	2021 年 9 月 30 日		
AWS Elemental MediaPackage	2021 年 9 月 30 日		
AWS Elemental MediaPackage VOD	2021 年 9 月 30 日		
AWS Elemental MediaStore	2021 年 9 月 30 日		
AWS Fault Injection Service	2021 年 9 月 30 日		
AWS Firewall Manager	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Glue	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Glue DataBrew	2021 年 9 月 30 日		
AWS IoT Greengrass	2021 年 9 月 30 日		
AWS Ground Station	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Identity and Access Management	2021 年 9 月 30 日		

サービス	初期サポート	更新	
AWS IoT	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS IoT 1-Click	2021 年 9 月 30 日		
AWS IoT Analytics	2021 年 9 月 30 日		
AWS IoT Core Device Advisor	2021 年 9 月 30 日		
AWS IoT Events	2021 年 9 月 30 日		
AWS IoT Events Data	2021 年 9 月 30 日		
AWS IoT Fleet Hub	2021 年 9 月 30 日		
AWS IoT Greengrass Version 2	2021 年 9 月 30 日		
AWS IoT Jobs Data Plane	2021 年 9 月 30 日		
AWS IoT Secure Tunneling	2021 年 9 月 30 日		
AWS IoT SiteWise	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS IoT Things Graph	2021 年 9 月 30 日		
AWS IoT Wireless	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Key Management Service	2021 年 9 月 30 日	2022 年 7 月 26 日	
AWS Lake Formation	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Lambda	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
AWS License Manager	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Marketplace	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Marketplace Commerce Analytics	2021 年 9 月 30 日		
AWS Marketplace Entitlement Service	2021 年 9 月 30 日		
AWS Elemental MediaTailor	2021 年 9 月 30 日	2022 年 7 月 26 日	
AWS Migration Hub	2021 年 9 月 30 日		
AWS Migration Hub Config	2021 年 9 月 30 日		
AWS Migration Hub Strategy Recommendations	2022 年 4 月 19 日	2023 年 2 月 17 日	
AWS Mobile	2021 年 9 月 30 日		
AWS Network Firewall	2021 年 9 月 30 日		
AWS OpsWorks	2021 年 9 月 30 日		
AWS OpsWorks CM	2021 年 9 月 30 日		
AWS Organizations	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Outposts	2021 年 9 月 30 日		
AWS Panorama	2022 年 4 月 19 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon Relational Database Service Performance Insights	2021 年 9 月 30 日		
AWS の料金表	2021 年 9 月 30 日		
Amazon Relational Database Service	2021 年 9 月 30 日		
AWS Resilience Hub	2022 年 4 月 19 日		
AWS Resource Access Manager	2021 年 9 月 30 日		
AWS Resource Groups	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Resource Groups Tagging API	2021 年 9 月 30 日		
AWS RoboMaker	2021 年 9 月 30 日		
AWS IAM Identity Center	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS SSO OIDC	2021 年 9 月 30 日		
AWS Secrets Manager	2021 年 9 月 30 日		
AWS Security Token Service	2021 年 9 月 30 日		
AWS Security Hub	2021 年 9 月 30 日		
AWS Server Migration Service	2021 年 9 月 30 日		

サービス	初期サポート	更新	
AWS Service Catalog	2021 年 9 月 30 日		
AWS Service Catalog AppRegistry	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Shield	2021 年 9 月 30 日		
AWS Signer	2021 年 9 月 30 日		
AWS IAM Identity Center	2021 年 9 月 30 日		
AWS IAM Identity Center Admin	2021 年 9 月 30 日		
AWS Step Functions	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Storage Gateway	2021 年 9 月 30 日		
AWS Support	2021 年 9 月 30 日		
AWS Transfer Family	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS WAF	2021 年 9 月 30 日		
AWS WAF Regional	2021 年 9 月 30 日		
AWS WAFV2	2021 年 9 月 30 日		
AWS Well-Architected Tool	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS X-Ray	2021 年 9 月 30 日	2023 年 2 月 17 日	
AWS Marketplace Metering Service	2021 年 9 月 30 日		

サービス	初期サポート	更新	
AWS Serverless Application Repository	2021 年 9 月 30 日		
AWS Identity and Access Management Access Analyzer	2021 年 9 月 30 日		
Alexa for Business	2021 年 9 月 30 日		
Amazon API Gateway	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon API Gateway	2021 年 9 月 30 日		
Amazon AppIntegrations	2021 年 9 月 30 日		
Amazon AppStream 2.0	2021 年 9 月 30 日		
Amazon AppFlow	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Athena	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Augmented AI	2021 年 9 月 30 日		
Amazon Braket	2021 年 9 月 30 日		
Amazon Chime	2021 年 9 月 30 日		
Amazon Chime Meetings	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amazon Cloud Directory	2021 年 9 月 30 日		
Amazon CloudFront	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon CloudSearch	2021 年 9 月 30 日		

サービス	初期サポート	更新
Amazon CloudWatch	2021 年 9 月 30 日	2023 年 2 月 17 日
Amazon CloudWatch Application Insights	2021 年 9 月 30 日	
CloudWatch Evidently	2022 年 4 月 19 日	
Amazon CloudWatch Logs	2021 年 9 月 30 日	
Amazon CloudWatch RUM	2022 年 4 月 19 日	2023 年 2 月 17 日
Amazon CloudWatch Synthetics	2021 年 9 月 30 日	
Amazon CodeGuru Profiler	2021 年 9 月 30 日	
Amazon CodeGuru Reviewer	2021 年 9 月 30 日	
Amazon Cognito	2021 年 9 月 30 日	
Amazon Cognito Identity Provider	2021 年 9 月 30 日	
Amazon Cognito Sync	2021 年 9 月 30 日	
Amazon Comprehend	2021 年 9 月 30 日	2023 年 2 月 17 日
Amazon Comprehend Medical	2021 年 9 月 30 日	
Amazon Connect Contact Lens	2021 年 9 月 30 日	

サービス	初期サポート	更新	
Amazon Connect Participant Service	2021 年 9 月 30 日		
Amazon Connect	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Connect Voice ID	2022 年 4 月 19 日		
Amazon Connect Wisdom	2022 年 4 月 19 日		
Amazon Data Lifecycle Manager	2021 年 9 月 30 日		
Amazon Detective	2021 年 9 月 30 日		
Amazon DevOps Guru	2021 年 9 月 30 日	2022 年 7 月 26 日	
Amazon DocumentDB (with MongoDB compatibility)	2021 年 9 月 30 日		
Amazon DynamoDB	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon DynamoDB Streams	2021 年 9 月 30 日		
Amazon EC2 Container Registry	2021 年 9 月 30 日		
Amazon EC2 Container Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon EC2 Systems Manager	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon EMR	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon ElastiCache	2021 年 9 月 30 日		
Amazon Elastic Inference	2021 年 9 月 30 日		
Amazon Elastic Block Store	2021 年 9 月 30 日		
Amazon Elastic Compute Cloud	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Elastic Container Registry Public	2021 年 9 月 30 日		
Amazon Elastic File System	2021 年 9 月 30 日		
Amazon Elastic Kubernetes Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon EMR	2021 年 9 月 30 日		
Amazon Elastic Transcoder	2021 年 9 月 30 日		
Amazon OpenSearch Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon OpenSearch Service	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amazon EventBridge	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon FSx	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Forecast Query	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon Forecast Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Fraud Detector	2021 年 9 月 30 日		
Amazon GameLift	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon GameSparks	2022 年 7 月 26 日		
Amazon S3 Glacier	2021 年 9 月 30 日		
Amazon GuardDuty	2021 年 9 月 30 日		
AWS HealthLake	2021 年 9 月 30 日		
Amazon Honeycode	2021 年 9 月 30 日		
Amazon Inspector	2021 年 9 月 30 日		
Amazon Inspector V2	2022 年 4 月 19 日		
Amazon Interactive Video Service	2021 年 9 月 30 日		
Amazon Kendra	2021 年 9 月 30 日		
Amazon Kinesis	2021 年 9 月 30 日		
Amazon Kinesis Analytics	2021 年 9 月 30 日		
Amazon Kinesis Analytics V2	2021 年 9 月 30 日		
Amazon Kinesis Firehose	2021 年 9 月 30 日		

サービス	初期サポート	更新	
Amazon Kinesis Video Signaling Channels	2021 年 9 月 30 日		
Amazon Kinesis Video Streams	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Kinesis Video Streams Archived Media	2021 年 9 月 30 日		
Amazon Kinesis video stream	2021 年 9 月 30 日		
Amazon Lex Model Building Service	2021 年 9 月 30 日		
Amazon Lex Model Building Service V2	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Lex	2021 年 9 月 30 日		
Amazon Lex Runtime V2	2021 年 9 月 30 日		
Amazon Lightsail	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Location Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Lookout for Equipment	2021 年 9 月 30 日		
Amazon Lookout for Metrics	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Lookout for Vision	2021 年 9 月 30 日		

サービス	初期サポート	更新	
Amazon MQ	2021 年 9 月 30 日		
Amazon Macie	2021 年 9 月 30 日		
Amazon Macie 2	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Managed Blockchain	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Managed Grafana	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amazon Managed Service for Prometheus	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Managed Streaming for Apache Kafka	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Managed Streaming for Apache Kafka x	2022 年 4 月 19 日		
Amazon Managed Workflows for Apache Airflow	2021 年 9 月 30 日		
Amazon Mechanical Turk	2021 年 9 月 30 日		
Amazon MemoryDB for Redis	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amazon Nimble Studio	2021 年 9 月 30 日		
Amazon Personalize	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon Personalize Events	2021 年 9 月 30 日		
Amazon Personalize Runtime	2021 年 9 月 30 日		
Amazon Pinpoint	2021 年 9 月 30 日		
Amazon Pinpoint Email Service	2021 年 9 月 30 日		
Amazon Pinpoint SMS and Voice Service	2021 年 9 月 30 日		
Amazon Pinpoint SMS and Voice V2 Service	2022 年 7 月 26 日		
Amazon Polly	2021 年 9 月 30 日		
Amazon QLDB	2021 年 9 月 30 日		
Amazon QLDB Session	2021 年 9 月 30 日		
Amazon QuickSight	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Redshift	2021 年 9 月 30 日		
Amazon Redshift Data API	2021 年 9 月 30 日		
Amazon Rekognition	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Relational Database Service	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon Route 53	2021 年 9 月 30 日		
Amazon Route 53 Recovery Control Config	2021 年 9 月 30 日	2022 年 7 月 26 日	
Amazon Route 53 Domains	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Route 53 Resolver	2021 年 9 月 30 日		
Amazon S3 on Outposts	2021 年 9 月 30 日	2022 年 7 月 26 日	
Amazon SageMaker Runtime Feature Store Runtime	2021 年 9 月 30 日		
Amazon SageMaker Runtime Runtime	2021 年 9 月 30 日		
Amazon SageMaker	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon SageMaker Edge Manager	2021 年 9 月 30 日		
Amazon Simple Email Service	2021 年 9 月 30 日		
Amazon Simple Email Service V2	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Simple Notification Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Simple Queue Service	2021 年 9 月 30 日	2023 年 2 月 17 日	

サービス	初期サポート	更新	
Amazon Simple Storage Service	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Simple Workflow Service	2021 年 9 月 30 日		
Amazon Textract	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon Transcribe	2021 年 9 月 30 日		
Amazon Translate	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon WorkDocs	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon WorkMail	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon WorkMail Message Flow	2021 年 9 月 30 日		
Amazon WorkSpaces	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon WorkSpaces Web	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amplify	2021 年 9 月 30 日		
Amplify UI Builder	2022 年 4 月 19 日	2023 年 2 月 17 日	
Application Auto Scaling	2021 年 9 月 30 日		
Amazon EC2 Auto Scaling	2021 年 9 月 30 日	2023 年 2 月 17 日	
CodeArtifact	2021 年 9 月 30 日		
DynamoDB Accelerator	2021 年 9 月 30 日		

サービス	初期サポート	更新	
EC2 Image Builder	2021 年 9 月 30 日		
AWS Elastic Disaster Recovery	2022 年 4 月 19 日	2023 年 2 月 17 日	
Elastic Load Balancing	2021 年 9 月 30 日		
Elastic Load Balancing V2	2021 年 9 月 30 日		
MediaConnect	2021 年 9 月 30 日		
Amazon S3 Control	2021 年 9 月 30 日	2023 年 2 月 17 日	
Recycle Bin for Amazon EBS	2022 年 4 月 19 日	2023 年 2 月 17 日	
Savings Plans	2021 年 9 月 30 日		
Amazon EventBridge Schema Registry	2021 年 9 月 30 日		
Service Quotas	2021 年 9 月 30 日		
AWS Snowball	2021 年 9 月 30 日		

Step Functions サンプルプロジェクト

AWS Step Functions コンソールでは、以下のスターターテンプレートのいずれかを選択して、ステートマシンを AWS アカウント にデプロイできます。これらのスターターテンプレートは、ワークフローのプロトタイプと定義、およびプロジェクトに関連するすべての AWS リソースを自動的に作成する、すぐに実行できるサンプルプロジェクトです。

これらのサンプルプロジェクトを使用してそのままデプロイして実行することも、ワークフロープロトタイプを使用してその上に構築することもできます。これらのプロジェクトに基づいて構築する場合、Step Functions はワークフロープロトタイプを作成しますが、ワークフロー定義にリストされているリソースはデプロイしません。

サンプルプロジェクトをデプロイすると、完全に機能するステートマシンがプロビジョニングされ、ステートマシンを実行するための関連リソースが作成されます。サンプルプロジェクトを作成すると、Step Functions では AWS CloudFormation を使用して、ステートマシンが参照する関連リソースが作成されます。

トピック

- バッチジョブの管理 (AWS Batch、Amazon SNS)
- コンテナタスクの管理 (Amazon ECS、Amazon SNS)
- データレコードの転送 (Lambda、DynamoDB、Amazon SQS)
- Job ステータスの投票 (Lambda、) AWS Batch
- タスクタイマー (Lambda、Amazon SNS)
- コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda)
- Amazon EMR ジョブを管理する
- EMR Serverless ジョブを実行する
- ワークフロー内でワークフローをスタートする (Step Functions、Lambda)
- マップ状態を使用してデータを動的に処理する
- 分散マップで CSV ファイルを処理する
- Amazon S3 バケットのデータを分散マップで処理する
- 機械学習モデルのトレーニング
- 機械学習モデルのチューニング
- Amazon SQS からの大容量メッセージの処理 (Express ワークフロー)
- 選択的チェックポイントの例 (Express ワークフロー)

- AWS CodeBuild プロジェクトを構築する (CodeBuild、Amazon SNS)
- データを前処理し、機械学習モデルをトレーニングする
- Lambda オーケストレーションの例
- Athena クエリをスタートする
- 複数のクエリを実行します (Amazon Athena、Amazon SNS)
- 大規模なデータセット (Amazon Athena、Amazon S3 AWS Glue、Amazon SNS) へのクエリ
- データを最新の状態に保つ (Amazon Athena、Amazon S3、 AWS Glue)
- Amazon EKS クラスターの管理
- API Gateway を呼び出す
- API Gateway 統合を使用して Fargate で実行されているマイクロサービスを呼び出す
- カスタムイベントの送信先 EventBridge
- 同期 Express ワークフローを呼び出す
- Amazon Redshift (Lambda、Amazon Redshift データ API) を使用して ETL/ELT ワークフローを実行する
- エラー処理で Step Functions と AWS Batch を使用する
- AWS Batch ジョブのファンアウト
- AWS Batch Lambda 付き
- Amazon Bedrock で AI プロンプトチェーンを実行する

バッチジョブの管理 (AWS Batch、Amazon SNS)

このサンプルプロジェクトでは、AWS Batch ジョブを送信後、ジョブの結果 (成功または失敗) に基づき Amazon SNS 通知を送信する方法について説明します。このサンプルプロジェクトをデプロイすると、AWS Step Functions ステートマシン、AWS Batch ジョブ、および Amazon SNS トピックが作成されます。

このプロジェクトでは、Step Functions は、ステートマシンを使用して、AWS Batch ジョブを同期的に呼び出します。その後、ジョブが成功または失敗するまで待機し、ジョブの成功または失敗に関するメッセージを含む Amazon SNS トピックを送信します。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

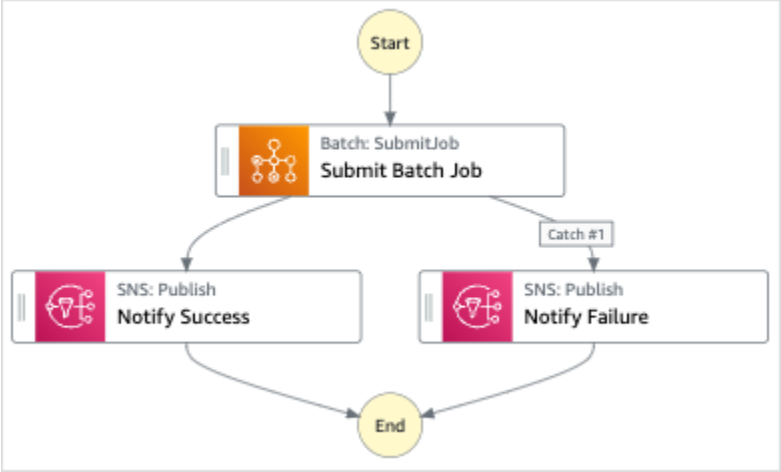
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

2. 検索ボックスに **`Manage a batch job`** と入力し、返された検索結果から [バッチジョブの管理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS Batch ジョブ
   - 1 つの Amazon SNS トピック
   - AWS Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[バッチジョブの管理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをリソースに直接渡すことにより、Amazon SNS AWS Batch と統合します。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し、サービス API `Parameters` に渡すことによって Amazon SNS AWS Batch を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "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 の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "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"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

コンテナタスクの管理 (Amazon ECS、Amazon SNS)

このサンプルプロジェクトでは、AWS Fargate タスクを実行後、ジョブの結果 (成功または失敗) に基づき Amazon SNS 通知を送信する方法について説明します。このサンプルプロジェクトをデプロイすると、AWS Step Functions ステートマシン、Fargate クラスター、および Amazon SNS トピックが作成されます。

このプロジェクトでは、Step Functions は、ステートマシンを使用して、Fargate タスクを同期的に呼び出します。その後、タスクが成功または失敗するまで待機し、ジョブの成功または失敗に関するメッセージを含む Amazon SNS トピックを送信します。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

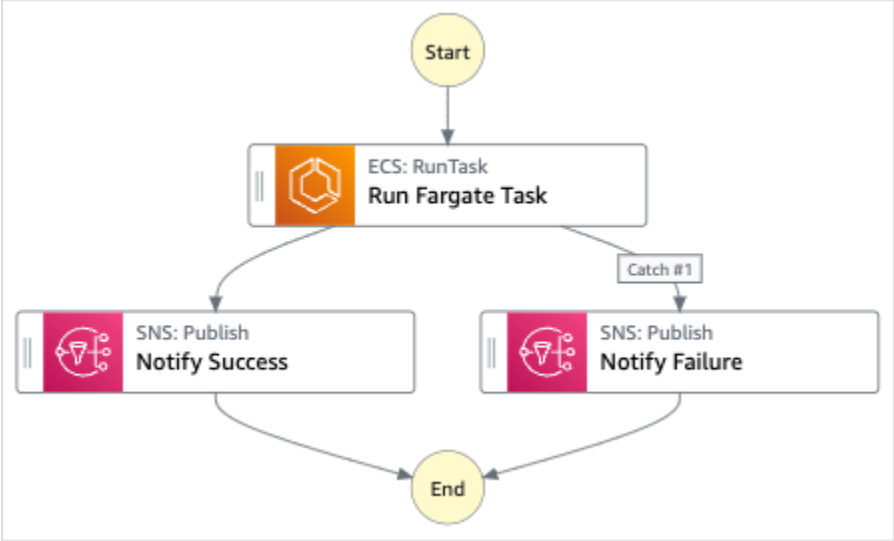
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Manage a container task`** と入力し、返された検索結果から [コンテナタスクの管理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築

するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

このサンプルプロジェクトは、以下のリソースをデプロイします。

- AWS Fargate クラスタ。
- 1 つの Amazon SNS トピック
- AWS Step Functions ステートマシン
- 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[コンテナタスクの管理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをリソースに直接渡すことにより、Amazon SNS AWS Fargate と統合します。この例のステートマシンを参照して、Step Functions がどのようにステートマシンを使用して Fargate タスクを同期的に呼び出し、タスクの成功または失敗を待機して、ジョブが成功したか失敗したかどうかに関するメッセージを Amazon SNS トピックに送信するかご覧ください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "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 の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。ベストプラクティスとして、IAM ポリシーで必要な許可のみ付与することをお勧めします。

```
{
    "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"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

データレコードの転送 (Lambda、DynamoDB、Amazon SQS)

このサンプルプロジェクトでは、Step Functions ステートマシンを使用して Amazon DynamoDB テーブルから項目を繰り返し読み取り、その項目を Amazon SQS キューに送信する方法を示しています。このサンプルプロジェクトをデプロイすると、Step Functions ステートマシン、DynamoDB テーブル、AWS Lambda 関数、および Amazon SQS キューが作成されます。

このプロジェクトでは、Step Functions は Lambda 関数を使用して DynamoDB テーブルにデータを入力します。ステートマシンはまた、`for` ループを使用して各エントリを読み込み、Amazon SQS キューに送信します。

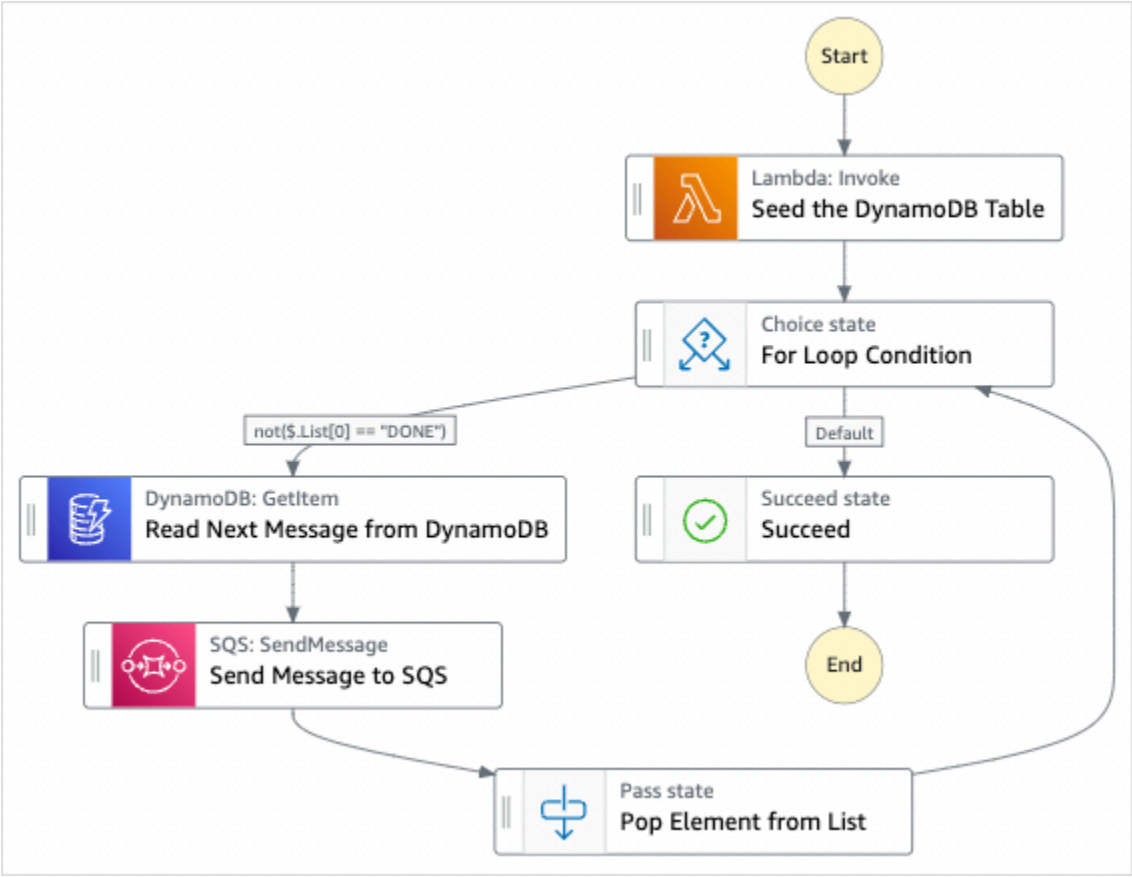
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Transfer data records`** と入力し、返された検索結果から [データレコードの転送] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - DynamoDB テーブルをシードするための Lambda 関数
   - Amazon SQS キュー
   - DynamoDB テーブル。
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[データレコードの転送] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルオブジェクトのステートマシンは、パラメータを直接あれらのリソースに渡すことで、DynamoDB および Amazon SQS と統合します。

このステートマシンの例を参照して、`Resource` フィールドの Amazon リソースネーム (ARN) に接続し、`Parameters` をサービス API に渡すことで、Step Functions が DynamoDB と Amazon SQS を制御する方法を確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "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"
    }
  }
}
```

パラメータを渡し、結果を管理する方法については、以下を参照してください。

- サービス API にパラメータを渡す
- ResultPath

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。ベストプラクティスとして、IAM ポリシーで必要な許可のみ付与することをお勧めします。

```
{
    "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"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Job ステータスの投票 (Lambda、) AWS Batch

AWS Batch このサンプルプロジェクトはジョブポーラーを作成します。 AWS Step Functions AWS Lambda Waitジョブをチェックするステートループを作成するために使用するステートマシンを実装しています。 AWS Batch

このサンプルプロジェクトは、Step Functions AWS Batch ワークフローがジョブを送信し、そのジョブが完了するのを待ってから正常に終了するようにすべてのリソースを作成および構成します。

> **Note**
>
> また、Lambda 関数を使用せずに、このパターンを実装することもできます。 AWS Batch 直接制御する方法については、を参照してくださいAWS Step Functions 他のサービスとの併用。

このサンプルプロジェクトは、ステートマシン、2 つの Lambda 関数、および 1 AWS Batch つのキューを作成し、関連する IAM 権限を設定します。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

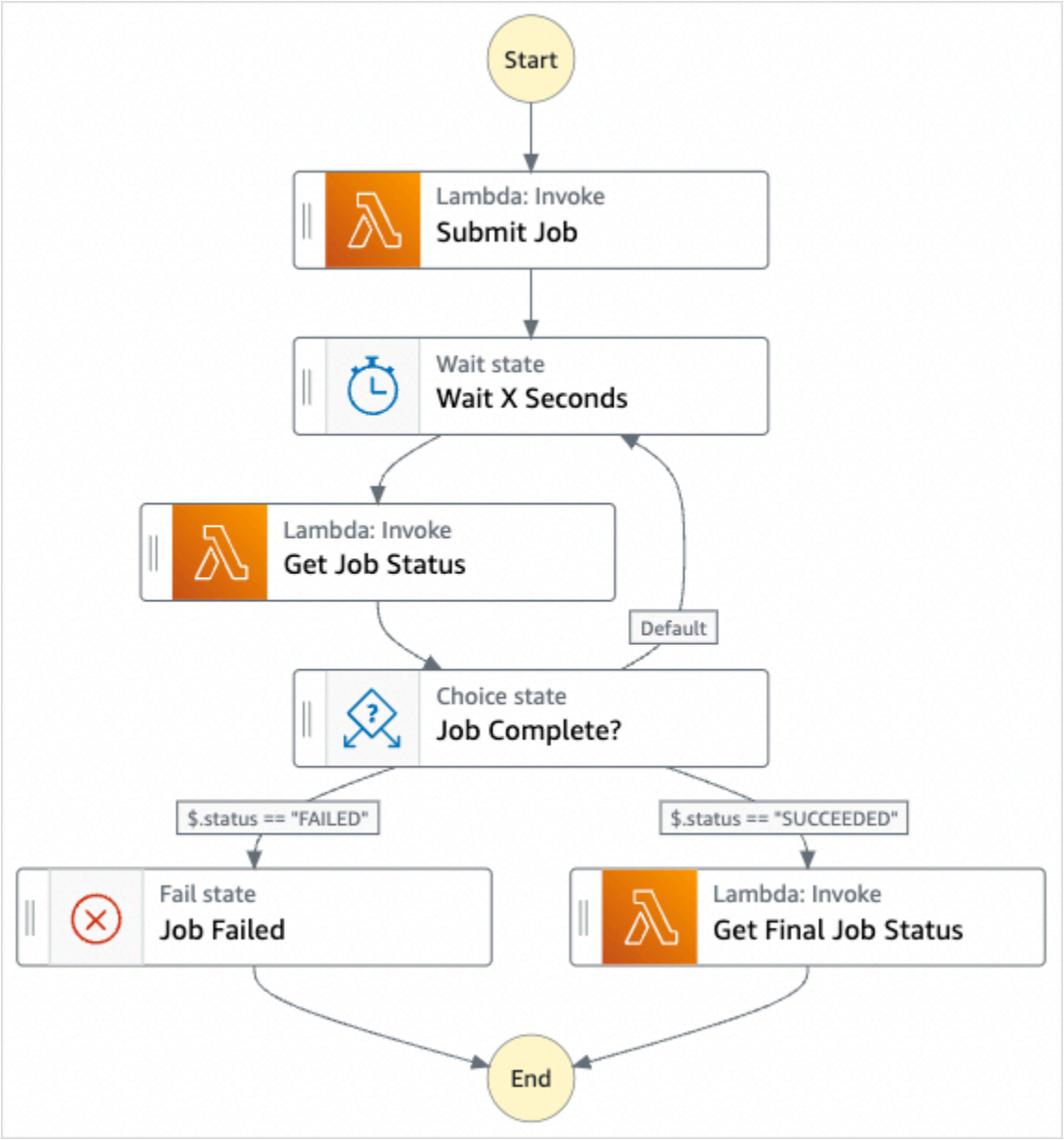
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Job Poller`** と入力し、返された検索結果から [ジョブポーリング] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - ジョブの送信、 AWS Batch AWS Batch 送信されたジョブの現在のステータス、および最終的なジョブの完了ステータスの取得を行う 3 つの Lambda 関数。
   - ジョブ AWS Batch
   - AWS Step Functions 任意のステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[ジョブポーリング] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してく

ださい。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

すべてのリソースのプロビジョニングとデプロイが完了すると、次のような入力例を含む [実行を開始] ダイアログボックスが表示されます。

```
{
    "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
  }
```

> **Note**
>
> `wait_time` は、`Wait` 状態が 60 秒ごとにループするように命令します。

- [実行を開始] ダイアログボックスで、以下の操作を行います。

  1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

     > **Note**
     >
     > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

  2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

     [デモの実行] を選択した場合、実行入力を入力する必要はありません。

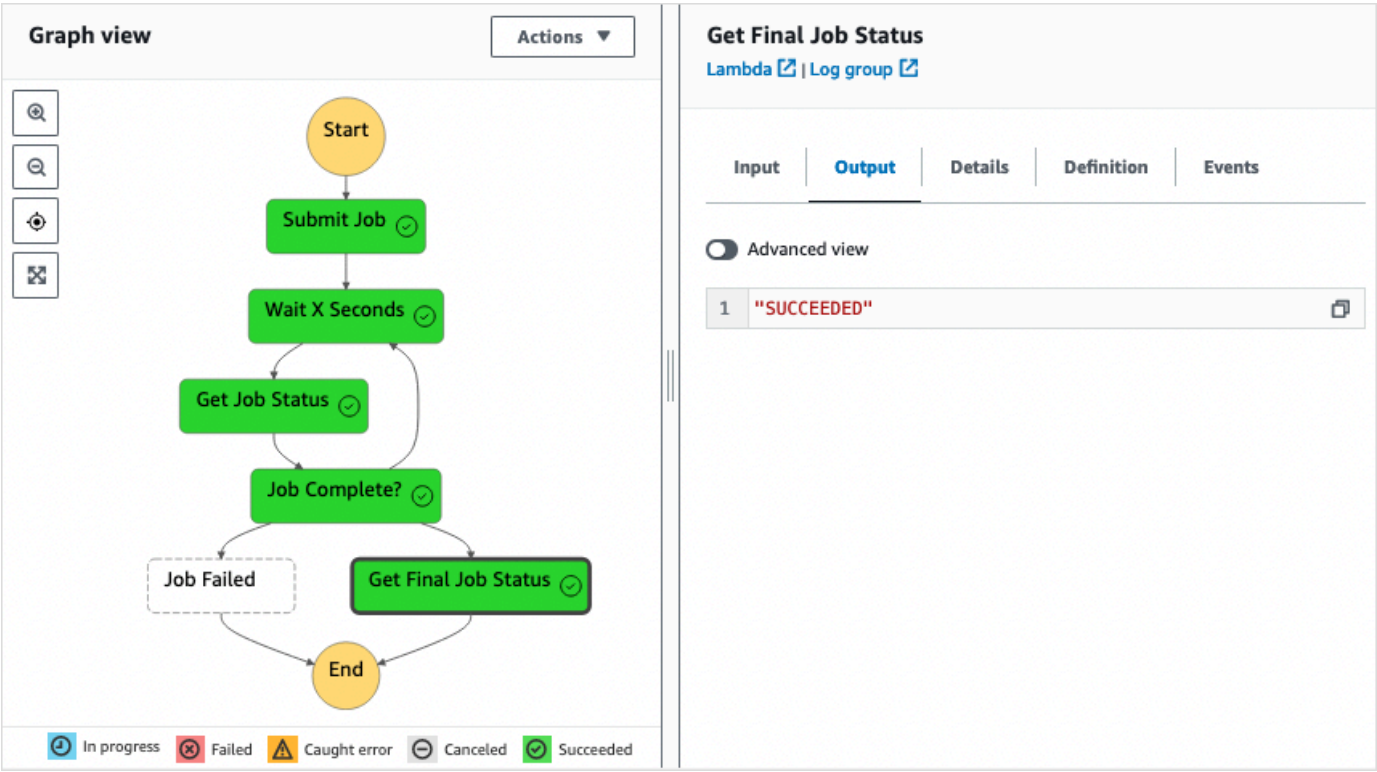
     > **Note**
     >
     > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

  3. [実行のスタート] を選択します。
  4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

たとえば、 AWS Batch ジョブのステータスの変化や実行のループ結果を確認するには、「出力」タブを選択します。

以下のイメージは、[グラフ表示] 内の実行ステータスグラフを示しています。また、[出力] タブには、選択したステップの実行出力も表示されます。



ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、 AWS Lambda と統合されてジョブを送信します。 AWS Batch このステートマシンの例を参照して、Step Functions が Lambda とをどのように制御するかを確認してください。 AWS Batch

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "Comment": "An example of the Amazon States Language that runs an AWS Batch job and
 monitors the job until it completes.",
    "StartAt": "Submit Job",
    "States": {
      "Submit Job": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPol-SubmitJobFunction-
jDaYcl4cx55r",
        "ResultPath": "$.guid",
        "Next": "Wait X Seconds"
      },
      "Wait X Seconds": {
        "Type": "Wait",
        "SecondsPath": "$.wait_time",
        "Next": "Get Job Status"
      },
      "Get Job Status": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPoll-
CheckJobFunction-1JkJwY10vonI",
        "Next": "Job Complete?",
        "InputPath": "$.guid",
        "ResultPath": "$.status"
      },
      "Job Complete?": {
        "Type": "Choice",
        "Choices": [
          {
            "Variable": "$.status",
            "StringEquals": "FAILED",
            "Next": "Job Failed"
          },
          {
            "Variable": "$.status",
            "StringEquals": "SUCCEEDED",
            "Next": "Get Final Job Status"
          }
        ],
        "Default": "Wait X Seconds"
      },
```

```
      "Job Failed": {
        "Type": "Fail",
        "Cause": "AWS Batch Job Failed",
        "Error": "DescribeJob returned FAILED"
      },
      "Get Final Job Status": {
        "Type": "Task",
        "Resource": "arn:aws::lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPoll-
CheckJobFunction-1JkJwY10vonI",
        "InputPath": "$.guid",
        "End": true
      }
    }
  }
```

タスクタイマー (Lambda、Amazon SNS)

このサンプルプロジェクトは、タスクタイマーを作成します。 AWS Step Functions ステートを実装するステートマシンを実装し、Amazon Simple Notification Service (Amazon SNS) AWS Lambda 通知を送信する関数を使用します。`Wait`待機 状態は、トリガーが 1 単位の作業を実行するのを待機する状態タイプです。

> **Note**
>
> このサンプルプロジェクトは、Amazon Simple Notification Service (Amazon SNS) AWS Lambda 通知を送信する関数を実装しています。また、Amazon SNS 通知を Amazon ステートメント言語から直接送信することもできます。AWS Step Functions 他のサービスとの併用を参照してください。

このサンプルプロジェクトでは、ステートマシン、Lambda 関数、Amazon SNS トピックを作成し、関連する AWS Identity and Access Management (IAM) 権限を設定します。[Task Timer] (タスクタイマー) サンプルプロジェクトで作成されるリソースの詳細については、以下を参照してください。

他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS AWS Step Functions 他のサービスとの併用

- AWS CloudFormation ユーザーガイド

- Amazon Simple Notification Service デベロッパーガイド
- AWS Lambda デベロッパーガイド
- IAM 開始方法ガイド

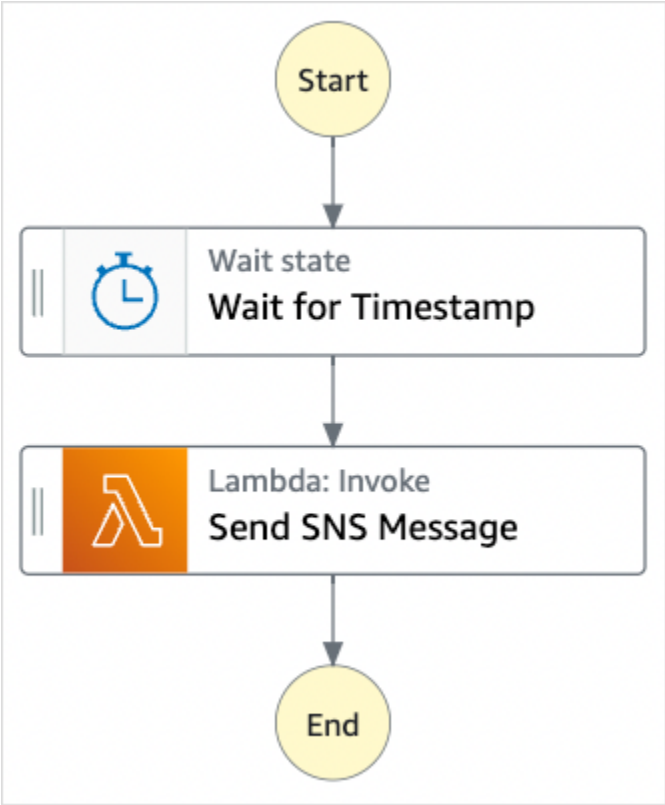
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Task Timer`** と入力し、返された検索結果から [タスクタイマー] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon SNS 通知を送信する Lambda 関数。
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[タスクタイマー] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

すべてのリソースのプロビジョニングとデプロイが完了すると、次のような入力例を含む [実行を開始] ダイアログボックスが表示されます。

```
{
    "jobName": "my-job",{
  "topic": "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-TaskTimercc68840e-
c3d3-42a8-911e-821b7ce248e5-SNSTopic-44UjcFxzhACT",
  "message": "HelloWorld",
  "timer_seconds": 10
}
```

- [実行を開始] ダイアログボックスで、以下の操作を行います。

  1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

     > **Note**
     >
     > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

  2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

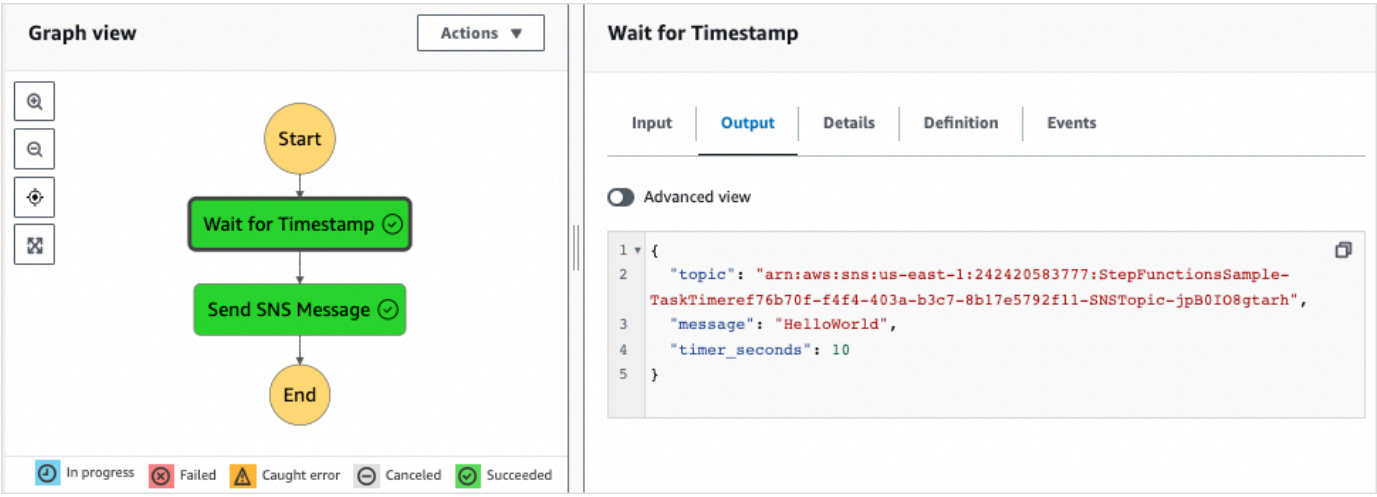
     [デモの実行] を選択した場合、実行入力を入力する必要はありません。

     > **Note**
     >
     > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

  3. [実行のスタート] を選択します。
  4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

     実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

例えば、次のイメージは、選択したステップ [タイムスタンプを待つ] の出力を示しています。このステップの出力は、[SNS メッセージを送信] ステップへの入力として渡されます。



コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda)

このサンプルプロジェクトは、 AWS Step Functions タスクの実行中に一時停止し、外部プロセスがタスクの開始時に生成されたタスクトークンを返すのを待つ方法を示しています。

このサンプルプロジェクトがデプロイされて、実行が開始すると、次の手順が行われます。

1. Step Functions は、タスクトークンを含むメッセージを Amazon Simple Queue Service (Amazon SQS) キューに渡します。
2. 次にStep Functions は一時停止し、タスクトークンが返されるまで待機します。
3. Amazon SQS キューは、 AWS Lambda `SendTaskSuccess`同じタスクトークンを使用して呼び出す関数をトリガーします。
4. タスクトークンを受信すると、ワークフローは続行します。
5. `"Notify Success"` タスクは、コールバックを受信した Amazon Simple Notification Service (Amazon SNS) メッセージを発行します。

Step Functions にコールバックパターンを実装する方法については、タスクトークンのコールバックまで待機する を参照してください。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、「」を参照してくださいAWS Step Functions 他のサービスとの併用。

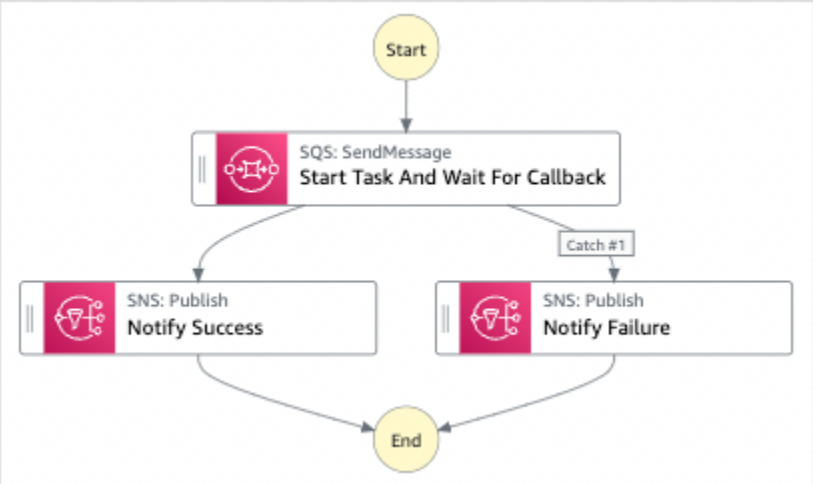
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Callback pattern example`** と入力し、返された検索結果から [コールバックパターンの例] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon SQS メッセージキュー。
   - ステップ関数 API アクションを呼び出す Lambda 関数。SendTaskSuccess
   - ワークフローを続行できるかどうかを示すタスクの成功あるいは失敗を通知する Amazon SNS トピック。
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[コールバックパターンの例] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。

6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

   - [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

     > **Tip**
     >
     > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

     準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

     これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

     サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

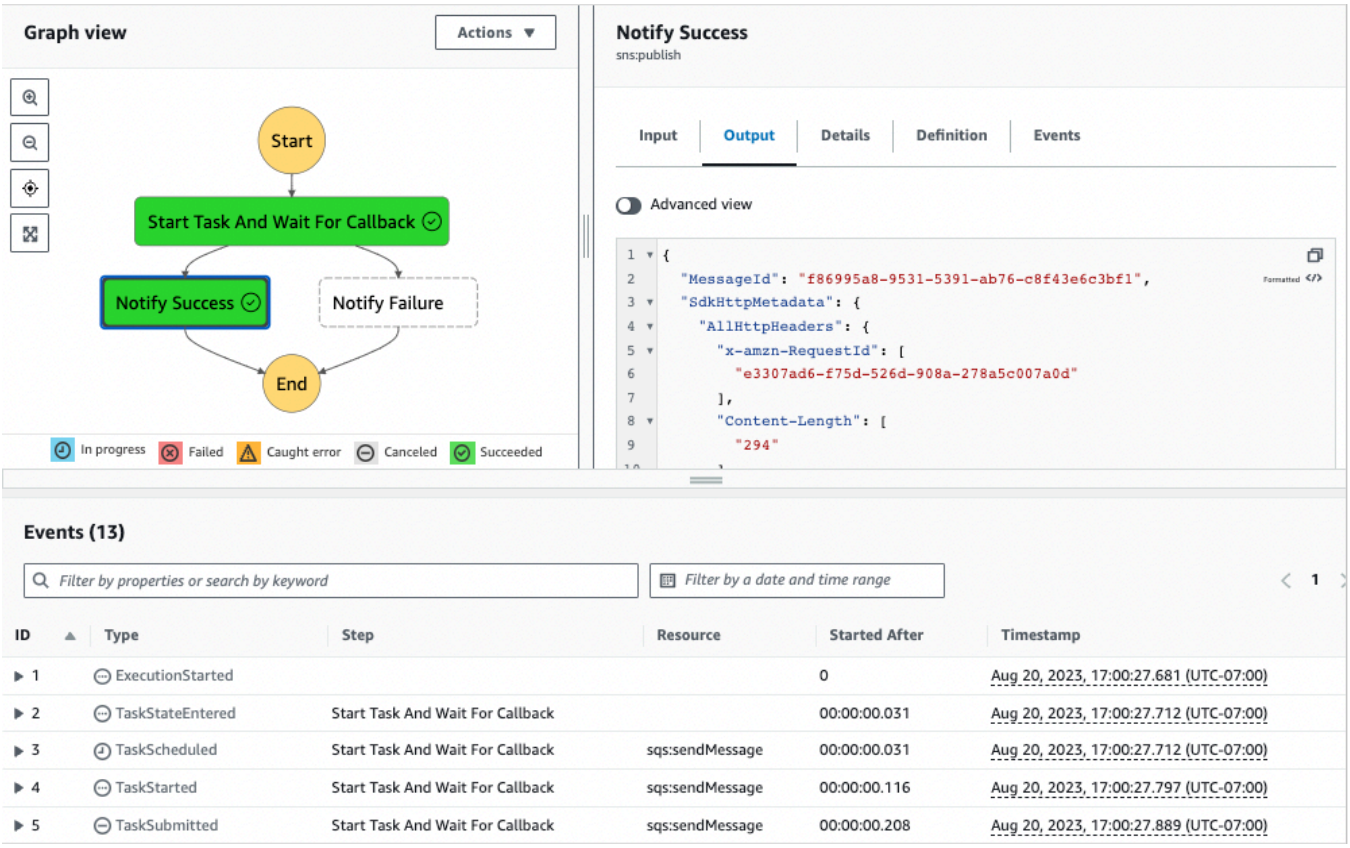
      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

例えば、Step Functions がどのようにワークフローをとおして進行し、Amazon SQS からコールバックを受信しているかを確認するには、[イベント] テーブルのエントリを確認します。次の図は、[成功を通知] ステップの実行出力を示しています。実行イベント履歴の最初の 5 つのイベントも表示されます。各イベントを展開すると、そのイベントの詳細が表示されます。



Lambda コールバックの例

このサンプルプロジェクトの各コンポーネントがどのように連携するかを確認するには、 AWS アカウントにデプロイされたリソースをご覧ください。例えば、次の Lambda 関数はタスクトークンを使用して Step Functions を呼び出します。

```
console.log('Loading function');
const aws = require('aws-sdk');
```

```
exports.lambda_handler = (event, context, callback) => {
    const stepfunctions = new aws.StepFunctions();

    for (const record of event.Records) {
        const messageBody = JSON.parse(record.body);
        const taskToken = messageBody.TaskToken;

        const params = {
            output: "\"Callback task completed successfully.\"",
            taskToken: taskToken
        };

        console.log(`Calling Step Functions to complete callback task with params
 ${JSON.stringify(params)}`);

        stepfunctions.sendTaskSuccess(params, (err, data) => {
            if (err) {
                console.error(err.message);
                callback(err.message);
                return;
            }
            console.log(data);
            callback(null);
        });
    }
};
```

Amazon EMR ジョブを管理する

このサンプルプロジェクトは Amazon EMR AWS Step Functions と統合を示しています。

Amazon EMR クラスターを作成し、複数のステップを追加して実行して、クラスターを終了する方法を示します。

> **Important**
>
> Amazon EMR には無料利用枠はありません。サンプルプロジェクトを実行すると、費用が発生します。料金情報は、Amazon EMRの料金ページに記載されています。Amazon EMR サービス統合の可用性は、Amazon EMR API の可用性により決定します。このため、 AWS

このサンプルプロジェクトは一部のリージョンでは正しく動作しない場合があります。特別なリージョンにおける制限については、Amazon EMR ドキュメントを参照してください。

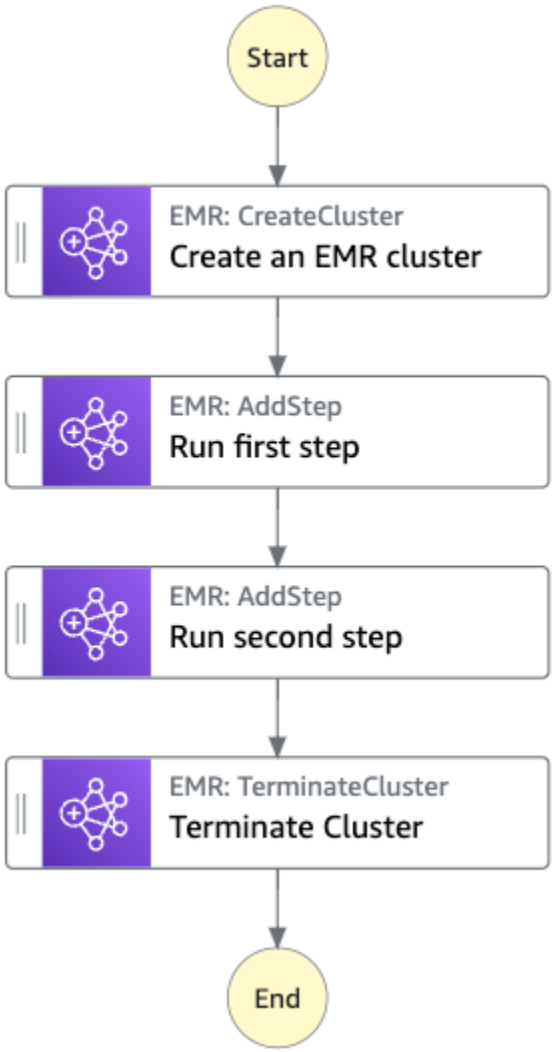
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Manage an EMR job`** と入力し、返された検索結果から [EMR ジョブを管理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトをデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon S3 バケット
   - Amazon EMR クラスター
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[EMR ジョブを管理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルオブジェクトのステートマシンはこれらのリソースにパラメータを直接渡すことで、Amazon EMR と統合します。このサンプルのステートマシンを参照して、Step Functions がどのようにステートマシンを使用して Amazon EMR タスクを同期的に呼び出し、タスクが成功または失敗するのを待機して、クラスターを終了するかをご確認ください。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "An example of the Amazon States Language for running jobs on Amazon EMR",
  "StartAt": "Create an EMR cluster",
  "States": {
    "Create an EMR cluster": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:createCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster",
        "VisibleToAllUsers": true,
        "ReleaseLabel": "emr-5.26.0",
        "Applications": [
          { "Name": "Hive" }
        ],
        "ServiceRole": "<EMR_SERVICE_ROLE>",
        "JobFlowRole": "<EMR_EC2_INSTANCE_PROFILE>",
        "LogUri": "s3://<EMR_LOG_S3_BUCKET>/logs/",
        "Instances": {
          "KeepJobFlowAliveWhenNoSteps": true,
          "InstanceFleets": [
            {
              "Name": "MyMasterFleet",
              "InstanceFleetType": "MASTER",
              "TargetOnDemandCapacity": 1,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
                }
              ]
            },
            {
              "Name": "MyCoreFleet",
              "InstanceFleetType": "CORE",
              "TargetOnDemandCapacity": 1,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
                }
              ]
            }
```

```
          ]
        }
      },
      "ResultPath": "$.cluster",
      "Next": "Run first step"
    },
    "Run first step": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:addStep.sync",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId",
        "Step": {
          "Name": "My first EMR step",
          "ActionOnFailure": "CONTINUE",
          "HadoopJarStep": {
            "Jar": "command-runner.jar",
            "Args": ["<COMMAND_ARGUMENTS>"]
          }
        }
      },
      "Retry" : [
        {
          "ErrorEquals": [ "States.ALL" ],
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
          "BackoffRate": 2.0
        }
      ],
      "ResultPath": "$.firstStep",
      "Next": "Run second step"
    },
    "Run second step": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:addStep.sync",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId",
        "Step": {
          "Name": "My second EMR step",
          "ActionOnFailure": "CONTINUE",
          "HadoopJarStep": {
            "Jar": "command-runner.jar",
            "Args": ["<COMMAND_ARGUMENTS>"]
          }
        }
```

```
      },
      "Retry" : [
        {
          "ErrorEquals": [ "States.ALL" ],
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
          "BackoffRate": 2.0
        }
      ],
      "ResultPath": "$.secondStep",
      "Next": "Terminate Cluster"
    },
    "Terminate Cluster": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:terminateCluster",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId"
      },
      "End": true
    }
  }
}
```

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。ベストプラクティスとして、IAM ポリシーで必要な許可のみ付与することをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:RunJobFlow",
                "elasticmapreduce:DescribeCluster",
                "elasticmapreduce:TerminateJobFlows"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/StepFunctionsSample-EMRJobManagement-
EMRServiceRole-ANPAJ2UCCR6DPCEXAMPLE",
                "arn:aws:iam::123456789012:role/StepFunctionsSample-
EMRJobManagementWJALRXUTNFEMI-ANPAJ2UCCR6DPCEXAMPLE-
EMREc2InstanceProfile-1ANPAJ2UCCR6DPCEXAMPLE"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForEMRRunJobFlowRule"
            ]
        }
    ]
}
```

次のポリシーでは、addStep に十分なアクセス許可があることを確認します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:AddJobFlowSteps",
                "elasticmapreduce:DescribeStep",
                "elasticmapreduce:CancelSteps"
            ],
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
```

```
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForEMRAddJobFlowStepsRule"
            ]
        }
    ]
}
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

EMR Serverless ジョブを実行する

このサンプルプロジェクトは、EMR Serverless アプリケーションを作成して起動する方法を示しています。このプロジェクトでは、そのアプリケーション内で複数のジョブを実行する方法も示しています。

このサンプルプロジェクトは、 AWS ステートマシンとサポートリソースを作成し、関連するIAM権限を設定します。このサンプルプロジェクトについて調べ、Step Functions ステートマシンを使用して EMR Serverless ジョブを実行する方法を学習したり、独自のプロジェクトの出発点として使用したりする方法について説明します。

> **Important**
>
> EMR Serverless には無料利用枠はありません。サンプルプロジェクトを実行すると、費用が発生します。価格情報は、Amazon EMR Serverless の料金表ページに記載されています。また、EMR Serverless サービス統合が利用できるかどうかは、EMR Serverless API が利用できるかどうかによります。このため、このサンプルプロジェクトは正しく動作しないか、一部の AWS リージョンで利用できない可能性があります。 AWS リージョンでの EMR Serverless の可用性については、「その他の考慮事項」トピックを参照してください。

AWS CloudFormation テンプレートと追加リソース

このサンプルプロジェクトをデプロイするには、CloudFormation テンプレートを使用します。このテンプレートは、以下のリソースをユーザーに作成します。 AWS アカウント

- Step Functions ステートマシン。
- ステートマシンの実行ロール。このロールは、 AWS のサービス EMR ServerlessCreateApplicationステートマシンがその他やアクションなどのリソースにアクセスするために必要な権限を付与します。
- EMR Serverless のジョブ実行ロール。このロールは、EMR Serverless ジョブ実行がユーザーに代わって他のサービスを呼び出す際に引き受けることができるアクセス許可を付与します。

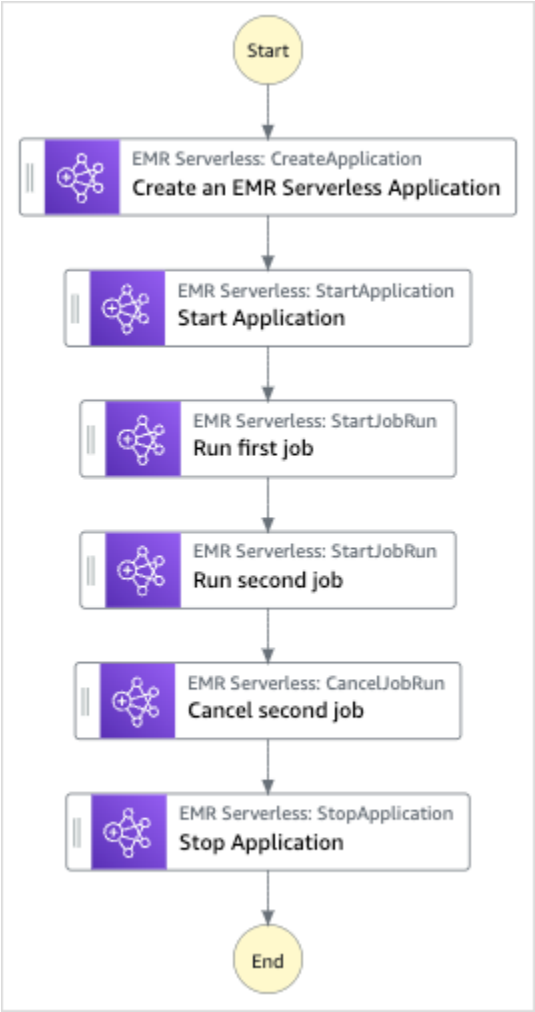
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`EMR Serverless`** と入力し、返された検索結果から [EMR Serverless ジョブを実行] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[EMR Serverless ジョブを実行] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ワークフロー内でワークフローをスタートする (Step Functions、Lambda)

このサンプルプロジェクトは、 AWS Step Functions ステートマシンを使用して他のステートマシンの実行を開始する方法を示しています。別のステートマシンからステートマシンの実行を開始する方法については、「タスク状態からワークフロー実行を開始する」を参照してください。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

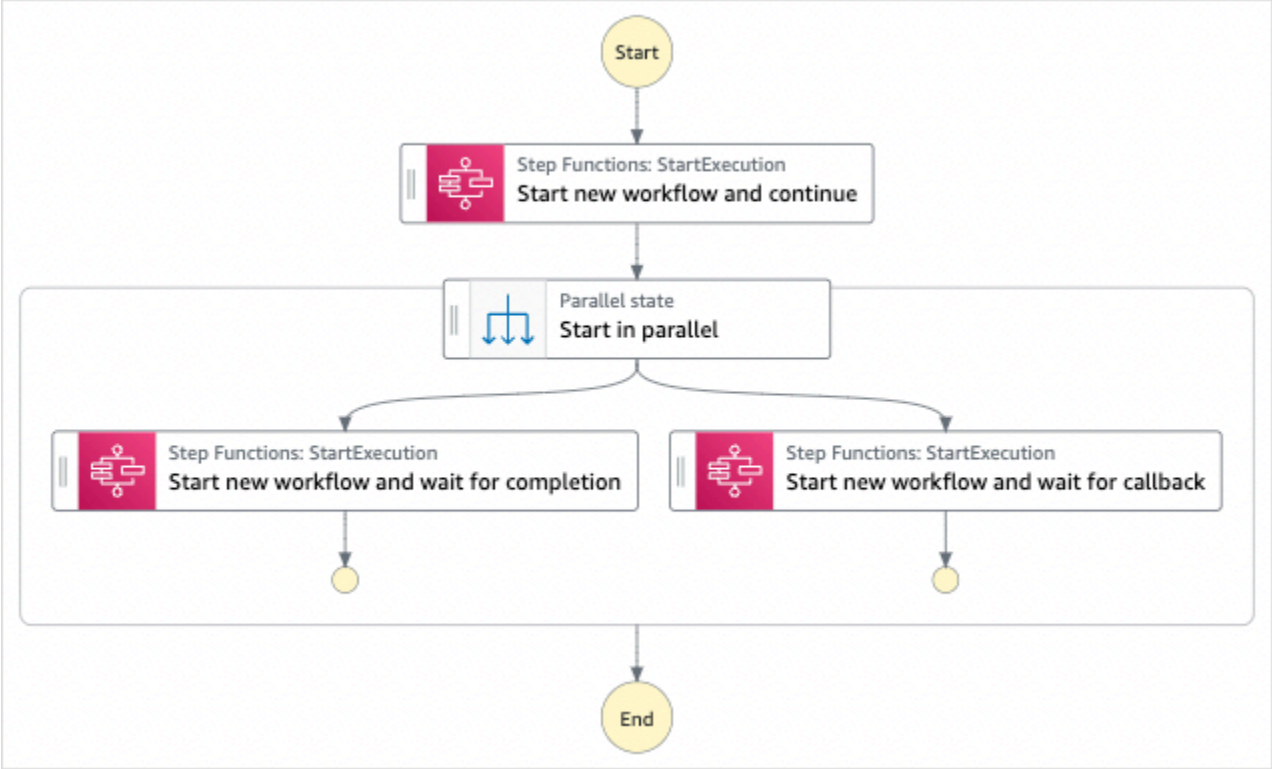
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Start a workflow within a workflow`** と入力し、返された検索結果から [ワークフロー内でワークフローを開始する] を選択します。
3. [次へ] を選択して続行します。

4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - 追加のステートマシン このステートマシンの実行は、実行するステートマシンによって開始されます。
   - コールバック Lambda 関数。この関数は、コールバックメカニズムを実装するために追加のステートマシンで使用されます。
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[ワークフロー内でワークフローを開始する] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは別のステートマシンを統合し、 AWS Lambda パラメーターをそれらのリソースに直接渡します。

このステートマシンの例を参照して、Step Functions が他のステートマシンの `StartExecution` API アクションを呼び出す方法を確認してください。他のステートマシンの 2 つのインスタンスを並行して起動します。1 つは ジョブの実行 (.sync) パターンを使用し、もう 1 つは タスクトークンのコールバックまで待機する パターンを使用します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
  "Comment": "An example of combining workflows using a Step Functions StartExecution
 task state with various integration patterns.",
  "StartAt": "Start new workflow and continue",
  "States": {
    "Start new workflow and continue": {
      "Comment": "Start an execution of another Step Functions state machine and
 continue",
      "Type": "Task",
      "Resource": "arn:aws:states:::states:startExecution",
      "Parameters": {
        "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
        "Input": {
          "NeedCallback": false,
          "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
        }
      },
      "Next": "Start in parallel"
    },
    "Start in parallel": {
      "Comment": "Start two executions of the same state machine in parallel",
```

```
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Start new workflow and wait for completion",
          "States": {
            "Start new workflow and wait for completion": {
              "Comment": "Start an execution of the same
 'NestingPatternAnotherStateMachine' and wait for its completion",
              "Type": "Task",
              "Resource": "arn:aws:states:::states:startExecution.sync",
              "Parameters": {
                "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
                "Input": {
                  "NeedCallback": false,
                  "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
                }
              },
              "OutputPath": "$.Output",
              "End": true
            }
          }
        },
        {
          "StartAt": "Start new workflow and wait for callback",
          "States": {
            "Start new workflow and wait for callback": {
              "Comment": "Start an execution and wait for it to call back with a task
 token",
              "Type": "Task",
              "Resource": "arn:aws:states:::states:startExecution.waitForTaskToken",
              "Parameters": {
                "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
                "Input": {
                  "NeedCallback": true,
                  "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id",
                  "TaskToken.$": "$$.Task.Token"
                }
              },
              "End": true
            }
          }
```

```
        }
      ]
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

マップ状態を使用してデータを動的に処理する

このサンプルプロジェクトでは、 マッピング 状態を使用した動的な並列処理を示します。

このプロジェクトでは、Step Functions AWS Lambda は関数を使用して Amazon SQS キューからメッセージを取得し、それらのメッセージの JSON Map 配列をステートに渡します。キューのメッセージごとに、ステートマシンはメッセージを DynamoDB に書き込み、他の Lambda 関数を呼び出して Amazon SQS からメッセージを削除し、Amazon SNS トピックにメッセージを発行します。

Map 状態と Step Functions サービス統合の詳細については、以下を参照してください。

- マッピング
- AWS Step Functions 他のサービスとの併用

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

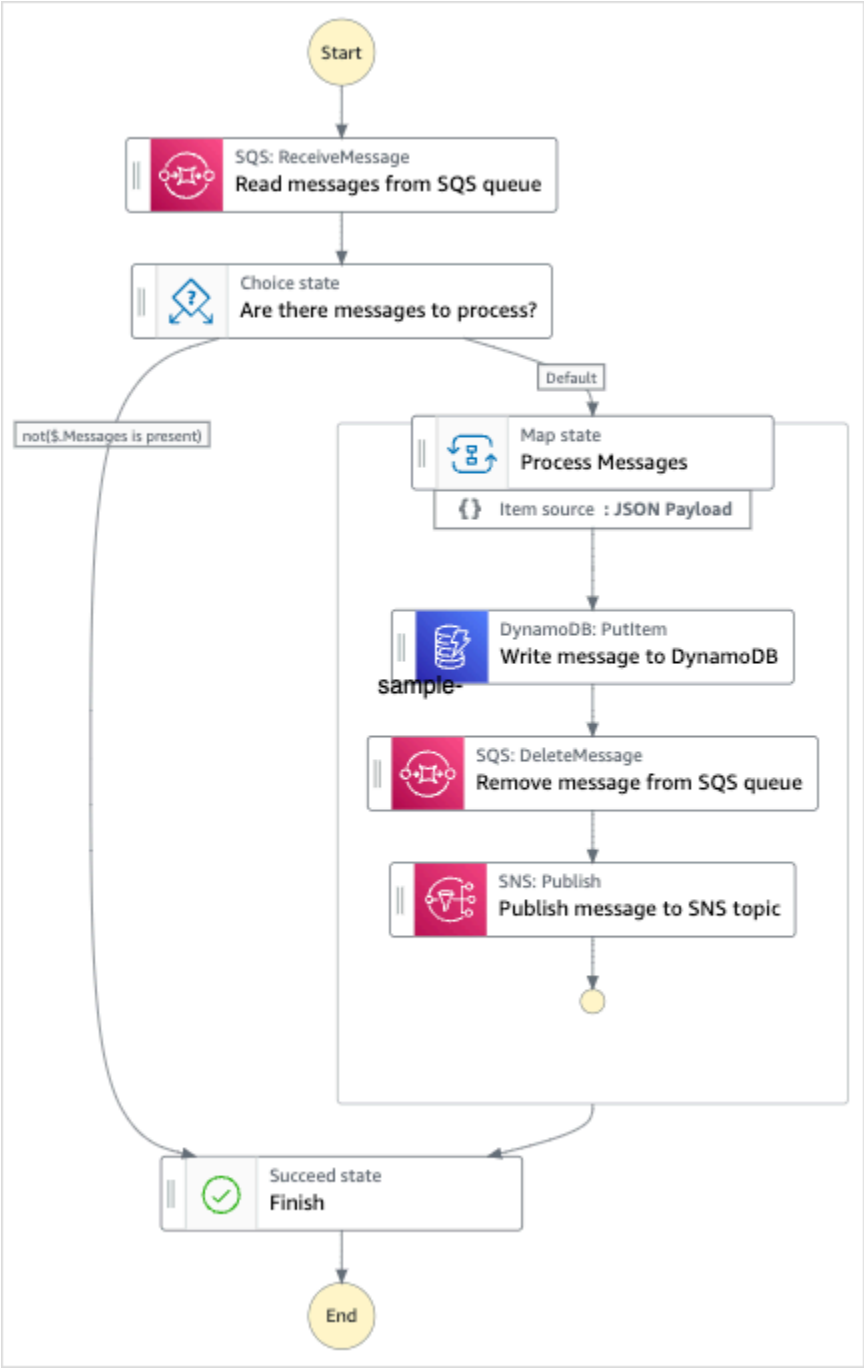
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **Dynamically process data with a Map state** と入力し、返された検索結果から [Map ステートでデータを動的に処理する] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Map ステートによってメッセージが繰り返し読み込まれ、削除される Amazon SQS キュー。

- Map ステートによってメッセージが繰り返し書き込まれる DynamoDB テーブル。
- Step Functions が Amazon SQS キューから読み取ったメッセージをパブリッシュする Amazon SNS トピック。
- 2 AWS Lambda つの機能
- AWS Step Functions ステートマシン
- 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[Map ステートでデータを動的に処理する] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

サンプルプロジェクトのリソースがデプロイされたら、ステートマシンを実行する前に、Amazon SQS キューに項目を追加し、Amazon SNS トピックに登録する必要があります。

ステップ 2: Amazon SNS トピックを登録する

1. [Amazon SNS console] (Amazon SNS コンソール) を開きます。
2. [Topics] (トピック) を選択し、`Map` 状態サンプルプロジェクトで作成されたトピックを選択します。

   名前は MapSampleProj-snStopic-1CQO4HQ3Ir1KN と同じになります。
3. [サブスクリプションを作成] を選択します。

   [Create subscription] (サブスクリプションの作成) ページが表示され、トピックの [Topic ARN] (トピック ARN) が一覧表示されます。
4. [Protocol] (プロトコル) で [Email] (E メール) を選択します。
5. [Endpoint] (エンドポイント) で、トピックにサブスクライブする E メールアドレスを入力します。
6. [Create subscription] (サブスクリプションの作成) を選択します。

   > **Note**
   > サブスクリプションがアクティブになる前に、E メールでの確認が必要です。
7. 関連するアカウントでサブスクリプション確認用 E メールを開き、[Confirm subscription] (サブスクリプションを確認) するための URL を開きます。

   [Subscription confirmed!] (サブスクリプションを確認しました) ページが表示されます。

ステップ 3: Amazon SQS キューにメッセージを追加する

1. [Amazon SQS コンソール] (Amazon SQS コンソール) を開きます。
2. `Map` 状態のサンプルプロジェクトで作成されたキューを選択します。

   名前は MapSampleProj-SQsqueue-1udic9vzDorn7 と似ています。
3. [メッセージの送信と受信] を選択します。
4. [メッセージを送受信] ページで、メッセージを入力し、[メッセージを送信] を選択します。

5. 別のメッセージを入力して [メッセージを送信] を選択します。Amazon SQS キューに複数のメッセージが追加されるまで、メッセージの入力を続けます。

ステップ 4: ステートマシンを実行する

> **Note**
>
> Amazon SNS のキューは結果整合性があります。最良の結果を得るには、キューに入力してからステートマシンの実行を開始するまで数分待ちます。

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。

4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、Amazon SQS、Amazon SNS、および Lambda にパラメータを直接渡すことで、これらのリソースと統合します。

このステートマシンの例を参照し、`Resource` フィールドの Amazon リソースネーム (ARN) に接続し、`Parameters` をサービス API に渡すことで、Step Functions が Lambda、DynamoDB、および Amazon SNS を制御する方法を確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "An example of the Amazon States Language for reading messages from an SQS
 queue and iteratively processing each message.",
  "StartAt": "Read messages from SQS Queue",
  "States": {
    "Read messages from SQS Queue": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "MapSampleProj-ReadFromSQSQueueLambda-1MY3M63RMJVA9"
      },
      "Next": "Are there messages to process?"
    },
    "Are there messages to process?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$",
          "StringEquals": "No messages",
```

```
          "Next": "Finish"
        }
      ],
      "Default": "Process messages"
    },
    "Process messages": {
      "Type": "Map",
      "Next": "Finish",
      "ItemsPath": "$",
      "Parameters": {
        "MessageNumber.$": "$$.Map.Item.Index",
        "MessageDetails.$": "$$.Map.Item.Value"
      },
      "Iterator": {
        "StartAt": "Write message to DynamoDB",
        "States": {
          "Write message to DynamoDB": {
            "Type": "Task",
            "Resource": "arn:aws:states:::dynamodb:putItem",
            "ResultPath": null,
            "Parameters": {
              "TableName": "MapSampleProj-DDBTable-YJDJ1MKIN6C5",
              "ReturnConsumedCapacity": "TOTAL",
              "Item": {
                "MessageId": {
                  "S.$": "$.MessageDetails.MessageId"
                },
                "Body": {
                  "S.$": "$.MessageDetails.Body"
                }
              }
            },
            "Next": "Remove message from SQS queue"
          },
          "Remove message from SQS queue": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "InputPath": "$.MessageDetails",
            "ResultPath": null,
            "Parameters": {
              "FunctionName": "MapSampleProj-DeleteFromSQSQueueLambda-198J2839ZO5K2",
              "Payload": {
                "ReceiptHandle.$": "$.ReceiptHandle"
              }
```

```
            },
            "Next": "Publish message to SNS topic"
          },
          "Publish message to SNS topic": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sns:publish",
            "InputPath": "$.MessageDetails",
            "Parameters": {
              "Subject": "Message from Step Functions!",
              "Message.$": "$.Body",
              "TopicArn": "arn:aws:sns:us-east-1:012345678910:MapSampleProj-
SNSTopic-1CQO4HQ3IR1KN"
            },
            "End": true
          }
        }
      }
    },
    "Finish": {
      "Type": "Succeed"
    }
  }
}
```

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:012345678901:function:MapSampleProj-
ReadFromSQSQueueLambda-1MY3M63RMJVA9",
                "arn:aws:lambda:us-east-1:012345678901:function:MapSampleProj-
DeleteFromSQSQueueLambda-198J2839ZO5K2"
            ],
```

```
            "Effect": "Allow"
        },
        {
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:012345678901:table/MapSampleProj-DDBTable-
YJDJ1MKIN6C5"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:012345678901:MapSampleProj-
SNSTopic-1CQO4HQ3IR1KN"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

分散マップで CSV ファイルを処理する

このサンプルプロジェクトでは、分散マップ状態を使用して、Lambda 関数を使用して生成された CSV ファイルの 10,000 行以上を反復処理する方法を示しています。CSV ファイルにはカスタマーの注文の配送情報が含まれ、Amazon S3 バケットに保存されます。分散マップは CSV ファイル内の 10 行のバッチを繰り返し処理してデータ分析を行います。

分散マップには、遅延した注文を検出する Lambda 関数が組み込まれています。分散マップには、遅延した注文を一括処理し、遅延した注文を配列で返すインラインマップも含まれています。遅延した注文ごとに、インラインマップは Amazon SQS キューにメッセージを送信します。最後に、このサンプルプロジェクトは、マップ実行の結果を AWS アカウント 内の別の Amazon S3 バケットに保存します。

分散マップを使用すると、同時に最大 10,000 件の子ワークフローを並列実行できます。このサンプルプロジェクトでは、分散マップの最大同時実行数は 1000 に設定されており、子ワークフローの並列実行数は 1000 に制限されています。

このサンプルプロジェクトでは、ステートマシンとサポート AWS リソースを作成し、関連する IAM アクセス許可を設定します。このサンプルプロジェクトについて調べ、分散マップを使用して大規模な並列ワークロードをオーケストレーションしたり、独自のプロジェクトの出発点として使用したりする方法について説明します。

AWS CloudFormation テンプレートと追加リソース

CloudFormation テンプレートを使用して、このサンプルプロジェクトをデプロイします。このテンプレートは、 に次のリソースを作成します AWS アカウント。

- Step Functions ステートマシン
- ステートマシンの実行ロール。このロールは、ステートマシンが Lambda 関数の呼び出しアクションなどの他の AWS のサービス およびリソースにアクセスするために必要なアクセス許可を付与します。
- 顧客の注文の詳細を含む CSV ファイルを生成する `CSVGeneratorFunction` という名前の Lambda 関数。
- CSV ジェネレーター Lambda 関数の実行ロール このロールは、他の にアクセスするアクセス許可を関数に付与します AWS のサービス。
- 生成された CSV ファイルを保存する Amazon S3 入力バケット。
- CSV ファイルのデータを分析し、遅延注文を検出する遅延注文検出 Lambda 関数。
- 注文の遅延 Lambda 関数の実行ロール。このロールは、他の にアクセスするアクセス許可を関数に付与します AWS のサービス。
- 顧客の注文の分析結果を保存する Amazon S3 出力バケット。
- Step Functions が遅延した注文ごとにメッセージを送信する Amazon SQS キュー。これらのメッセージには、顧客と注文の ID が含まれます。
- ステートマシンの実行履歴に関連する情報を保存する CloudWatch ロググループ。

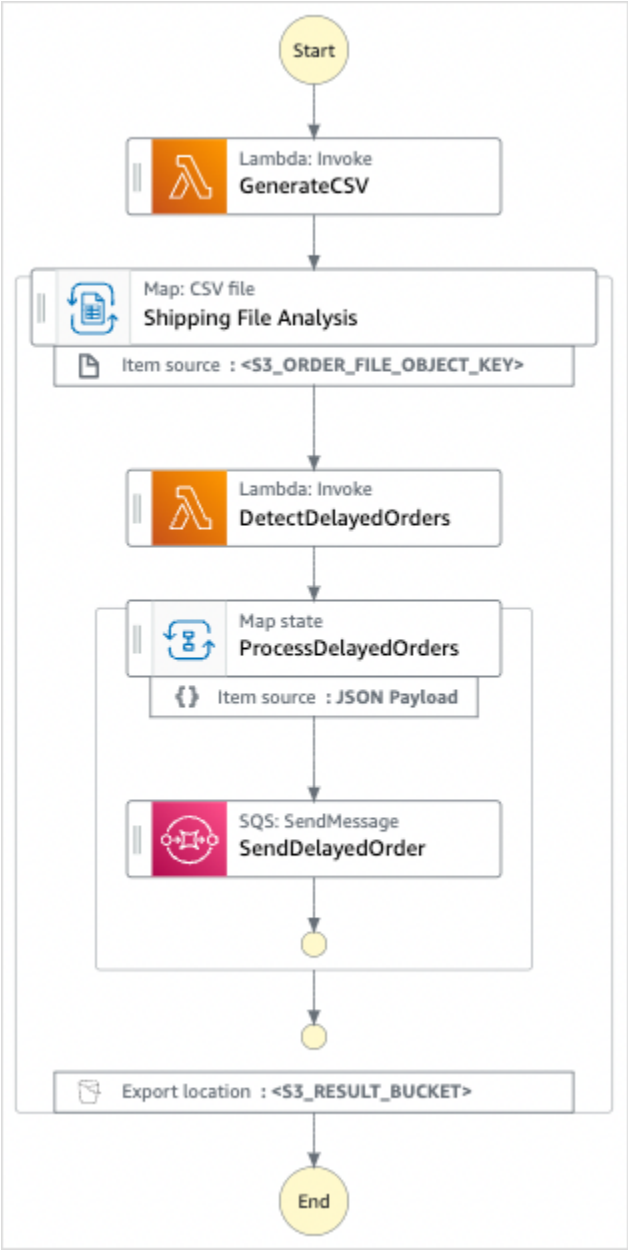
> **Important**
>
> 各サービスには標準料金が適用されます。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Distributed Map to process a CSV file in S3`** と入力し、返された検索結果から [S3 の CSV ファイルを処理する分散マップ] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions は、選択したサンプルプロジェクトで AWS のサービス 使用されている を一覧表示します。サンプルプロジェクトのワークフローグラフも表示されます。このプロジェクトをにデプロイ AWS アカウント するか、独自のプロジェクトを構築するための出発点として使用します。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクト用に作成されるリソースについては、「AWS CloudFormation テンプレートと追加リソース」を参照してください。

   以下のイメージは、[S3 の CSV ファイルを処理する分散マップ] のワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions

コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- デモの実行を選択した場合、Step Functions は、 AWS CloudFormation テンプレートを使用して、そのテンプレートにリストされている AWS リソースを にデプロイする読み取り専用サンプルプロジェクトを作成します AWS アカウント。

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に、 CloudFormation スタック ID リンクを開いて、プロビジョニングされているリソースを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用されるサービスごとに、標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

すべてのリソースをプロビジョニングしてデプロイしたら、ステートマシンを実行できます。

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) JSON 形式で入力値を入力して、サンプルプロジェクトを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   b. [実行のスタート] を選択します。

   c. (オプション) Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行が完了したら、[グラフビュー] で個々のステートを選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各ステートの詳細がそれぞれ表示されます。

      - [実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。
      - 分散マップ状態の実行をコンソールに表示する方法の詳細については、「マップ実行の確認」を参照してください。

   d. (オプション) Amazon S3 バケットにエクスポートされた実行結果を確認します。これらの結果には、実行の入力と出力、ARN、実行ステータスなどのデータが含まれます。詳細については、「ResultWriter」を参照してください。

Amazon S3 バケットのデータを分散マップで処理する

このサンプルプロジェクトでは、分散マップ状態を使用して大規模なデータを処理する方法を示しています。例えば、過去の気象データを分析し、地球上で毎月の平均気温が最も高い気象観測所を特定します。気象データは 12,000 を超える CSV ファイルに記録され、Amazon S3 バケットに保存されています。

このサンプルプロジェクトには、[分散 S3 コピー NOA データ] と [ProcessNOAAData] という名前の 2 つの分散マップ状態が含まれています。分散 S3 コピー NOA データは、 という名前のパブリック Amazon S3 バケット内の CSV ファイルを反復処理noaa-gsod-pdsし、 の Amazon S3 バケットにコピーします AWS アカウント。[ProcessNOAAData] はコピーされたファイルを繰り返し処理し、温度分析を実行する Lambda 関数を含めます。

サンプルプロジェクトは、まず ListObjectsV2 API アクションを呼び出して Amazon S3 バケットの内容を確認します。この呼び出しに応答して返されたキーの数に基づいて、サンプルプロジェクトでは以下のいずれかの判断を行います。

- キー数が 1 以上の場合、プロジェクトは [ProcessNOAAData] ステートに移行します。この分散マップ状態には、毎月平均温度TemperatureFunctionが最も高い気象ステーションを検索する という名前のLambda関数が含まれています。この関数は、キーとして `year-month` を含む辞書と、値として気象観測所に関する情報を含む辞書を返します。
- 返されたキー数が 1 を超えない場合、分散 S3 コピー NOA データ状態は、パブリックバケットのすべてのオブジェクトを一覧表示noaa-gsod-pdsし、個々のオブジェクトを 100 のバッチでアカウント内の別のバケットに繰り返しコピーします。インラインマップはオブジェクトの反復コピーを実行します。

  すべてのオブジェクトがコピーされると、プロジェクトは [ProcessNOAAData] ステートに移行して気象データを処理します。

サンプルプロジェクトは最後に、Lambda関数によって返された結果の最終集計を実行し、結果をAmazon DynamoDBテーブルに書き込むリデューサーTemperatureFunction関数に移行します。

分散マップを使用すると、同時に最大 10,000 件の子ワークフローを並列実行できます。このサンプルプロジェクトでは、[ProcessNOAAData] 分散マップの最大同時実行数は 3000 件に設定されており、子ワークフローの並列実行数は 3000 件に制限されています。

このサンプルプロジェクトでは、ステートマシンとサポート AWS リソースを作成し、関連する IAM アクセス許可を設定します。このサンプルプロジェクトについて調べ、分散マップを使用して大規模な並列ワークロードをオーケストレーションしたり、独自のプロジェクトの出発点として使用したりする方法について説明します。

> **Important**
>
> このサンプルプロジェクトは米国東部 (バージニア北部) リージョン限定で利用できます。

AWS CloudFormation テンプレートと追加リソース

CloudFormation テンプレートを使用して、このサンプルプロジェクトをデプロイします。このテンプレートは、 に次のリソースを作成します AWS アカウント。

- Step Functions ステートマシン
- ステートマシンの実行ロール。このロールは、ステートマシンが Lambda 関数の呼び出しアクションなどの他の AWS のサービス およびリソースにアクセスするために必要なアクセス許可を付与します。
- `NOAADataBucket` という名前の Amazon S3 バケット。このバケットには、気象データを含む CSV ファイルが含まれています。
- 気象データの最終的な集計を実行し、その結果を Amazon DynamoDB テーブルに書き込む `ReducerFunction` という名前の Lambda 関数。
- リデューサー Lambda 関数の実行ロール このロールは、他の にアクセスするアクセス許可を関数に付与します AWS のサービス。
- 気象分析の結果を保存するように `ResultsBucket` と名付けられた Amazon S3 出力バケット。
- `ReducerFunction` によって返された結果が格納されている `ResultsDynamoDBTable` という名前の DynamoDB テーブル。
- 月間平均気温の最高値を求める `TemperatureFunction` という名前の Lambda 関数。
- Lambda 関数の実行ロール このロールは、他の にアクセスするアクセス許可を関数に付与します AWS のサービス。
- ステートマシンの実行履歴に関連する情報を保存する CloudWatch ロググループ。

> **Important**
>
> 各サービスには標準料金が適用されます。

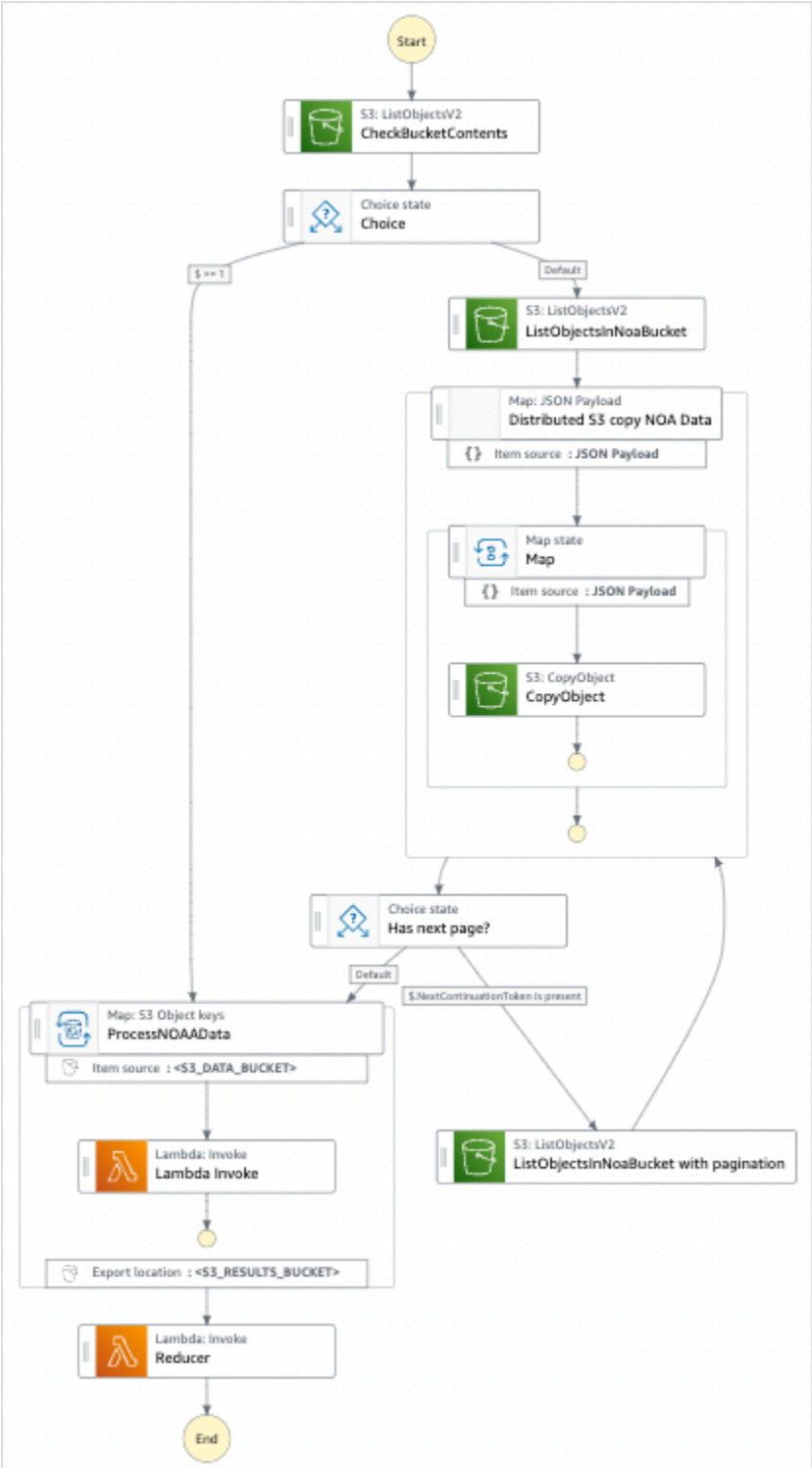
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Distributed Map to process files in S3`** と入力し、返された検索結果から [S3 のファイルを処理する分散マップ] を選択します。
3. [次へ] を選択して続行します。

4. Step Functions は、選択したサンプルプロジェクトで AWS のサービス 使用されている を一覧表示します。サンプルプロジェクトのワークフローグラフも表示されます。このプロジェクトをにデプロイ AWS アカウント するか、独自のプロジェクトを構築するための出発点として使用します。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクト用に作成されるリソースについては、「AWS CloudFormation テンプレートと追加リソース」を参照してください。

   以下のイメージは、[S3 のファイルを処理する分散マップ] のワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- デモの実行を選択した場合、Step Functions は、 AWS CloudFormation テンプレートを使用して、そのテンプレートにリストされている AWS リソースを にデプロイする読み取り専用サンプルプロジェクトを作成します AWS アカウント。

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に、 CloudFormation スタック ID リンクを開いて、プロビジョニングされているリソースを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用されるサービスごとに、標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

すべてのリソースをプロビジョニングしてデプロイしたら、ステートマシンを実行できます。

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   a. (オプション) JSON 形式で入力値を入力して、サンプルプロジェクトを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   b. [実行のスタート] を選択します。

   c. (オプション) Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行が完了したら、[グラフビュー] で個々のステートを選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各ステートの詳細がそれぞれ表示されます。

      - [実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。
      - 分散マップ状態の実行をコンソールに表示する方法の詳細については、「マップ実行の確認」を参照してください。

d. (オプション) Amazon S3 バケットにエクスポートされた実行結果を確認します。これらの結果には、実行の入力と出力、ARN、実行ステータスなどのデータが含まれます。詳細については、「ResultWriter」を参照してください。

機械学習モデルのトレーニング

このサンプルプロジェクトでは、 SageMaker AWS Step Functions 機械学習モデルの使用方法とトレーニング方法、およびテストデータセットのバッチ変換方法を示します。

このプロジェクトでは、Step Functions は Lambda 関数を使用して Amazon S3 バケットにテストデータセットをシードします。次に、SageMaker サービスインテグレーションを使用して機械学習モデルをトレーニングし、バッチ変換を実行します。

Step Functions サービスインテグレーションの詳細については、以下を参照してください。 SageMaker

- AWS Step Functions 他のサービスとの併用
- SageMaker Step Functions による管理

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「SageMaker 料金表」を参照してください。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

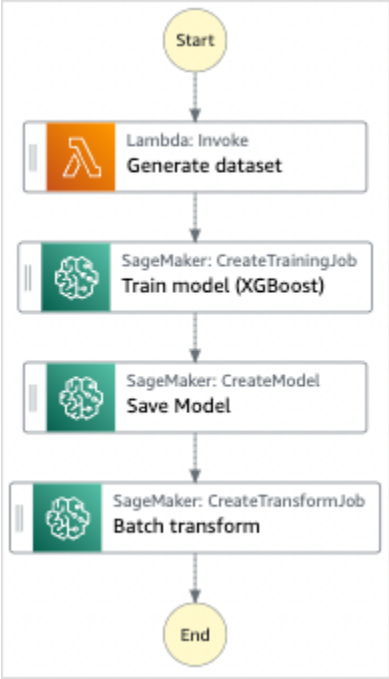
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Train a machine learning model`** と入力し、返された検索結果から [機械学習モデルをトレーニングする] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築

するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

このサンプルプロジェクトは、以下のリソースをデプロイします。

- AWS Lambda 関数。
- 1 つの Amazon Simple Storage Service (Amazon S3) バケット
- AWS Step Functions ステートマシン
- 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[機械学習モデルをトレーニングする] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions

コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。

2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、 SageMaker AWS Lambda これらのリソースと統合したり、パラメータを直接渡したりして、トレーニングデータソースと出力に Amazon S3 バケットを使用します。

このステートマシンの例を参照して、Step Functions が Lambda とをどのように制御するかを確認してください。 SageMaker

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "StartAt": "Generate dataset",
  "States": {
    "Generate dataset": {
      "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:TrainAndBatchTransform-SeedingFunction-17RNSOTG97HPV",
      "Type": "Task",
      "Next": "Train model (XGBoost)"
    },
    "Train model (XGBoost)": {
      "Resource": "arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters": {
        "AlgorithmSpecification": {
          "TrainingImage": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
          "TrainingInputMode": "File"
        },
        "OutputDataConfig": {
          "S3OutputPath": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/models"
        },
        "StoppingCondition": {
          "MaxRuntimeInSeconds": 86400
        },
        "ResourceConfig": {
          "InstanceCount": 1,
          "InstanceType": "ml.m4.xlarge",
          "VolumeSizeInGB": 30
        },
        "RoleArn": "arn:aws:iam::123456789012:role/TrainAndBatchTransform-
SageMakerAPIExecutionRole-Y9IX3DLF6EUO",
        "InputDataConfig": [
          {
            "DataSource": {
              "S3DataSource": {
                "S3DataDistributionType": "ShardedByS3Key",
                "S3DataType": "S3Prefix",
```

```
                "S3Uri": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/csv/
train.csv"
              }
            },
            "ChannelName": "train",
            "ContentType": "text/csv"
          }
        ],
        "HyperParameters": {
          "objective": "reg:logistic",
          "eval_metric": "rmse",
          "num_round": "5"
        },
        "TrainingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "Next": "Save Model"
    },
    "Save Model": {
      "Parameters": {
        "PrimaryContainer": {
          "Image": "433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest",
          "Environment": {},
          "ModelDataUrl.$": "$.ModelArtifacts.S3ModelArtifacts"
        },
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/TrainAndBatchTransform-
SageMakerAPIExecutionRole-Y9IX3DLF6EUO",
        "ModelName.$": "$.TrainingJobName"
      },
      "Resource": "arn:aws:states:::sagemaker:createModel",
      "Type": "Task",
      "Next": "Batch transform"
    },
    "Batch transform": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
      "Parameters": {
        "ModelName.$": "$$.Execution.Name",
        "TransformInput": {
          "CompressionType": "None",
          "ContentType": "text/csv",
          "DataSource": {
            "S3DataSource": {
              "S3DataType": "S3Prefix",
```

```
              "S3Uri": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/csv/
test.csv"
            }
          }
        },
        "TransformOutput": {
          "S3OutputPath": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/output"
        },
        "TransformResources": {
          "InstanceCount": 1,
          "InstanceType": "ml.m4.xlarge"
        },
        "TransformJobName.$": "$$.Execution.Name"
      },
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー](#)。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なあれらの許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
```

```
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

次のポリシーでは、Amazon S3 バケットにサンプルデータをシードすることを Lambda 関数に許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::trainandbatchtransform-s3bucket-1jn1le6gadwfz/*",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

機械学習モデルのチューニング

このサンプルプロジェクトでは、 SageMaker 機械学習モデルのハイパーパラメータを調整したり、テストデータセットをバッチ変換したりする方法を示しています。

このプロジェクトでは、Step Functions は Lambda 関数を使用して Amazon S3 バケットにテストデータセットをシードします。次に、サービスインテグレーションを使用してハイパーパラメータチューニングジョブを作成します。SageMaker次に、Lambda 関数を使用してデータパスを抽出し、チューニングモデルを保存し、モデル名を抽出し、バッチ変換ジョブを実行して推論を実行します。 SageMaker

Step Functions サービスインテグレーションの詳細については、以下を参照してください。 SageMaker

- AWS Step Functions 他のサービスとの併用
- SageMaker Step Functions による管理

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用してもー定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「SageMaker料金表」を参照してください。

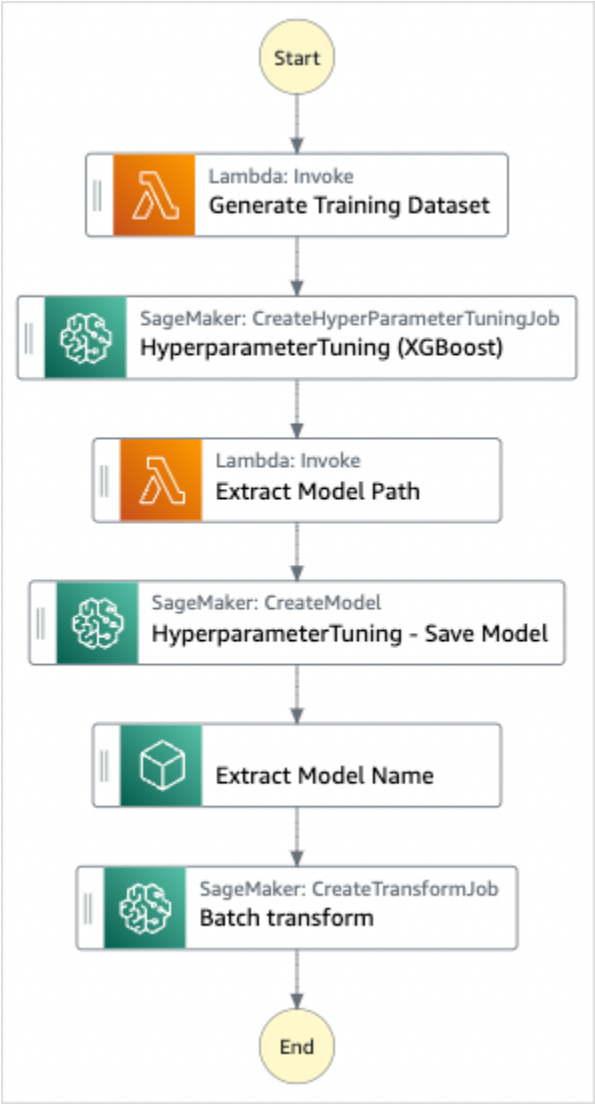
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Tune a machine learning model`** と入力し、返された検索結果から [機械学習モデルをチューニング] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトをデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - 3 AWS Lambda つの機能
   - 1 つの Amazon Simple Storage Service (Amazon S3) バケット
   - AWS Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[機械学習モデルをチューニング] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、 SageMaker AWS Lambda これらのリソースと統合したり、パラメータを直接渡したりして、トレーニングデータソースと出力に Amazon S3 バケットを使用します。

このステートマシンの例を参照して、Step Functions が Lambda とをどのように制御するかを確認してください。 SageMaker

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "StartAt": "Generate Training Dataset",
    "States": {
        "Generate Training Dataset": {
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageMa-
LambdaForDataGeneration-1TF67BUE5A12U",
            "Type": "Task",
            "Next": "HyperparameterTuning (XGBoost)"
        },
        "HyperparameterTuning (XGBoost)": {
            "Resource":
 "arn:aws:states:::sagemaker:createHyperParameterTuningJob.sync",
            "Parameters": {
                "HyperParameterTuningJobName.$": "$.body.jobName",
                "HyperParameterTuningJobConfig": {
                    "Strategy": "Bayesian",
                    "HyperParameterTuningJobObjective": {
                        "Type": "Minimize",
                        "MetricName": "validation:rmse"
                    },
                    "ResourceLimits": {
                        "MaxNumberOfTrainingJobs": 2,
                        "MaxParallelTrainingJobs": 2
                    },
                    "ParameterRanges": {
                        "ContinuousParameterRanges": [{
                                "Name": "alpha",
                                "MinValue": "0",
                                "MaxValue": "1000",
                                "ScalingType": "Auto"
                            },
                            {
                                "Name": "gamma",
                                "MinValue": "0",
                                "MaxValue": "5",
                                "ScalingType": "Auto"
                            }
                        ],
                        "IntegerParameterRanges": [{
```

```
                                "Name": "max_delta_step",
                                "MinValue": "0",
                                "MaxValue": "10",
                                "ScalingType": "Auto"
                            },
                            {
                                "Name": "max_depth",
                                "MinValue": "0",
                                "MaxValue": "10",
                                "ScalingType": "Auto"
                            }
                        ]
                    }
                },
                "TrainingJobDefinition": {
                    "AlgorithmSpecification": {
                        "TrainingImage": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
                        "TrainingInputMode": "File"
                    },
                    "OutputDataConfig": {
                        "S3OutputPath": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/models"
                    },
                    "StoppingCondition": {
                        "MaxRuntimeInSeconds": 86400
                    },
                    "ResourceConfig": {
                        "InstanceCount": 1,
                        "InstanceType": "ml.m4.xlarge",
                        "VolumeSizeInGB": 30
                    },
                    "RoleArn": "arn:aws:iam::012345678912:role/StepFunctionsSample-
SageM-SageMakerAPIExecutionRol-1MNH1VS5CGGOG",
                    "InputDataConfig": [{
                        "DataSource": {
                            "S3DataSource": {
                                "S3DataDistributionType": "FullyReplicated",
                                "S3DataType": "S3Prefix",
                                "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/train.csv"
                            }
                        },
                        "ChannelName": "train",
```

```
                        "ContentType": "text/csv"
                    },
                    {
                        "DataSource": {
                            "S3DataSource": {
                                "S3DataDistributionType": "FullyReplicated",
                                "S3DataType": "S3Prefix",
                                "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/validation.csv"
                            }
                        },
                        "ChannelName": "validation",
                        "ContentType": "text/csv"
                    }],
                    "StaticHyperParameters": {
                        "precision_dtype": "float32",
                        "num_round": "2"
                    }
                }
            },
            "Type": "Task",
            "Next": "Extract Model Path"
        },
        "Extract Model Path": {
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageM-LambdaToExtractModelPath-
V0R37CVARUS9",
            "Type": "Task",
            "Next": "HyperparameterTuning - Save Model"
        },
        "HyperparameterTuning - Save Model": {
            "Parameters": {
                "PrimaryContainer": {
                    "Image": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
                    "Environment": {},
                    "ModelDataUrl.$": "$.body.modelDataUrl"
                },
                "ExecutionRoleArn": "arn:aws:iam::012345678912:role/
StepFunctionsSample-SageM-SageMakerAPIExecutionRol-1MNH1VS5CGGOG",
                "ModelName.$": "$.body.bestTrainingJobName"
            },
            "Resource": "arn:aws:states:::sagemaker:createModel",
            "Type": "Task",
```

```
            "Next": "Extract Model Name"
        },
        "Extract Model Name": {
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageM-
LambdaToExtractModelName-8FUOB30SM5EM",
            "Type": "Task",
            "Next": "Batch transform"
        },
        "Batch transform": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
            "Parameters": {
                "ModelName.$": "$.body.jobName",
                "TransformInput": {
                    "CompressionType": "None",
                    "ContentType": "text/csv",
                    "DataSource": {
                        "S3DataSource": {
                            "S3DataType": "S3Prefix",
                            "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/test.csv"
                        }
                    }
                },
                "TransformOutput": {
                    "S3OutputPath": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/output"
                },
                "TransformResources": {
                    "InstanceCount": 1,
                    "InstanceType": "ml.m4.xlarge"
                },
                "TransformJobName.$": "$.body.jobName"
            },
            "End": true
        }
    }
}
```

他の AWS のサービスで Step Functions を使用する時に IAM を設定する方法の情報については、統合サービスの IAM ポリシー を参照してください。

IAM の例

サンプルプロジェクトによって生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

次の IAM ポリシーがステートマシンにアタッチされ、 SageMakerステートマシンの実行が必要な Lambda および Amazon S3 リソースにアクセスできるようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sagemaker:CreateHyperParameterTuningJob",
                "sagemaker:DescribeHyperParameterTuningJob",
                "sagemaker:StopHyperParameterTuningJob",
                "sagemaker:ListTags",
                "sagemaker:CreateModel",
                "sagemaker:CreateTransformJob",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageMa-LambdaForDataGeneration-1TF67BUE5A12U",
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageM-LambdaToExtractModelPath-V0R37CVARUS9",
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageM-LambdaToExtractModelName-8FUOB30SM5EM"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
```

```
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTrainingJobsRule",
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTransformJobsRule",
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTuningJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

次の IAM ポリシーは、HyperparameterTuning 状態のTrainingJobDefinition フィールドおよび HyperparameterTuning フィールドで参照されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "sagemaker:DescribeHyperParameterTuningJob",
                "sagemaker:StopHyperParameterTuningJob",
                "sagemaker:ListTags"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
```

```
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f",
            "Effect": "Allow"
        }
    ]
}
```

次の IAM ポリシー では、Amazon S3 バケットにサンプルデータをシードすることを Lambda 関数に許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/*",
            "Effect": "Allow"
        }
    ]
}
```

他の AWS のサービスで Step Functions を使用する時に IAM を設定する方法の情報については、「統合サービスの IAM ポリシー」を参照してください。

Amazon SQS からの大容量メッセージの処理 (Express ワークフロー)

このサンプルプロジェクトでは、 AWS Step Functions Express ワークフローを使用して、Amazon Simple Queue Service (Amazon SQS) などの大量のイベントソースからのメッセージまたはデータ

を処理する方法を示します。Express ワークフローは非常に高いレートで開始できるため、大容量のイベント処理やストリーミングデータワークロードに最適です。

イベントソースからステートマシンを実行するために一般的に使用される 2 つの方法は次のとおりです。

- イベントソースが CloudWatch イベントを発行するたびにステートマシンの実行を開始するように Amazon Events ルールを設定します。詳細については、CloudWatch 「イベント でトリガーするイベントルールの作成」を参照してください。
- イベントソースを Lambda 関数にマッピングし、ステートマシンを実行する関数コードを記述します。 AWS Lambda 関数は、イベントソースがイベントを発行するたびに呼び出され、ステートマシンの実行が開始されます。詳細については、Amazon SQS を使った AWS Lambda の使用を参照してください。

このサンプルプロジェクトでは、2 番目の方法を使用して Amazon SQS キューがメッセージを送信するたびに実行をスタートします。同様の設定を使用して、Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB、Amazon Kinesis などの他のイベントソースから Express ワークフローの実行をトリガーできます。

Express ワークフローと Step Functions サービス統合の詳細については、以下を参照してください。

- 標準ワークフロー対 Express ワークフロー
- AWS Step Functions 他のサービスとの併用
- クォータ

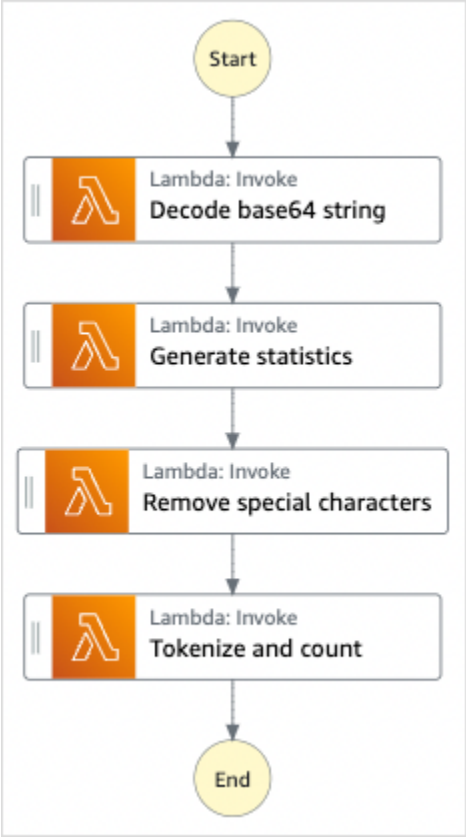
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Process high-volume messages from SQS`** と入力し、返された検索結果から [SQS からの大容量メッセージを処理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions は、選択したサンプルプロジェクトで AWS のサービス 使用されている を一覧表示します。サンプルプロジェクトのワークフローグラフも表示されます。このプロジェクトをにデプロイ AWS アカウント するか、独自のプロジェクトを構築するための出発点として使用します。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

このサンプルプロジェクトは、以下のリソースをデプロイします。

- 4 つの Lambda 関数
- Amazon SQSキュー
- AWS Step Functions ステートマシン
- 関連する AWS Identity and Access Management (IAM) ロール

以下のイメージは、[SQS からの大容量メッセージを処理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS

Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- デモの実行を選択した場合、Step Functions は、 AWS CloudFormation テンプレートを使用して、そのテンプレートにリストされている AWS リソースを にデプロイする読み取り専用サンプルプロジェクトを作成します AWS アカウント。

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に、 CloudFormation スタック ID リンクを開いて、プロビジョニングされているリソースを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用されるサービスごとに、標準料金が適用される場合があります。

ステップ 2: ステートマシンの実行をトリガーする

1. [Amazon SQS console] (Amazon SQS コンソール) を開きます。
2. サンプルプロジェクトで作成されたキューを選択します。

   名前は Example-SQSQueue-wJalrXUtnFEMI のようになっています。
3. [Queue Actions] (キュー操作) リストで、[Send a Message] (メッセージの送信) を選択します。
4. コピーボタンを使用して次のメッセージをコピーし、[Send a Message] (メッセージの送信) ウィンドウに入力して、[Send Message] (メッセージの送信) を選択します。

> **Note**
>
> このサンプルメッセージでは、`input:` 行がページに合わせて改行されています。コピーボタンを使用するか、改行なしの 1 行として入力されていることを確認します。

```
{
      "input":
 "QW5kIGxpa2UgdGhlIGJhc2VsZXNzIGZhYnJpYyBvZiB0aGlzIHZpc2lvbiwgVGhlIGNsb3VkLWNhcHBlZCB0b3dlc

  91cyBwYWxhY2VzLCBUaGUgc29sZW1uIHRlbXBsZXMsIHRoZSBncmVhdCBnbG9iZSBpdHNlbGbigJQgWWVhLCBhbGw

  ZXJpdOKAlHNoYWxsIGRpc3NvbHZlLCBBbmQgbGlrZSB0aGlzIGluc3Vic3RhbnRpYWwgcGFnZWFudCBmYWRlZCwgT

  FjayBiZWhpbmQuIFdlIGFyZSBzdWNoIHN0dWZmIEFzIGRyZWFtcyBhcmUgbWFkZSBvbiwgYW5kIG91ciBsaXR0bGU

  ZGVkIHdpdGggYSBzbGVlcC4gU2lyLCBJIGFtIHZleGVkLiBCZWFyIHdpdGggbXkgd2Vha25lc3MuIE15IG9sZCBic

  xlZC4gQmUgbm90IGRpc3R1cmJlZCB3aXRoIG15IGluZmlybWl0eS4gSWYgeW91IGJlIHBsZWFzZWQsIHJldGlyZSB

  QW5kIHRoZXJlIHJlcG9zZS4gQSB0dXJuIG9yIHR3byBJ4oCZbGwgd2FsayBUbyByBzdGlsbCBteSBiZWF0aW5nIG1pb
}
```

5. [Close] (閉じる) を選択します。
6. Step Functions コンソールを開きます。
7. Amazon CloudWatch Logs ロググループに移動し、ログを調べます。ロググループの名前は example-ExpressLogGroup-wJalrXUtnFEMI のようになります。

Lambda 関数のコードの例

以下は、開始する Lambda 関数が AWS SDK を使用してステートマシンの実行を開始する方法を示す Lambda 関数コードです。

```
import boto3

def lambda_handler(event, context):
    message_body = event['Records'][0]['body']
    client = boto3.client('stepfunctions')
    response = client.start_execution(
        stateMachineArn='${ExpressStateMachineArn}',
        input=message_body
    )
```

ステートマシンのコード例

このサンプルプロジェクトの Express ワークフローは、テキスト処理用の Lambda 関数のセットで構成されています。

AWS Step Functions が他の AWS のサービスをコントロールする方法の詳細については、「」を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
  "Comment": "An example of using Express workflows to run text processing for each
 message sent from an SQS queue.",
  "StartAt": "Decode base64 string",
  "States": {
    "Decode base64 string": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<BASE64_DECODER_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "Next": "Generate statistics"
    },
    "Generate statistics": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
```

```
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<TEXT_STATS_GENERATING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "Next": "Remove special characters"
    },
    "Remove special characters": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<STRING_CLEANING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "Next": "Tokenize and count"
    },
    "Tokenize and count": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<TOKENIZING_AND_WORD_COUNTING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "End": true
    }
  }
}
```

IAM の例

サンプルプロジェクトによって生成されたこの AWS Identity and Access Management (IAM) ポリシーの例には、ステートマシンと関連リソースを実行するために必要な最小権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
```

```
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:example-
Base64DecodeLambda-wJalrXUtnFEMI",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
StringCleanerLambda-je7MtGbClwBF",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
TokenizerCounterLambda-wJalrXUtnFEMI",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
GenerateStatsLambda-je7MtGbClwBF"
            ],
            "Effect": "Allow"
        }
    ]
}
```

次のポリシーは、 CloudWatch ログに対する十分なアクセス許可があることを確認します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

他の AWS サービスで Step Functions を使用するときに IAM を設定する方法については、「」を参照してください統合サービスの IAM ポリシー。

選択的チェックポイントの例 (Express ワークフロー)

このサンプルプロジェクトでは、選択的チェックポイントを行う模擬 e コマースワークフローを実行して、標準ワークフローと Express ワークフローを組み合わせる方法を示します。このサンプルプロジェクトをデプロイすると、スタンダードワークフローステートマシン、ネストされた Express ワークフローステートマシン、 AWS Lambda 関数、Amazon Simple Queue Service (Amazon SQS) キュー、および Amazon Simple Notification Service (Amazon SNS) トピックが作成されます。

Express ワークフロー、ネストされたワークフロー、および Step Functions サービス統合の詳細については、以下を参照してください。

- 標準ワークフロー対 Express ワークフロー
- タスク状態からワークフロー実行を開始する
- AWS Step Functions 他のサービスとの併用

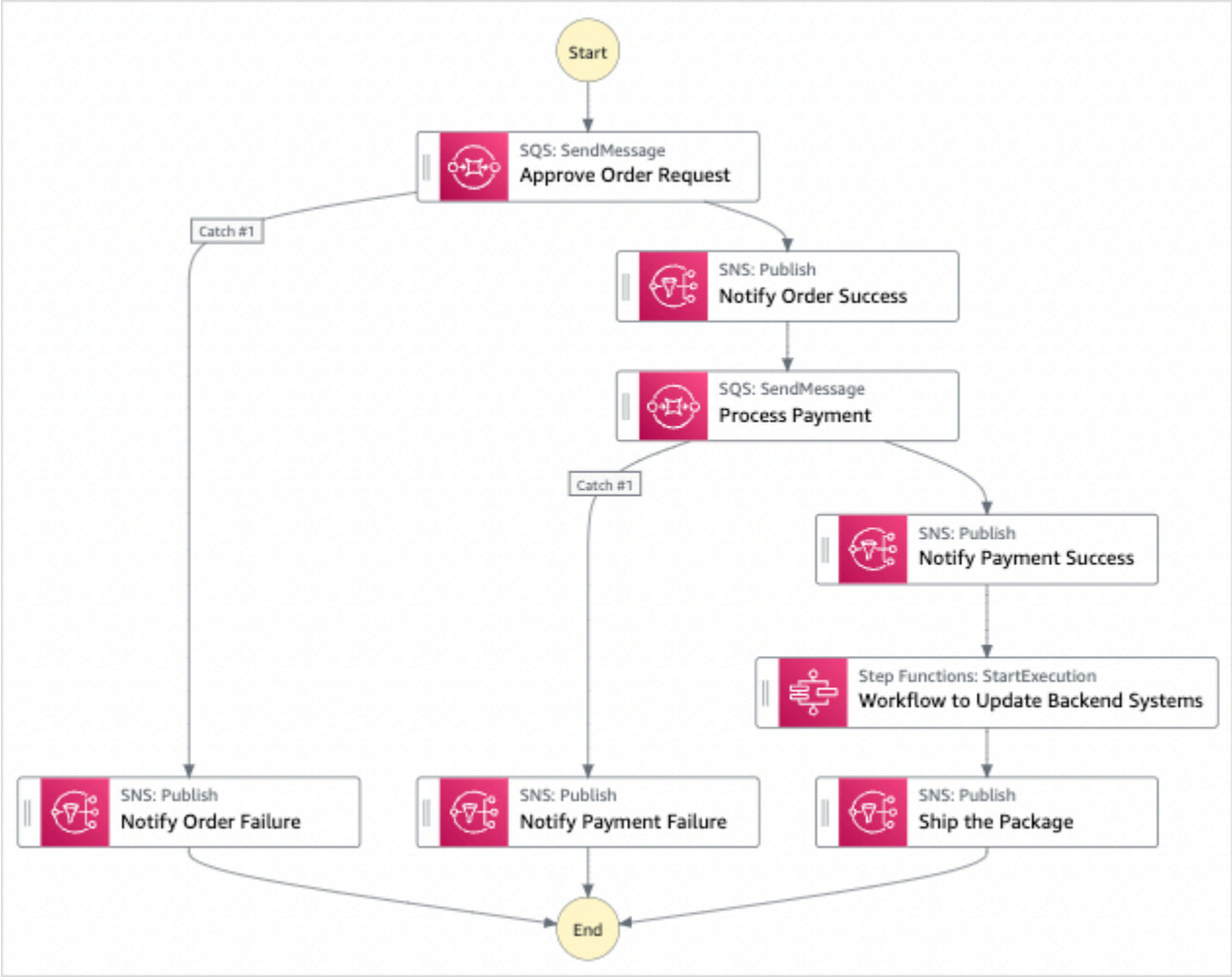
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Selective checkpointing example`** と入力し、返された検索結果から [選択的なチェックポイントの例] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS Lambda 関数
   - Amazon SQSキュー
   - 1 つの Amazon SNS トピック
   - AWS Step Functions スタンダードタイプのステートマシン。
   - タイプ Express の Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[選択的なチェックポイントの例] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してく

ださい。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

サンプルプロジェクトのリソースがデプロイされたら、次の操作を行います。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。

2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

4. CloudWatch Logs ロググループに移動して、ログを調べてください。ロググループの名前は、例-ExpressLogGroup-wJalrXUtnFEMI のようになります。

親のステートマシンのコード例 (標準ワークフロー)

このサンプルプロジェクトのステートマシンは、Amazon SQS、Amazon SNS および Step Functions Express ワークフローと統合されています。

このサンプルステートマシンを参照して、Step Functions プロセスが、Amazon SQS と Amazon SNS からの入力を処理する方法を確認し、ネストされた Express ワークフローステートマシンを使用してバックエンドシステムを更新します。

他のサービスを制御する方法の詳細については、を参照してください。 AWS Step Functions AWS [AWS Step Functions 他のサービスとの併用]

```
{
    "Comment": "An example of combining standard and express workflows to run a mock e-commerce workflow that does selective checkpointing.",
    "StartAt": "Approve Order Request",
    "States": {
        "Approve Order Request": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "<SQS_QUEUE_URL>",
                "MessageBody": {
                    "MessageTitle": "Order Request received. Pausing workflow to wait for manual approval. ",
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "Next": "Notify Order Success",
            "Catch": [
                {
                    "ErrorEquals": [
                        "States.ALL"
                    ],
                    "Next": "Notify Order Failure"
                }
            ]
        },
        "Notify Order Success": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order has been approved. Resuming workflow.",
                "TopicArn": "<SNS_ARN>"
            },
            "Next": "Process Payment"
        },
        "Notify Order Failure": {
```

```
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order not approved. Order failed.",
                "TopicArn": "<SNS_ARN>"
            },
            "End": true
        },
        "Process Payment": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "<SQS_QUEUE_URL>",
                "MessageBody": {
                    "MessageTitle": "Payment sent to third-party for processing.
 Pausing workflow to wait for response.",
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "Next": "Notify Payment Success",
            "Catch": [
                {
                    "ErrorEquals": [
                        "States.ALL"
                    ],
                    "Next": "Notify Payment Failure"
                }
            ]
        },
        "Notify Payment Success": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Payment processing succeeded. Resuming workflow.",
                "TopicArn": "<SNS_ARN>"
            },
            "Next": "Workflow to Update Backend Systems"
        },
        "Notify Payment Failure": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Payment processing failed.",
                "TopicArn": "<SNS_ARN>"
```

```
            },
            "End": true
        },
        "Workflow to Update Backend Systems": {
            "Comment": "Starting an execution of an Express workflow to handle backend
 updates. Express workflows are fast and cost-effective for steps where checkpointing
 isn't required.",
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::states:startExecution.sync",
            "Parameters": {
                "StateMachineArn": "<UPDATE_DATABASE_EXPRESS_STATE_MACHINE_ARN>",
                "Input": {
                    "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
                }
            },
            "Next": "Ship the Package"
        },
        "Ship the Package": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order and payment received, database is updated and the
 package is ready to ship.",
                "TopicArn": "<SNS_ARN>"
            },
            "End": true
        }
    }
}
```

親ステートマシンの IAM ロール の例

サンプルプロジェクトによって生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

Amazon SNS ポリシー:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
```

```
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:us-east-1:123456789012:Checkpoint-SNSTopic-
wJalrXUtnFEMI",
            "Effect": "Allow"
        }
    ]
}
```

Amazon SQS ポリシー:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:us-east-1:123456789012:Checkpoint-SQSQueue-
je7MtGbClwBF",
            "Effect": "Allow"
        }
    ]
}
```

状態実行ポリシー:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "states:StartExecution",
                "states:DescribeExecution",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
```

```
                "events:DescribeRule"
            ],
            "Resource": "arn:aws:events:us-east-1:123456789012:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule",
            "Effect": "Allow"
        }
    ]
}
```

ネストされたステートマシン (Express ワークフロー) のステートマシンコードの例

このサンプルプロジェクトのステートマシンは、親ステートマシンから呼び出されたときにバックエンド情報を更新します。

このサンプルステートマシンを参照して、Step Functions が模擬 e コマースバックエンドシステムのさまざまなコンポーネントを更新する方法を確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
```

```
  "StartAt": "Update Order History",
  "States": {
    "Update Order History": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update order history."
        }
      },
      "Next": "Update Data Warehouse"
    },
    "Update Data Warehouse": {
      "Type" : "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update data warehouse."
        }
      },
      "Next": "Update Customer Profile"
    },
    "Update Customer Profile": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update customer profile."
        }
      },
      "Next": "Update Inventory"
    },
    "Update Inventory": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update inventory."
        }
      },
```

```
      "End": true
    }
  }
}
```

子ステートマシンの IAM ロールの例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:Example-
UpdateDatabaseLambdaFunction-wJalrXUtnFEMI"
            ],
            "Effect": "Allow"
        }
    ]
}
```

以下のポリシーは、 CloudWatch ログに十分な権限があることを確認するものです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
```

```
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

他の AWS のサービスで Step Functions を使用する時に IAM を設定する方法の情報については、統合サービスの IAM ポリシー を参照してください。

AWS CodeBuild プロジェクトを構築する (CodeBuild、Amazon SNS)

このサンプルプロジェクトは、 AWS Step Functions AWS CodeBuild を使用してプロジェクトを構築し、テストを実行して Amazon SNS 通知を送信する方法を示しています。

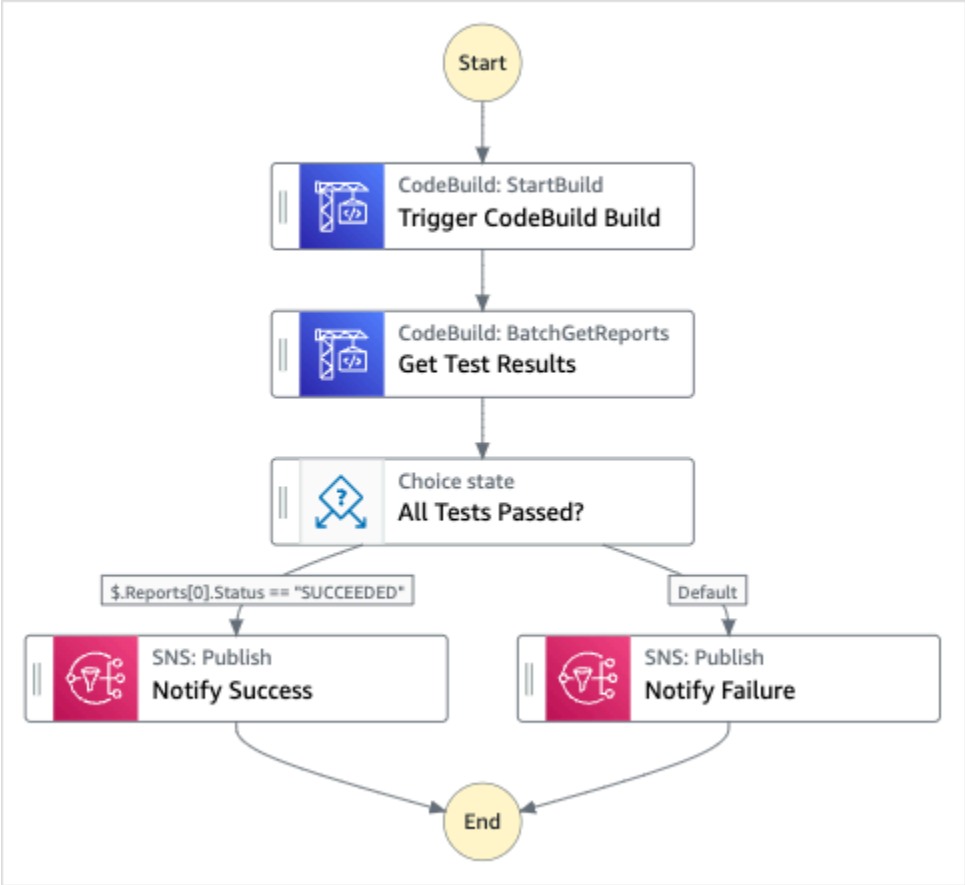
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. **`Start a CodeBuild build`**検索ボックスに入力し、返された検索結果から [Start a CodeBuild build] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS CodeBuild ビルド
   - 1 つの Amazon SNS トピック
   - AWS Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

以下の画像は、Start a CodeBuild build サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは Amazon SNS CodeBuild と統合されています。

このステートマシンの例を参照して、Step Functions CodeBuild がステートマシンを使用してプロジェクトを構築し、ジョブが成功したか失敗したかを示すメッセージとともに Amazon SNS トピックを送信する方法を確認してください。

Step Functions AWS が他のサービスを制御する方法の詳細については、を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
  "Comment": "An example of using CodeBuild to run tests, get test results and send a
 notification.",
  "StartAt": "Trigger CodeBuild Build",
  "States": {
    "Trigger CodeBuild Build": {
      "Type": "Task",
      "Resource": "arn:aws:states:::codebuild:startBuild.sync",
      "Parameters": {
        "ProjectName": "CodeBuildProject-Dtw1jBhEYGDf"
      },
      "Next": "Get Test Results"
    },
    "Get Test Results": {
      "Type": "Task",
      "Resource": "arn:aws:states:::codebuild:batchGetReports",
      "Parameters": {
        "ReportArns.$": "$.Build.ReportArns"
      },
      "Next": "All Tests Passed?"
    },
    "All Tests Passed?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.Reports[0].Status",
          "StringEquals": "SUCCEEDED",
          "Next": "Notify Success"
        }
      ],
      "Default": "Notify Failure"
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "CodeBuild build tests succeeded",
        "TopicArn": "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution3da9ead6-bc1f-4441-99ac-591c140019c4-SNSTopic-EVYLVNGW85JP"
      },
```

```
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "CodeBuild build tests failed",
        "TopicArn": "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution3da9ead6-bc1f-4441-99ac-591c140019c4-SNSTopic-EVYLVNGW85JP"
      },
      "End": true
    }
  }
}
```

他の AWS のサービスで Step Functions を使用する時に IAM を設定する方法の情報については、統合サービスの IAM ポリシー を参照してください。

データを前処理し、機械学習モデルをトレーニングする

このサンプルプロジェクトでは、 SageMaker AWS Step Functions データを使用および前処理し、機械学習モデルをトレーニングする方法を紹介します。

このプロジェクトでは、Step Functions は Lambda 関数を使用して Amazon S3 バケットにテストデータセットとデータ処理用の Python スクリプトをシードします。次に、SageMaker サービスインテグレーションを使用して機械学習モデルをトレーニングし、バッチ変換を実行します。

Step Functions サービスインテグレーションの詳細については、以下を参照してください。
SageMaker

- AWS Step Functions 他のサービスとの併用
- SageMaker Step Functions による管理

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用してもー定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「SageMaker 料金表」を参照してください。

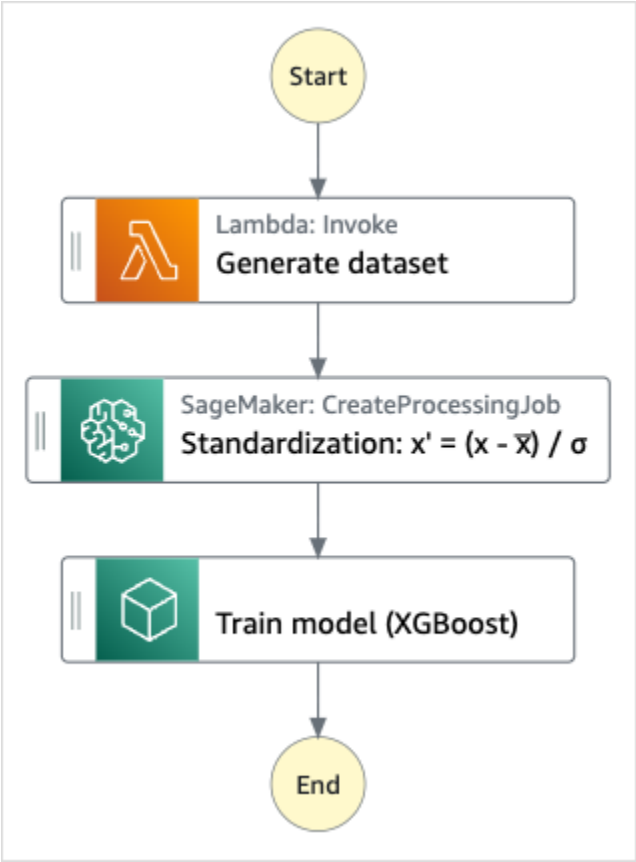
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Preprocess data and train a machine learning model`** と入力し、返された検索結果から [データを事前処理し、機械学習モデルをトレーニングする] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS Lambda 関数
   - Amazon S3 バケット
   - AWS Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[データを事前処理して機械学習モデルをトレーニングする] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、 SageMaker AWS Lambda これらのリソースと統合したり、パラメータを直接渡したりして、トレーニングデータソースと出力に Amazon S3 バケットを使用します。

このステートマシンの例を参照して、Step Functions が Lambda とをどのように制御するかを確認してください。 SageMaker

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "StartAt": "Generate dataset",
  "States": {
    "Generate dataset": {
      "Resource": "arn:aws:lambda:sa-east-1:1234567890:function:FeatureTransform-
LambaForDataGeneration-17M8LX7IO9LUW",
      "Type": "Task",
      "Next": "Standardization: x' = (x - x̄) / σ"
    },
    "Standardization: x' = (x - x̄) / σ": {
      "Resource": "arn:aws:states:::sagemaker:createProcessingJob.sync",
      "Parameters": {
        "ProcessingResources": {
          "ClusterConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.m5.xlarge",
            "VolumeSizeInGB": 10
          }
        },
        "ProcessingInputs": [
            {
                "InputName": "input-1",
                "S3Input": {
                    "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
input/raw.csv",
                    "LocalPath": "/opt/ml/processing/input",
                    "S3DataType": "S3Prefix",
                    "S3InputMode": "File",
                    "S3DataDistributionType": "FullyReplicated",
                    "S3CompressionType": "None"
                }
            },
            {
                "InputName": "code",
                "S3Input": {
                    "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
code/transform.py",
                    "LocalPath": "/opt/ml/processing/input/code",
                    "S3DataType": "S3Prefix",
                    "S3InputMode": "File",
                    "S3DataDistributionType": "FullyReplicated",
                    "S3CompressionType": "None"
                }
```

```
            }
        ],
        "ProcessingOutputConfig": {
            "Outputs": [
                {
                    "OutputName": "train_data",
                    "S3Output": {
                        "S3Uri": "s3://featuretransform-
bucketforcodeanddata-1jn1le6gadwfz/train",
                        "LocalPath": "/opt/ml/processing/output/train",
                        "S3UploadMode": "EndOfJob"
                    }
                }
            ]
        },
        "AppSpecification": {
            "ImageUri": "737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-
learn:0.20.0-cpu-py3",
            "ContainerEntrypoint": [
                "python3",
                "/opt/ml/processing/input/code/transform.py"
            ]
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 300
        },
        "RoleArn": "arn:aws:iam::1234567890:role/SageMakerAPIExecutionRole-
AIDACKCEVSQ6C2EXAMPLE",
        "ProcessingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "Next": "Train model (XGBoost)"
    },
    "Train model (XGBoost)": {
      "Resource": "arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters": {
        "AlgorithmSpecification": {
          "TrainingImage": "855470959533.dkr.ecr.sa-east-1.amazonaws.com/
xgboost:latest",
          "TrainingInputMode": "File"
        },
        "OutputDataConfig": {
          "S3OutputPath": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
models"
```

```
        },
        "StoppingCondition": {
          "MaxRuntimeInSeconds": 86400
        },
        "ResourceConfig": {
          "InstanceCount": 1,
          "InstanceType": "ml.m5.xlarge",
          "VolumeSizeInGB": 30
        },
        "RoleArn": "arn:aws:iam::1234567890:role/SageMakerAPIExecutionRole-
AIDACKCEVSQ6C2EXAMPLE",
        "InputDataConfig": [
          {
            "DataSource": {
              "S3DataSource": {
                "S3DataDistributionType": "ShardedByS3Key",
                "S3DataType": "S3Prefix",
                "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz"
              }
            },
            "ChannelName": "train",
            "ContentType": "text/csv"
          }
        ],
        "HyperParameters": {
          "objective": "reg:logistic",
          "eval_metric": "rmse",
          "num_round": "5"
        },
        "TrainingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー]。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なあれらの許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

次のポリシーでは、Amazon S3 バケットにサンプルデータをシードすることを Lambda 関数に許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::featuretransform-
bucketforcodeanddata-1jn1le6gadwfz/*",
```

```
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Lambda オーケストレーションの例

このサンプルプロジェクトでは、Step AWS Lambda Functions ステートマシンに関数を統合する方法を紹介します。

このプロジェクトでは、Step Functions は Lambda 関数を使用して株価をチェックし、売買取引レコメンデーションを決定します。その後、ユーザーにはこのレコメンデーションが提供され、株式の売買を選択できます。取引の結果は SNS トピックを使用して返されます。

Step Functions サービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- 以下の IAM ポリシー:
  - の IAM ポリシー AWS Lambda
  - Amazon SQS の IAM ポリシー
  - Amazon SNS の IAM ポリシー

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「料金表」を参照してください。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

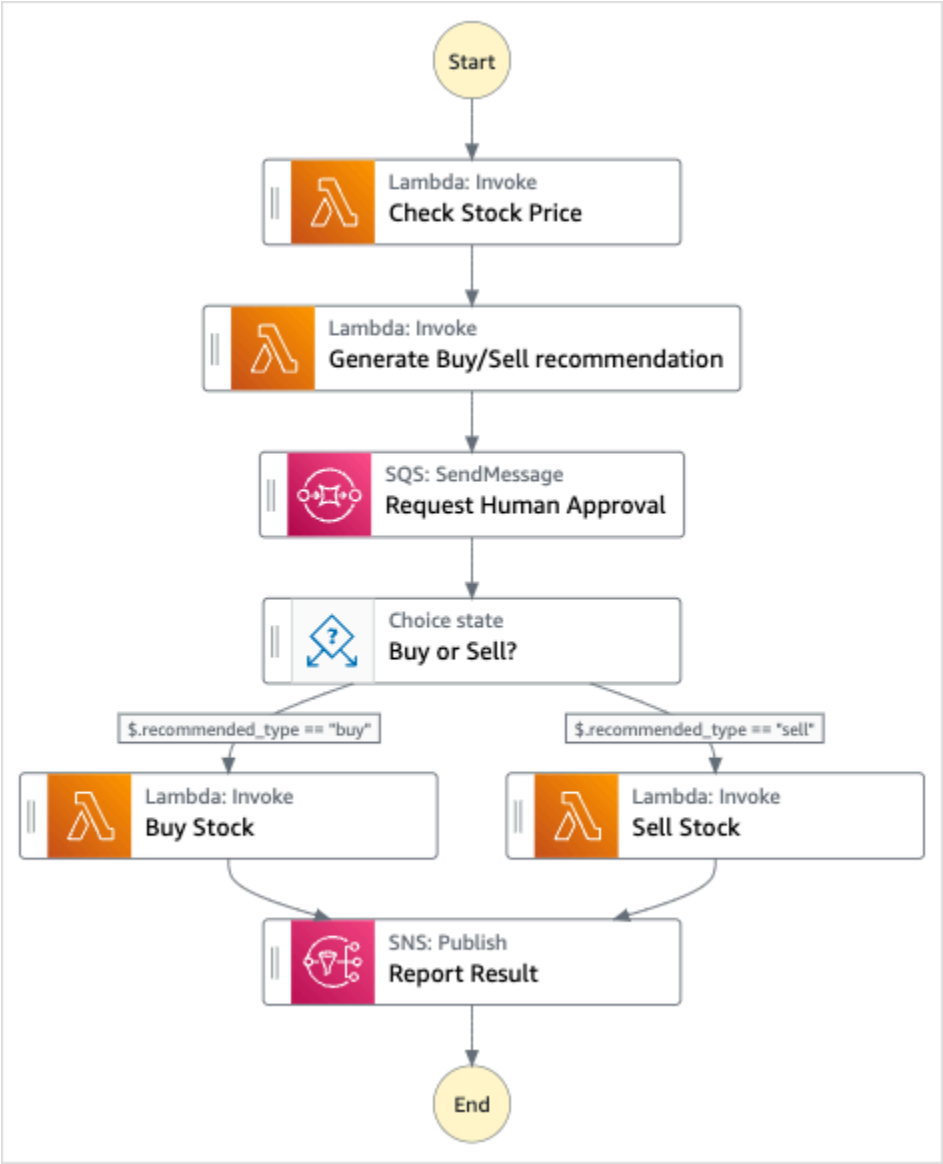
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。

2. 検索ボックスに **`Orchestrate Lambda functions`** と入力し、返された検索結果から [Lambda 関数のオーケストレーション] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - 5 つの Lambda 関数
   - Amazon Simple Queue Service キュー
   - Amazon Simple Notification Service トピック
   - AWS Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[Lambda 関数のオーケストレーション] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してく

ださい。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

すべてのリソースのプロビジョニングとデプロイが完了すると、[実行を開始] ダイアログボックスが表示されます。

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

    1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

        > **Note**
        >
        > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

    2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

        [デモの実行] を選択した場合、実行入力を入力する必要はありません。

        > **Note**
        >
        > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

    3. [実行のスタート] を選択します。
    4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

        実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンとその実行について

このサンプルプロジェクトのステートマシンは、 AWS Lambda パラメータを直接リソースに渡すことでと統合し、Amazon SQS キューを使用して人間による承認リクエストを管理し、Amazon SNS トピックを使用してクエリの結果を返します。

Step Functions を実行すると、JSON テキストを入力として受け取り、その入力をワークフローの最初の状態に渡します。それぞれの状態で JSON データを入力として受け取ります。また、通常 JSON データを出力として次の状態に渡します。このサンプルプロジェクトでは、各ステップの出力がワークフローの次のステップへの入力として渡されます。例えば、[買い/売りのレコメンデーションを生成] ステップでは、[株価を確認] ステップの出力を入力として受け取ります。また、[買い/売りのレコメンデーションを生成] ステップの出力は、人間の承認ステップを模倣した次のステップ [人間による承認をリクエスト] に入力として渡されます。

> **Note**
>
> ステップによって返された出力とステップに渡された入力を確認するには、ワークフロー実行の [実行の詳細] ページを開きます。ステップの詳細セクションでは、表示モードで選択した各ステップの入力と出力が表示されます。

人間による承認ステップを実装するには、通常、タスクトークンが返されるまでワークフローの実行を一時停止します。このサンプルプロジェクトでは、メッセージが Amazon SQS キューに渡され、コールバック機能を処理するために定義された Lambda 関数のトリガーとして使用されます。メッセージにはタスクトークンと、前のステップで返された出力が含まれています。Lambda 関数はメッセージのペイロードで呼び出されます。ワークフローの実行は、`SendTaskSuccess` API 呼び出しでタスクトークンが返されるまで一時停止されます。タスクの詳細については、「タスクトークンのコールバックまで待機する」を参照してください。

次の `StepFunctionsSample-HelloLambda-ApproveSqsLambda` 関数のコードは、Step Functions ステートマシンを通じて Amazon SQS キューから送信されたタスクを自動的に承認するように定義する方法を示しています。

コールバック機能を処理してタスクトークンを返す Lambda 関数コードの例

```
exports.lambdaHandler = (event, context, callback) => {
  const stepfunctions = new aws.StepFunctions();

  // For every record in sqs queue
  for (const record of event.Records) {
      const messageBody = JSON.parse(record.body);
      const taskToken = messageBody.TaskToken;

      const params = {
          output: "\"approved\"",
          taskToken: taskToken
```

```
      };

      console.log(`Calling Step Functions to complete callback task with params
 ${JSON.stringify(params)}`);

      // Approve
      stepfunctions.sendTaskSuccess(params, (err, data) => {
          if (err) {
              console.error(err.message);
              callback(err.message);
              return;
          }
          console.log(data);
          callback(null);
      });
  }
};
```

このステートマシンの例を参照して、 Step Functions が Lambda と Amazon SQS をどのように制御するかを確認します。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "StartAt": "Check Stock Price",
    "States": {
        "Check Stock Price": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLam-
CheckStockPriceLambda-444455556666",
            "Next": "Generate Buy/Sell recommendation"
        },
        "Generate Buy/Sell recommendation": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-Hello-
GenerateBuySellRecommend-123456789012",
            "ResultPath": "$.recommended_type",
            "Next": "Request Human Approval"
        },
        "Request Human Approval": {
```

```
            "Type": "Task",
            "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "https://sqs.us-west-1.amazonaws.com/111122223333/
StepFunctionsSample-HelloLambda4444-5555-6666-RequestHumanApprovalSqs-777788889999",
                "MessageBody": {
                    "Input.$": "$",
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "ResultPath": null,
            "Next": "Buy or Sell?"
        },
        "Buy or Sell?": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.recommended_type",
                    "StringEquals": "buy",
                    "Next": "Buy Stock"
                },
                {
                    "Variable": "$.recommended_type",
                    "StringEquals": "sell",
                    "Next": "Sell Stock"
                }
            ]
        },
        "Buy Stock": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
BuyStockLambda-000000000000",
            "Next": "Report Result"
        },
        "Sell Stock": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
SellStockLambda-111111111111",
            "Next": "Report Result"
        },
        "Report Result": {
            "Type": "Task",
```

```
            "Resource": "arn:aws:states:::sns:publish",
            "Parameters": {
                "TopicArn": "arn:aws:sns:us-west-1:111122223333:StepFunctionsSample-
HelloLambda1111-2222-3333-ReportResultSnsTopic-222222222222",
                "Message": {
                    "Input.$": "$"
                }
            },
            "End": true
        }
    }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLam-
CheckStockPriceLambda-444455556666",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
```

```
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-Hello-
GenerateBuySellRecommend-123456789012",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
BuyStockLambda-777788889999",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
SellStockLambda-000000000000",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage*"
            ],
```

```
            "Resource": "arn:aws:sqs:us-west-1:111122223333:StepFunctionsSample-
HelloLambda4444-5555-6666-RequestHumanApprovalSqs-111111111111",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:us-west-1:111122223333:StepFunctionsSample-
HelloLambda1111-2222-3333-ReportResultSnsTopic-222222222222",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Athena クエリをスタートする

このサンプルプロジェクトは標準ワークフローに基づいており、Step Functions と Amazon Athena を使用して Athena クエリをスタートし、クエリ結果を含む通知を送信する方法を示します。

このプロジェクトでは、Step Functions は Lambda AWS Glue 関数とクローラーを使用してサンプルデータのセットを生成します。次に、Athena サービス統合を使ってクエリを実施し、SNS トピックを使用して結果を返します。

Athena と Step Functions のサービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Step Functions で Athena を呼び出す

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。

AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠の詳細については、「Athena の料金表」を参照してください。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Start an Athena query`** と入力し、返された検索結果から [Athena クエリを開始する] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトをデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon Athena クエリ
   - AWS Glue クローラー
   - Amazon SNS トピック
   - AWS Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[Athena クエリを開始する] サンプルプロジェクトのワークフローグラフを示しています。

5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは Athena AWS Lambda と統合し、パラメータをそれらのリソースに直接渡し、SNS トピックを使用してクエリの結果を返します。

このステートマシンの例を参照して、Step Functions が Lambda と Athena をどのように制御するかを確認します。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "StartAt": "Generate example log",
  "States": {
    "Generate example log": {
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athena-LambdaForDataGeneration-AKIAIOSFODNN7EXAMPLE",
      "Type": "Task",
      "Next": "Run Glue crawler"
    },
    "Run Glue crawler": {
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athen-LambdaForInvokingCrawler-AKIAI44QH8DHBEXAMPLE",
      "Type": "Task",
      "Next": "Start an Athena query"
    },
    "Start an Athena query": {
      "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
      "Parameters": {
        "QueryString": "SELECT * FROM \"athena-sample-project-db-wJalrXUtnFEMI\".\"log
\" limit 1",
        "WorkGroup": "stepfunctions-athena-sample-project-workgroup-wJalrXUtnFEMI"
      },
      "Type": "Task",
      "Next": "Get query results"
    },
    "Get query results": {
      "Resource": "arn:aws:states:::athena:getQueryResults",
      "Parameters": {
        "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
      },
      "Type": "Task",
      "Next": "Send query results"
    },
    "Send query results": {
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:StepFunctionsSample-
AthenaDataQueryd1111-2222-3333-777788889999-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "$.ResultSet.Rows"
```

```
        }
      },
      "Type": "Task",
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athena-LambdaForDataGeneration-AKIAIOSFODNN7EXAMPLE",
                "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athen-LambdaForInvokingCrawler-AKIAI44QH8DHBEXAMPLE"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:111122223333:StepFunctionsSample-
AthenaDataQueryd1111-2222-3333-777788889999-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Effect": "Allow"
        },
        {
```

```
            "Action": [
                "athena:getQueryResults",
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:111122223333:workgroup/stepfunctions-athena-
sample-project-workgroup-wJalrXUtnFEMI",
                "arn:aws:athena:us-east-1:111122223333:datacatalog/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
```

```
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:111122223333:database/*",
                "arn:aws:glue:us-east-1:111122223333:table/*",
                "arn:aws:glue:us-east-1:111122223333:catalog"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

複数のクエリを実行します (Amazon Athena、Amazon SNS)

このサンプルプロジェクトでは、Athena クエリを連続して実行した後、並列的にエラーを処理し、その後、クエリの成否に基づいて Amazon SNS 通知を送信する方法のデモを行います。

このプロジェクトでは、Step Functions は、ステートマシンを使用して、Athena クエリを同期的に呼び出します。クエリ結果が返された後、2 つの Athena クエリを並列実行して並列状態を入力します。その後、ジョブが成功または失敗するまで待機し、ジョブの成功または失敗に関するメッセージを含む Amazon SNS トピックを送信します。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

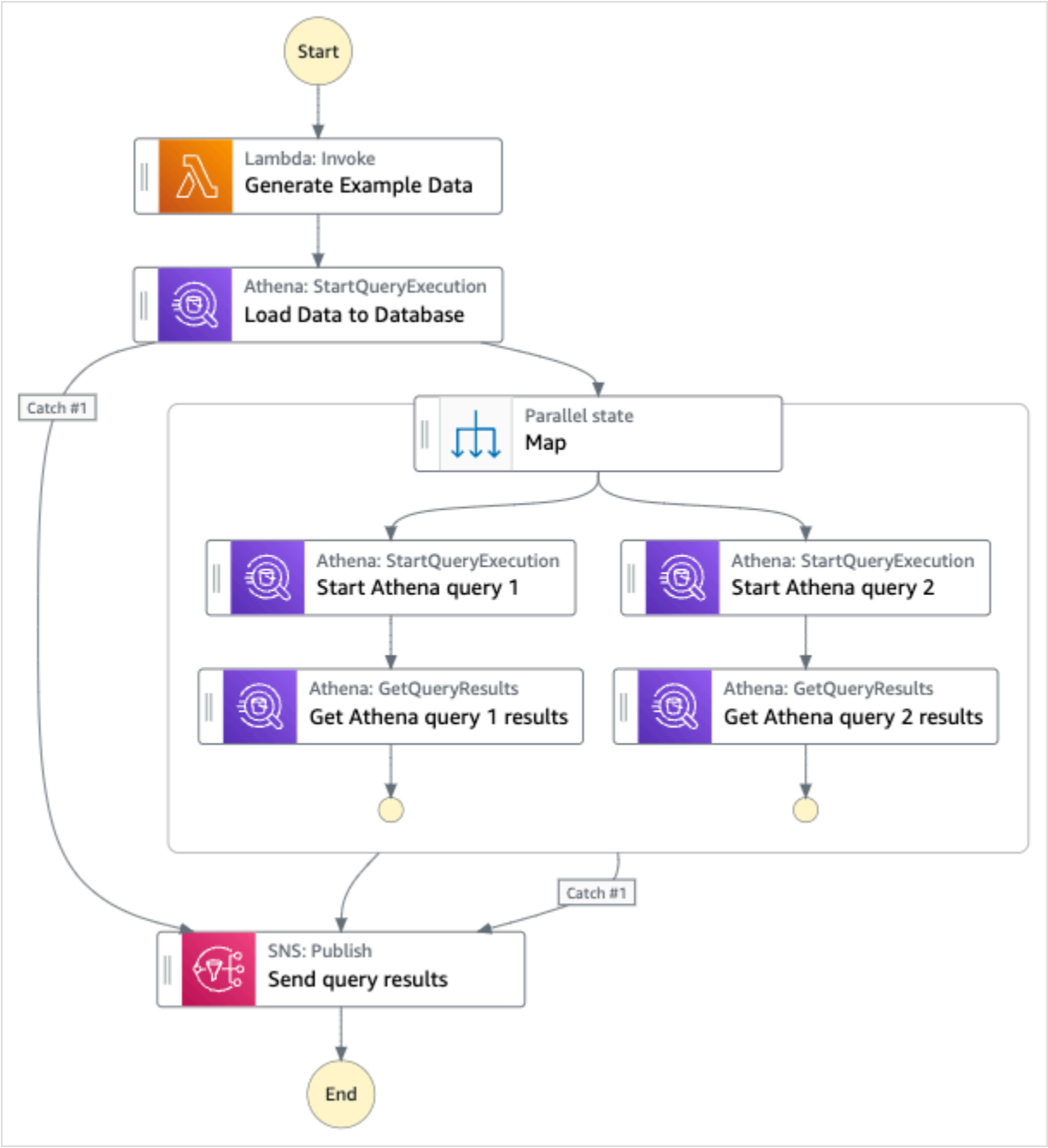
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Execute multiple queries`** と入力し、返された検索結果から [複数のクエリを実行] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築

するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

このサンプルプロジェクトは、以下のリソースをデプロイします。

- Amazon Athena クエリ
- Amazon SNS トピック
- AWS Step Functions ステートマシン
- 関連する AWS Identity and Access Management (IAM) ロール

以下のイメージは、[複数のクエリを実行] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS

Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

    1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

        > **Note**
        >
        > Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

    2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

        [デモの実行] を選択した場合、実行入力を入力する必要はありません。

    3. [実行のスタート] を選択します。
    4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

        実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルオブジェクトのステートマシンは、パラメータを直接リソースに渡すことで、Amazon Athena および Amazon SNS と統合します。

このステートマシンの例を参照して、`Resource` フィールドの Amazon リソースネーム (ARN) に接続し、`Parameters` をサービス API に渡すことで、Step Functions が Amazon Athena と Amazon SNS を制御する方法を確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "Comment": "An example of using Athena to execute queries in sequence and parallel,
 with error handling and notifications.",
    "StartAt": "Generate Example Data",
    "States": {
      "Generate Example Data": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
          "FunctionName": "<ATHENA_FUNCTION_NAME>"
        },
        "Next": "Load Data to Database"
      },
      "Load Data to Database": {
        "Type": "Task",
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
          "QueryString": "<ATHENA_QUERYSTRING>",
          "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Catch": [
          {
            "ErrorEquals": [
              "States.ALL"
            ],
            "Next": "Send query results"
          }
        ],
        "Next": "Map"
      },
      "Map": {
        "Type": "Parallel",
        "ResultSelector": {
          "Query1Result.$": "$[0].ResultSet.Rows",
          "Query2Result.$": "$[1].ResultSet.Rows"
        },
        "Catch": [
          {
            "ErrorEquals": [
```

```
              "States.ALL"
            ],
            "Next": "Send query results"
          }
        ],
        "Branches": [
          {
            "StartAt": "Start Athena query 1",
            "States": {
              "Start Athena query 1": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
                "Parameters": {
                  "QueryString": "<ATHENA_QUERYSTRING>",
                  "WorkGroup": "<ATHENA_WORKGROUP>"
                },
                "Next": "Get Athena query 1 results"
              },
              "Get Athena query 1 results": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:getQueryResults",
                "Parameters": {
                  "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
                },
                "End": true
              }
            }
          },
          {
            "StartAt": "Start Athena query 2",
            "States": {
              "Start Athena query 2": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
                "Parameters": {
                    "QueryString": "<ATHENA_QUERYSTRING>",
                    "WorkGroup": "<ATHENA_WORKGROUP>"
                },
                "Next": "Get Athena query 2 results"
              },
              "Get Athena query 2 results": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:getQueryResults",
                "Parameters": {
```

```
                  "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
                },
                "End": true
              }
            }
          }
        ],
        "Next": "Send query results"
      },
      "Send query results": {
        "Type": "Task",
        "Resource": "arn:aws:states:::sns:publish",
        "Parameters": {
          "Message.$": "$",
          "TopicArn": "<SNS_TOPIC_ARN>"
        },
        "End": true
      }
    }
  }
```

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

AthenaStartQueryExecution

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
```

```
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-ztuvu9yuix",
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
```

```
            "Resource": [
                "arn:aws:glue:us-east-2:123456789012:catalog",
                "arn:aws:glue:us-east-2:123456789012:database/*",
                "arn:aws:glue:us-east-2:123456789012:table/*",
                "arn:aws:glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AthenaGetQueryResults

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:getQueryResults"
            ],
            "Resource": [
                "arn:aws:us-east-2:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        }
```

```
    ]
}
```

SNSPublish

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-
AthenaMultipleQueriese1ec229b-5cbe-4754-a8a8-078474bac878-SNSTopic-9AID0HEJT7TH"
            ]
        }
    ]
}
```

LambdaInvokeFunction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-2:123456789012:function:StepFunctionsSample-
Athen-LambdaForStringGeneratio-GQFQjN7mE9gl:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
```

```
            "Resource": [
                "arn:aws:lambda:us-east-2:123456789012:function:StepFunctionsSample-
Athen-LambdaForStringGeneratio-GQFQjN7mE9gl"
            ]
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

大規模なデータセット (Amazon Athena、Amazon S3 AWS Glue、Amazon SNS) へのクエリ

このサンプルプロジェクトは、Amazon S3 に大量のデータセットを取り込み、 AWS Glue クローラーを使用してパーティション化し、そのパーティションに対して Amazon Athena クエリを実行する方法を示しています。

このプロジェクトでは、Step Functions ステートマシンが Amazon S3 AWS Glue の大きなデータセットを分割するクローラーを呼び出します。 AWS Glue クローラーが成功メッセージを返すと、ワークフローはそのパーティションに対して Athena クエリを実行します。クエリの実行が正常に完了すると、Amazon SNS 通知が Amazon SNS トピックに送信されます。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

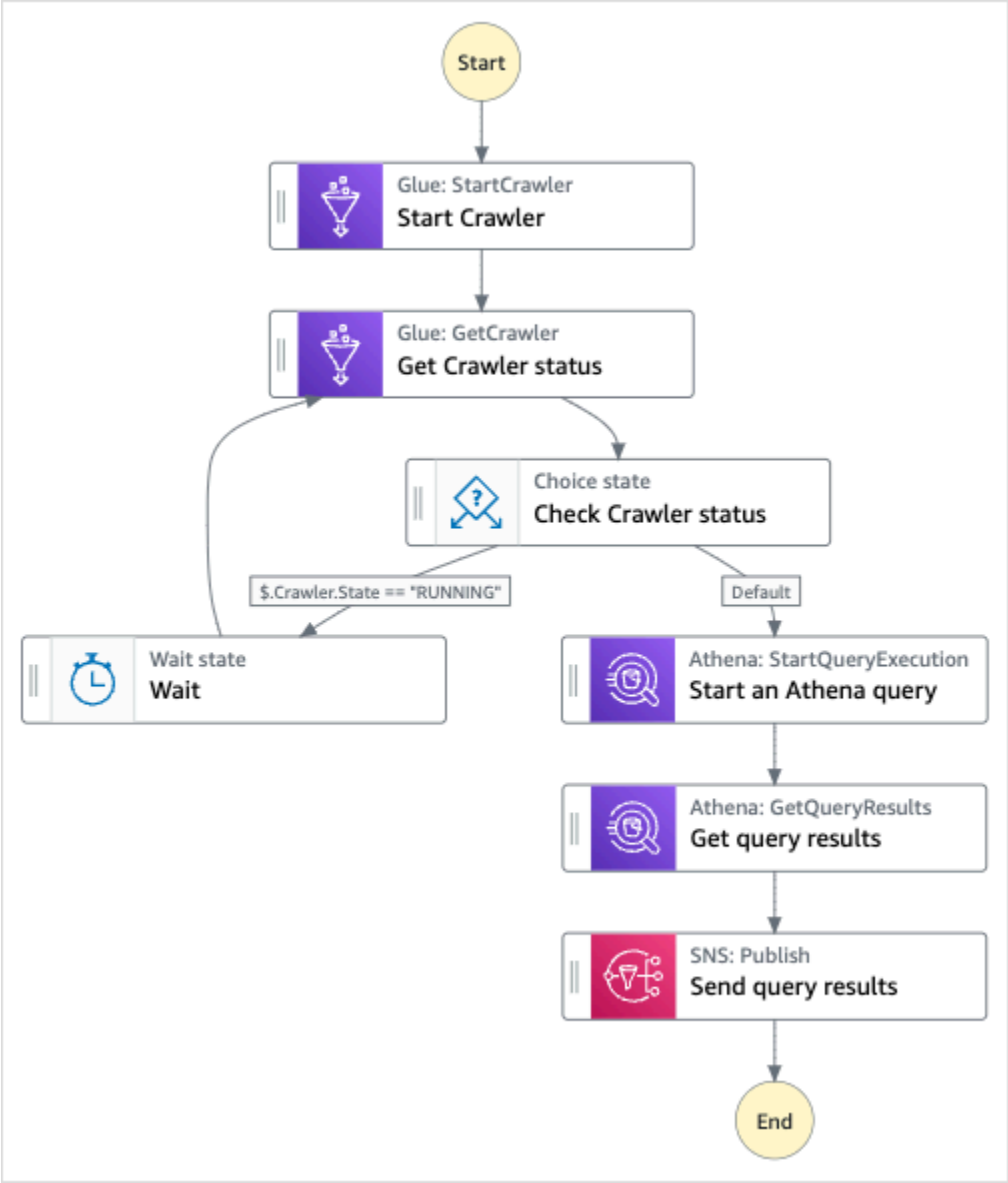
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Query large datasets`** と入力し、返された検索結果から [大規模なデータセットをクエリ] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon S3 バケット

- AWS Glue クローラー
- Amazon SNS トピック
- AWS Step Functions ステートマシン
- 関連する AWS Identity and Access Management (IAM) ロール

以下のイメージは、[大規模なデータセットをクエリ] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

    1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

        > **Note**
        >
        > Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

    2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

        [デモの実行] を選択した場合、実行入力を入力する必要はありません。

    3. [実行のスタート] を選択します。
    4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

        実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをこれらのリソースに直接渡すことにより AWS Glue、Amazon S3、Amazon Athena、および Amazon SNS と統合されます。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し AWS Glue、サービス API `Parameters` に渡すことによって Amazon S3、Amazon Athena、および Amazon SNS を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "Comment": "An example demonstrates how to ingest a large data set in Amazon S3 and
 partition it through aws Glue Crawlers, then execute Amazon Athena queries against
 that partition.",
   "StartAt": "Start Crawler",
   "States": {
     "Start Crawler": {
       "Type": "Task",
       "Next": "Get Crawler status",
       "Parameters": {
         "Name": "<GLUE_CRAWLER_NAME>"
       },
       "Resource": "arn:aws:states:::aws-sdk:glue:startCrawler"
     },
     "Get Crawler status": {
       "Type": "Task",
       "Parameters": {
         "Name": "<GLUE_CRAWLER_NAME>"
       },
       "Resource": "arn:aws:arn:aws:states:::aws-sdk:glue:getCrawler",
       "Next": "Check Crawler status"
     },
     "Check Crawler status": {
       "Type": "Choice",
       "Choices": [
         {
           "Variable": "$.Crawler.State",
           "StringEquals": "RUNNING",
           "Next": "Wait"
         }
       ],
```

```
        "Default": "Start an Athena query"
      },
      "Wait": {
        "Type": "Wait",
        "Seconds": 30,
        "Next": "Get Crawler status"
      },
      "Start an Athena query": {
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
            "QueryString": "<ATHENA_QUERYSTRING>",
            "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Type": "Task",
        "Next": "Get query results"
      },
      "Get query results": {
        "Resource": "arn:aws:states:::athena:getQueryResults",
        "Parameters": {
          "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
        },
        "Type": "Task",
        "Next": "Send query results"
      },
      "Send query results": {
        "Resource": "arn:aws:states:::sns:publish",
        "Parameters": {
          "TopicArn": "<SNS_TOPIC_ARN>",
          "Message": {
            "Input.$": "$.ResultSet.Rows"
          }
        },
        "Type": "Task",
        "End": true
      }
    }
  }
```

IAM の例

サンプルプロジェクトによって生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

AthenaGetQueryResults

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:getQueryResults"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        }
    ]
}
```

AthenaStartQueryExecution

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-8v7bshiv70",
```

```
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-east-2:123456789012:catalog",
```

```
                "arn:aws:glue:us-east-2:123456789012:database/*",
                "arn:aws:glue:us-east-2:123456789012:table/*",
                "arn:aws:glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

SNSPublish

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-
AthenaIngestLargeDataset92bc4949-abf8-4a1e-9236-5b7c81b3efa3-SNSTopic-8Y5ZLI5AASXV"
            ]
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

データを最新の状態に保つ (Amazon Athena、Amazon S3、 AWS Glue)

このサンプルプロジェクトでは、ターゲットテーブルにクエリを実行して AWS Glue Catalog で現在のデータを取得し、Amazon Athena を使用して他のソースからの新しいデータでそのテーブルを更新する方法を示します。

このプロジェクトでは、Step Functions ステートマシンが AWS Glue Catalog を呼び出して、ターゲットテーブルが Amazon S3 バケットに存在するかどうかを確認します。テーブルが見つからない場合は、新しいテーブルが作成されます。次に、Step Functions は Athena クエリを実行して、別のデータソースからターゲットテーブルに行を追加します。まず、ターゲットテーブルをクエリして最新の日付を取得し、ソーステーブルに最新のデータがないかとクエリしてターゲットテーブルに挿入します。

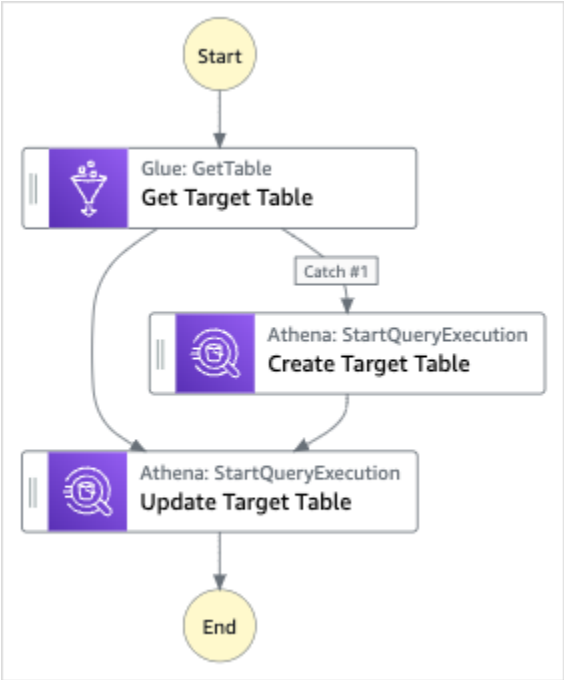
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Keep data up to date`** と入力し、返された検索結果から [データを最新の状態に保つ] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon S3 バケット
   - Amazon Athena クエリ
   - AWS Glue Data Catalog コール
   - AWS Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[データを最新の状態に保つ] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

   - [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。
    1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名はAmazonでは機能しませ

ん。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、Amazon S3、 AWS Glue、および Amazon Athena をこれらのリソースに直接渡すことで統合されます。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し AWS Glue、サービス API `Parameters` に渡すことによって Amazon S3 と Amazon Athena を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
    "Comment": "An example demonstrates how to use Athena to query a target table to
 get current data, then update it with new data from other sources.",
    "StartAt": "Get Target Table",
    "States": {
      "Get Target Table": {
        "Type": "Task",
        "Parameters": {
          "DatabaseName": "<GLUE_DATABASE_NAME>",
          "Name": "target"
        },
        "Catch": [
```

```
          {
            "ErrorEquals": [
              "Glue.EntityNotFoundException"
            ],
            "Next": "Create Target Table"
          }
        ],
        "Resource": "arn:aws:states:::aws-sdk:glue:getTable",
        "Next": "Update Target Table"
      },
      "Create Target Table": {
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
            "QueryString": "<ATHENA_QUERYSTRING>",
            "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Type": "Task",
        "Next": "Update Target Table"
      },
      "Update Target Table": {
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
            "QueryString": "<ATHENA_QUERYSTRING>",
            "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Type": "Task",
        "End": true
      }
    }
  }
```

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

AthenaStartQueryExecution

```
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-26ujlyawxg",
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
```

```
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws::glue:us-east-2:123456789012:catalog",
                "arn:aws::glue:us-east-2:123456789012:database/*",
                "arn:aws::glue:us-east-2:123456789012:table/*",
                "arn:aws::glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Amazon EKS クラスターの管理

このサンプルプロジェクトでは、Step Functions と Amazon Elastic Kubernetes Service を使用して、ノードグループを持つ Amazon EKS クラスターを作成し、Amazon EKS でジョブを実行し、出力を調べる方法を示します。終了すると、ノードグループと Amazon EKS クラスターが削除されます。

Step Functions と Step Functions サービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Step Functions で Amazon EKS を呼び出す

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用してもー定のレベル以下であれば無料です。 AWS 費用と無料利用枠の詳細については、「Amazon EKS 料金表」を参照してください。

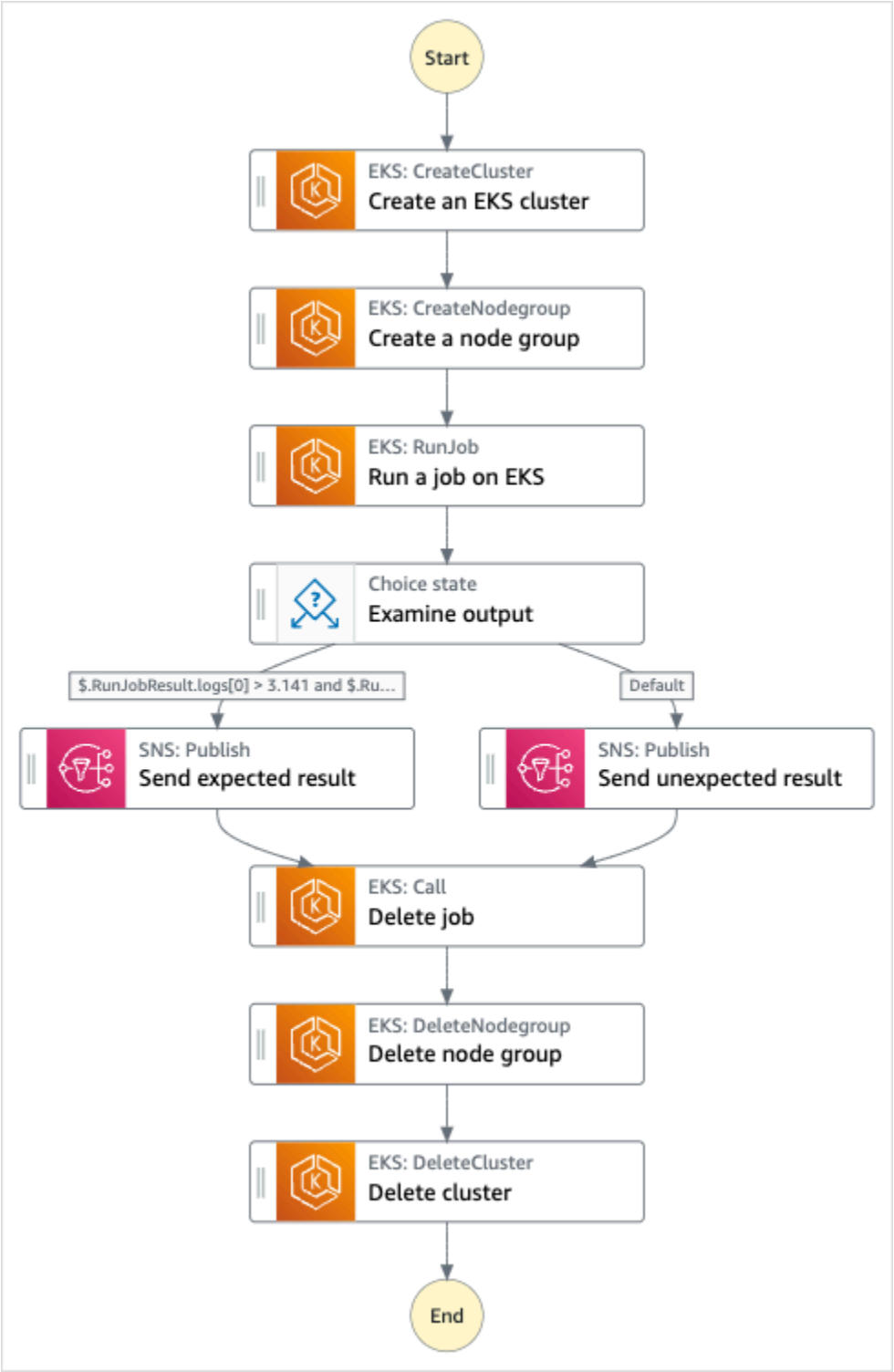
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Manage an EKS cluster`** と入力し、返された検索結果から [EMR クラスターを管理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon Elastic Kubernetes Service クラスター
   - Amazon SNS トピック
   - AWS Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[EKS クラスターを管理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、Amazon EKS クラスターとノードグループを作成して Amazon EKS と統合し、SNS トピックを使用して結果を返します。

このステートマシンの例を参照して、Step Functions が Amazon EKS クラスターとノードグループを管理する方法を確認してください。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "An example of the Amazon States Language for running Amazon EKS Cluster",
  "StartAt": "Create an EKS cluster",
  "States": {
    "Create an EKS cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster",
        "ResourcesVpcConfig": {
          "SubnetIds": [
            "subnet-0aacf887d9f00e6a7",
            "subnet-0e5fc41e7507194ab"
          ]
        },
        "RoleArn": "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
      },
      "Retry": [{
        "ErrorEquals": [ "States.ALL" ],
        "IntervalSeconds": 30,
        "MaxAttempts": 2,
        "BackoffRate": 2
      }],
      "ResultPath": "$.eks",
```

```
      "Next": "Create a node group"
    },
    "Create a node group": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createNodegroup.sync",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "NodegroupName": "ExampleNodegroup",
        "NodeRole": "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterMan-
NodeInstanceRole-ANPAJ2UCCR6DPCEXAMPLE",
        "Subnets": [
          "subnet-0aacf887d9f00e6a7",
          "subnet-0e5fc41e7507194ab"]
      },
      "Retry": [{
        "ErrorEquals": [ "States.ALL" ],
        "IntervalSeconds": 30,
        "MaxAttempts": 2,
        "BackoffRate": 2
      }],
      "ResultPath": "$.nodegroup",
      "Next": "Run a job on EKS"
    },
    "Run a job on EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:runJob.sync",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "CertificateAuthority.$": "$.eks.Cluster.CertificateAuthority.Data",
        "Endpoint.$": "$.eks.Cluster.Endpoint",
        "LogOptions": {
          "RetrieveLogs": true
        },
        "Job": {
          "apiVersion": "batch/v1",
          "kind": "Job",
          "metadata": {
            "name": "example-job"
          },
          "spec": {
            "backoffLimit": 0,
            "template": {
              "metadata": {
                "name": "example-job"
```

```
            },
            "spec": {
              "containers": [
                {
                  "name": "pi-20",
                  "image": "perl",
                  "command": [
                    "perl"
                  ],
                  "args": [
                    "-Mbignum=bpi",
                    "-wle",
                    "print '{ ' . '\"pi\": '. bpi(20) . ' }';"
                  ]
                }
              ],
              "restartPolicy": "Never"
            }
          }
        }
      }
    },
    "ResultSelector": {
      "status.$": "$.status",
      "logs.$": "$.logs..pi"
    },
    "ResultPath": "$.RunJobResult",
    "Next": "Examine output"
  },
  "Examine output": {
    "Type": "Choice",
    "Choices": [
      {
        "And": [
          {
            "Variable": "$.RunJobResult.logs[0]",
            "NumericGreaterThan": 3.141
          },
          {
            "Variable": "$.RunJobResult.logs[0]",
            "NumericLessThan": 3.142
          }
        ],
        "Next": "Send expected result"
```

```
        }
      ],
      "Default": "Send unexpected result"
    },
    "Send expected result": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "States.Format('Saw expected value for pi: {}',
 $.RunJobResult.logs[0])"
        }
      },
      "ResultPath": "$.SNSResult",
      "Next": "Delete job"
    },
    "Send unexpected result": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "States.Format('Saw unexpected value for pi: {}',
 $.RunJobResult.logs[0])"
        }
      },
      "ResultPath": "$.SNSResult",
      "Next": "Delete job"
    },
    "Delete job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "CertificateAuthority.$": "$.eks.Cluster.CertificateAuthority.Data",
        "Endpoint.$": "$.eks.Cluster.Endpoint",
        "Method": "DELETE",
        "Path": "/apis/batch/v1/namespaces/default/jobs/example-job"
      },
      "ResultSelector": {
        "status.$": "$.ResponseBody.status"
```

```
      },
      "ResultPath": "$.DeleteJobResult",
      "Next": "Delete node group"
    },
    "Delete node group": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteNodegroup.sync",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "NodegroupName": "ExampleNodegroup"
      },
      "Next": "Delete cluster"
    },
    "Delete cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster"
      },
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー](#)。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:CreateCluster"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:DeleteCluster"
            ],
            "Resource": "arn:aws:eks:sa-east-1:111122223333:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE"
            ]
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー]。

API Gateway を呼び出す

このサンプルプロジェクトでは、Step Functions を使用して API Gateway を呼び出して、呼び出しが成功したかどうかをチェックする方法を示します。

API Gateway と Step Functions サービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Step Functions を使用して API Gateway を呼び出し

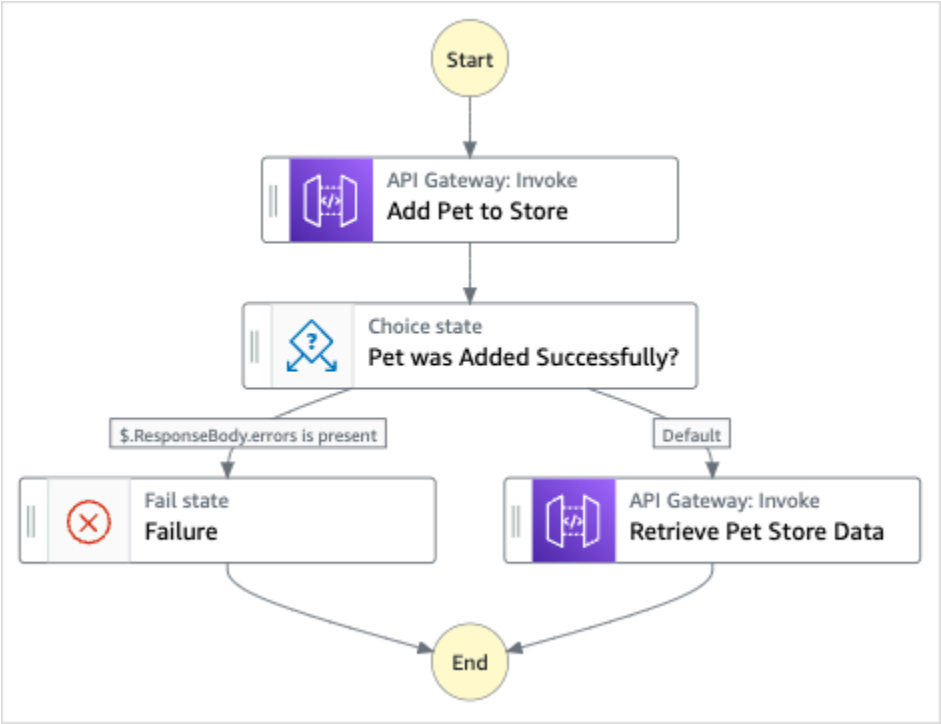
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Make a call to API Gateway`** と入力し、返された検索結果から [API Gateway への呼び出しを行う] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - ステートマシンによって呼び出される Amazon API Gateway REST API。
   - AWS Step Functions ステートマシン
   - 関連する AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[API Gateway への呼び出しを行う] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

   - [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。
   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能し

ません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルオブジェクト内のステートマシンは、API Gateway REST API を呼び出して、必要なパラメータを渡すことで API Gateway と統合します。

このステートマシンの例を参照して、Step Functions が API Gateway とどのように相互作用するかを確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "Calling APIGW REST Endpoint",
  "StartAt": "Add Pet to Store",
  "States": {
    "Add Pet to Store": {
      "Type": "Task",
```

```
      "Resource": "arn:aws:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<POST_PETS_API_ENDPOINT>",
        "Method": "POST",
        "Stage": "default",
        "Path": "pets",
        "RequestBody.$": "$.NewPet",
        "AuthType": "IAM_ROLE"
      },
      "ResultSelector": {
        "ResponseBody.$": "$.ResponseBody"
      },
      "Next": "Pet was Added Successfully?"
    },
    "Pet was Added Successfully?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.ResponseBody.errors",
          "IsPresent": true,
          "Next": "Failure"
        }
      ],
      "Default": "Retrieve Pet Store Data"
    },
    "Failure": {
      "Type": "Fail"
    },
    "Retrieve Pet Store Data": {
      "Type": "Task",
      "Resource": "arn:aws:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<GET_PETS_API_ENDPOINT>",
        "Method": "GET",
        "Stage": "default",
        "Path": "pets",
        "AuthType": "IAM_ROLE"
      },
      "ResultSelector": {
        "Pets.$": "$.ResponseBody"
      },
      "ResultPath": "$.ExistingPets",
      "End": true
    }
```

```
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-west-1:111122223333:c8hqe4kdg5/default/GET/
pets",
                "arn:aws:execute-api:us-west-1:111122223333:c8hqe4kdg5/default/POST/
pets"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

API Gateway 統合を使用して Fargate で実行されているマイクロサービスを呼び出す

このサンプルプロジェクトでは、Step Functions を使用して API Gateway を呼び出し、サービスを操作する方法と AWS Fargate、呼び出しが成功したかどうかを確認する方法を示しています。

API Gateway と Step Functions サービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Step Functions を使用して API Gateway を呼び出し

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「料金表」を参照してください。

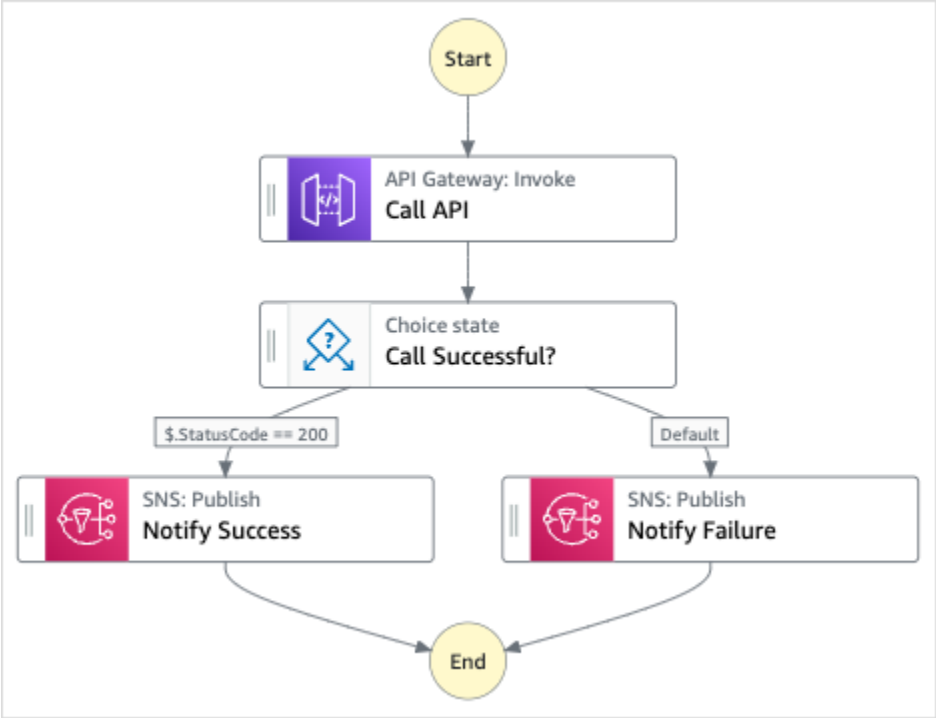
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Call a microservice with API Gateway`** と入力し、返された検索結果から [API Gateway を使用してマイクロサービスを呼び出す] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - ステートマシンによって呼び出される Amazon API Gateway HTTP API。
   - Amazon API Gateway Amazon VPCリンク。
   - Amazon Virtual Private Cloud。
   - Application Load Balancer。
   - Fargate クラスター。
   - Amazon SNS トピック
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール
   - これらのリソースの連携を有効にするために必要な複数の追加サービス。

以下のイメージは、[API Gateway を使用してマイクロサービスを呼び出す] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、Fargate 上のサービスに接続されている API Gateway HTTP API を呼び出すことにより、API Gateway と統合します。これはプライベートサブネットでホストされ、プライベートのApplication Load Balancerを介してアクセスされます。

このステートマシンの例を参照して、Step Functions が API Gateway と対話し結果を返す方法を確認します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "Calling APIGW HTTP Endpoint",
  "StartAt": "Call API",
  "States": {
    "Call API": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<API_ENDPOINT>",
        "Method": "GET",
        "AuthType": "IAM_ROLE"
      },
      "Next": "Call Successful?"
    },
    "Call Successful?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.StatusCode",
          "NumericEquals": 200,
          "Next": "Notify Success"
        }
      ],
      "Default": "Notify Failure"
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Call was successful",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Call was not successful",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    }
```

```
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:111122223333:apigw-ecs-sample-2000-
SNSTopic-444455556666"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:444444444444/*/GET/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "ec2:AttachNetworkInterface",
```

```
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:Describe*",
                "ec2:DetachNetworkInterface",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:Describe*",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

カスタムイベントの送信先 EventBridge

このサンプルプロジェクトは、Step Functions を使用して、複数のターゲット (Amazon、Amazon 簡易通知サービス、Amazon Simple Notification Service EventBridge AWS Lambda、Amazon Simple

Queue Service) を持つルールに一致するカスタムイベントをイベントバスに送信する方法を示しています。

Step Functions と Step Functions サービス統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Step EventBridge Functions による呼び出し

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、「EventBridge 料金表」を参照してください。

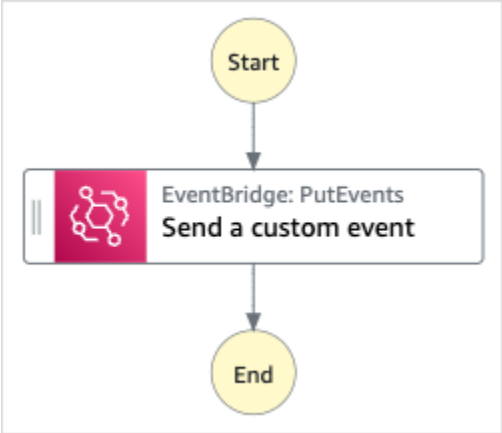
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Send a custom event to EventBridge`** と入力し、返された検索結果から [EventBridge にカスタムイベントを送信] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon EventBridge イベント
   - Amazon SNS トピック
   - Amazon SQS キュー
   - Lambda 関数
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[EventBridge にカスタムイベントを送信] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

   - [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。
    1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能し

ません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、 EventBridge カスタムイベントをイベントバスに送信することでと統合されます。 EventBridge イベントバスに送信されるイベントは、Amazon SNS トピックと Amazon SQS キューにメッセージを送信する Lambda EventBridge 関数をトリガーするルールと一致します。

このステートマシンの例を参照して、Step Functions がどのように管理されるかを確認してください EventBridge。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
  "Comment": "An example of the Amazon States Language for sending a custom event to
 Amazon EventBridge",
  "StartAt": "Send a custom event",
```

```
  "States": {
    "Send a custom event": {
      "Resource": "arn:<PARTITION>:states:::events:putEvents",
      "Type": "Task",
      "Parameters": {
        "Entries": [{
          "Detail": {
            "Message": "Hello from Step Functions!"
          },
          "DetailType": "MessageFromStepFunctions",
          "EventBusName": "<EVENT_BUS_NAME>",
          "Source": "my.statemachine"
        }]
      },
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー]。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:1234567890:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
```

```
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

同期 Express ワークフローを呼び出す

このサンプルプロジェクトでは、Amazon API Gateway を通じて同期 Express ワークフローを呼び出して、従業員データベースを管理する方法について説明します。

このプロジェクトでは、Step Functions は API Gateway エンドポイントを使用して Step Functions 同期 Express ワークフローをスタートします。次に、DynamoDB を使用して、従業員データベース内の従業員を検索、追加、および削除します。

Step Functions の同期 Express ワークフローの詳細については、同期および非同期 Express ワークフロー を参照してください。

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS コストと無料利用枠の詳細については、「Step Functions 料金表」を参照してください。

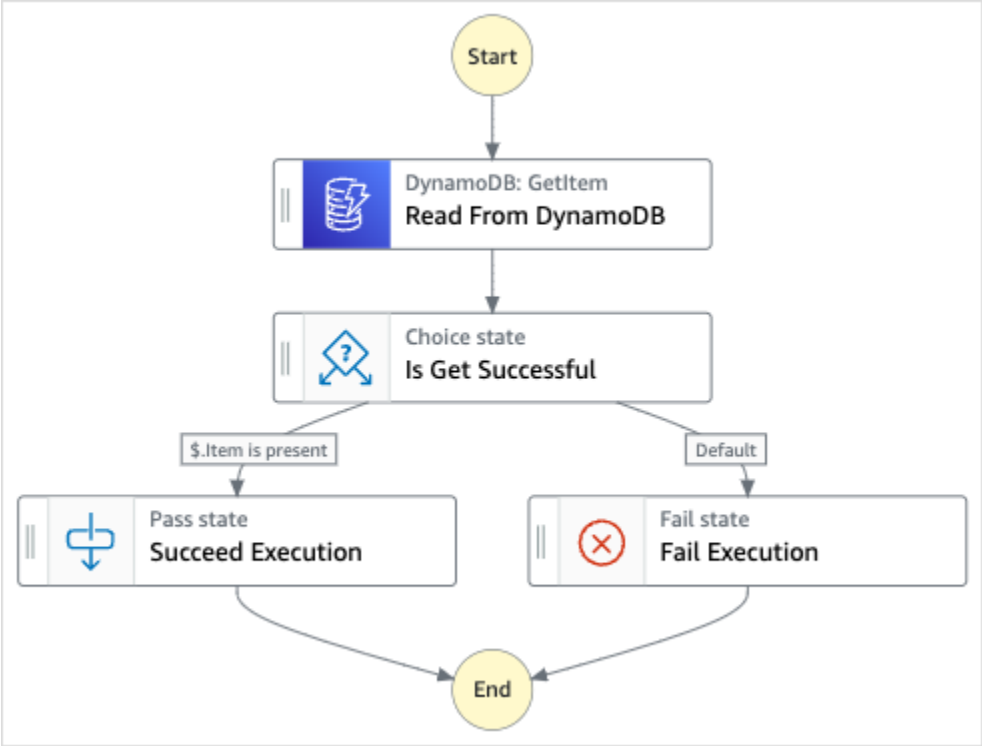
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Invoke Synchronous Express Workflows through API Gateway`** と入力し、返された検索結果から [API Gateway を介して同期高速ワークフローを呼び出す] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

このサンプルプロジェクトは、以下のリソースをデプロイします。

- ステートマシンによって呼び出される Amazon API Gateway HTTPS API。
- Amazon DynamoDB テーブル。
- 3 AWS Step Functions 台のステートマシン。
- 関連 AWS Identity and Access Management (IAM) ロール。

以下のイメージは、[API Gateway を介して同期高速ワークフローを呼び出す] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions

コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。

2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、API Gateway を使用して同期 Express ワークフローを呼び出して API Gateway および DynamoDB と統合し、DynamoDB を使用して従業員データベースを更新または読み取ります。

このステートマシンの例を参照して、Step Functions が DynamoDB から読み取りを行い、従業員情報を取得する方法を確認してください。

API Gateway を使用してStep Functions を呼び出す方法の詳細については、以下を参照してください。

- Step Functions を使用して API Gateway を呼び出し
- API Gateway デベロッパーガイドのプライベートゲートウェイを呼び出す方法

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "This state machine returns an employee entry from DynamoDB",
  "StartAt": "Read From DynamoDB",
  "States": {
    "Read From DynamoDB": {
      "Type": "Task",
      "Resource": "arn:aws:states:::dynamodb:getItem",
      "Parameters": {
        "TableName": "StepFunctionsSample-
SynchronousExpressWorkflowAKIAIOSFODNN7EXAMPLE-DynamoDBTable-ANPAJ2UCCR6DPCEXAMPLE",
        "Key": {
          "EmployeeId": {"S.$": "$.employee"}
        }
      },
      "Retry": [
        {
          "ErrorEquals": [
            "DynamoDB.AmazonDynamoDBException"
          ],
          "IntervalSeconds": 3,
          "MaxAttempts": 2,
          "BackoffRate": 1.5
        }
      ],
      "Next": "Is Get Successful"
    },
    "Is Get Successful": {
      "Type": "Choice",
      "Choices": [
        {
```

```
          "Variable": "$.Item",
          "IsPresent": true,
          "Next": "Succeed Execution"
        }
      ],
      "Default": "Fail Execution"
    },
    "Succeed Execution": {
      "Type": "Pass",
      "Parameters" : {
        "employee.$": "$.Item.EmployeeId.S",
        "jobTitle.$": "$.Item.JobTitle.S"
      },
      "End": true
    },
    "Fail Execution": {
      "Type": "Fail",
      "Error": "EmployeeDoesNotExist"
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
```

```
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/Write"
            ]
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Amazon Redshift (Lambda、Amazon Redshift データ API) を使用して ETL/ELT ワークフローを実行する

このサンプルプロジェクトでは、Step Functions と Amazon Redshift データ API を使用して、Amazon Redshift データウェアハウスにデータをロードする ETL/ELT ワークフローを実行する方法を示します。

このプロジェクトでは、Step Functions AWS Lambda は関数と Amazon Redshift Data API を使用して必要なデータベースオブジェクトを作成し、一連のサンプルデータを生成します。次に、ディメンションテーブルの読み込みとそれに続くファクトテーブルの読み込みを行う 2 つのジョブをparallel

実行します。両方のディメンションロードジョブが正常に終了すると、Step Functions はファクトテーブルのロードジョブを実行し、検証ジョブを実行し、Amazon Redshift クラスターを一時停止します。

> **Note**
>
> ETL ロジックを変更して、Amazon S3 などの他のソースからデータを受信できます。[COPY] (コピー) コマンドを使用して、Amazon S3 から Amazon Redshift テーブルにデータをコピーします。

Amazon Redshift とStep Functions サービスの統合の詳細については、以下を参照してください。

- AWS Step Functions 他のサービスとの併用
- Amazon Redshift Data API の使用
- Amazon Redshift データ API サービス
- Lambda を使用する Step Functions ステートマシン状態の作成
- 以下の IAM ポリシー:
  - の IAM ポリシー AWS Lambda
  - Amazon Redshift Data API へのアクセスの認可

> **Note**
>
> このサンプルプロジェクトでは、料金が発生する場合があります。
> AWS 新規ユーザーには、無料利用枠が用意されています。この枠では、サービスを利用しても一定のレベル以下であれば無料です。 AWS 費用と無料利用枠について詳しくは、AWS Step Functions 料金をご覧ください。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

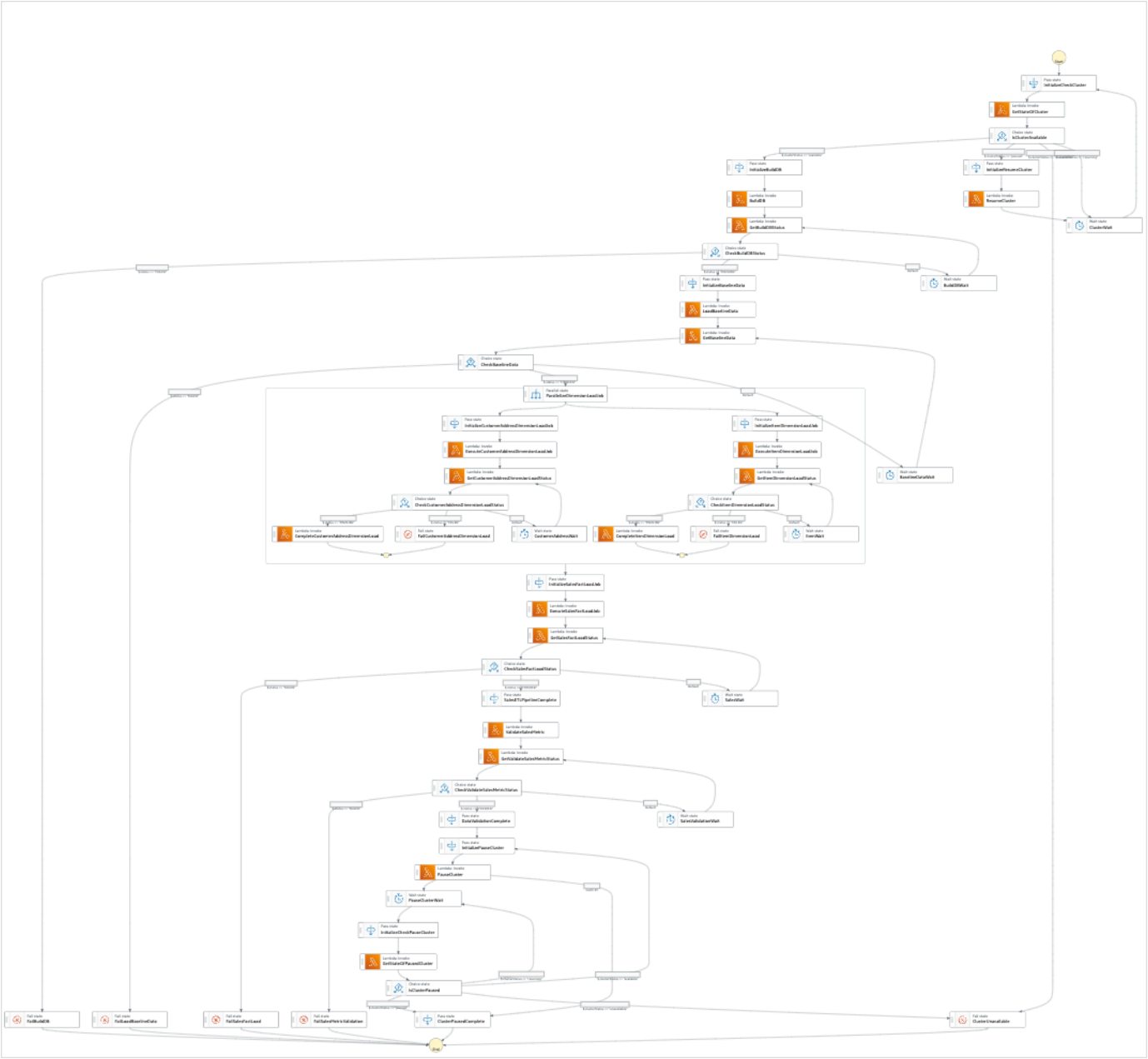
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`ETL job in Amazon Redshift`** と入力し、返された検索結果から [Amazon Redshift の ETL ジョブ] を選択します。
3. [次へ] を選択して続行します。

4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトをデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Amazon Redshift クラスター
   - 2 つの Lambda 関数
   - Amazon Redshift スキーマ
   - 5 つの Amazon Redshift テーブル
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール。

   以下のイメージは、[Amazon Redshift の ETL ジョブ] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions

コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。

2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

      > **Note**
      >
      > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

   3. [実行のスタート] を選択します。
   4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

      実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、ETL InputPath ロジックをそれらのリソースに直接渡し、Amazon Redshift Data API AWS Lambda を使用して非同期で実行することでと統合されます。

このステートマシンの例を参照して、Step Functions が Amazon Redshift データ API AWS Lambda をどのように制御するかを確認してください。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してくださいAWS Step Functions 他のサービスとの併用。

```
{
"Comment": "A simple ETL workflow for loading dimension and fact tables",
  "StartAt": "InitializeCheckCluster",
  "States": {
    "InitializeCheckCluster": {
      "Type": "Pass",
      "Next": "GetStateOfCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "status"
        }
      }
    },
    "GetStateOfCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "IsClusterAvailable",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus"
    },
    "IsClusterAvailable": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "available",
          "Next": "InitializeBuildDB"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "paused",
          "Next": "InitializeResumeCluster"
        },
```

```
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "unavailable",
          "Next": "ClusterUnavailable"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "resuming",
          "Next": "ClusterWait"
        }
      ]
    },
    "ClusterWait": {
      "Type": "Wait",
      "Seconds": 720,
      "Next": "InitializeCheckCluster"
    },
    "InitializeResumeCluster": {
      "Type": "Pass",
      "Next": "ResumeCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "resume"
        }
      }
    },
    "ResumeCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "ClusterWait",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "InitializeBuildDB": {
      "Type": "Pass",
      "Next": "BuildDB",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
```

```
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "action": "build_database",
          "sql_statement": [
            "create schema if not exists {0} authorization {1};",
            "create table if not exists {0}.customer",
            "(c_customer_sk          int4     not null encode az64",
            ",c_customer_id          char(16) not null encode zstd",
            ",c_current_addr_sk      int4              encode az64",
            ",c_first_name           char(20)          encode zstd",
            ",c_last_name            char(30)          encode zstd",
            ",primary key (c_customer_sk)",
            ") distkey(c_customer_sk);",
            "--",
            "create table if not exists {0}.customer_address",
            "(ca_address_sk    int4     not null encode az64",
            ",ca_address_id    char(16) not null encode zstd",
            ",ca_state         char(2)           encode zstd",
            ",ca_zip           char(10)          encode zstd",
            ",ca_country       varchar(20)       encode zstd",
            ",primary key (ca_address_sk)",
            ") distkey(ca_address_sk);",
            "--",
            "create table if not exists {0}.date_dim",
            "(d_date_sk           integer  not null encode az64",
            ",d_date_id           char(16) not null encode zstd",
            ",d_date              date              encode az64",
            ",d_day_name          char(9)           encode zstd",
            ",primary key (d_date_sk)",
            ") diststyle all;",
            "--",
            "create table if not exists {0}.item",
            "(i_item_sk        int4     not null encode az64",
            ",i_item_id        char(16) not null encode zstd",
            ",i_rec_start_date date              encode az64",
            ",i_rec_end_date   date              encode az64",
            ",i_current_price  numeric(7,2)      encode az64",
            ",i_category       char(50)          encode zstd",
            ",i_product_name   char(50)          encode zstd",
            ",primary key (i_item_sk)",
            ") distkey(i_item_sk) sortkey(i_category);",
            "--",
            "create table if not exists {0}.store_sales",
            "(ss_sold_date_sk       int4",
```

```
            ",ss_item_sk            int4 not null encode az64",
            ",ss_customer_sk        int4          encode az64",
            ",ss_addr_sk            int4          encode az64",
            ",ss_store_sk           int4          encode az64",
            ",ss_ticket_number      int8 not null encode az64",
            ",ss_quantity           int4          encode az64",
            ",ss_net_paid           numeric(7,2)  encode az64",
            ",ss_net_profit         numeric(7,2)  encode az64",
            ",primary key (ss_item_sk, ss_ticket_number)",
            ") distkey(ss_item_sk) sortkey(ss_sold_date_sk);"
          ]
        }
      }
    },
    "BuildDB": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetBuildDBStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetBuildDBStatus": {
      "Type": "Task",
      "Next": "CheckBuildDBStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckBuildDBStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailBuildDB"
        },
        {
          "Variable": "$.status",
```

```
          "StringEquals": "FINISHED",
          "Next": "InitializeBaselineData"
        }
      ],
      "Default": "BuildDBWait"
    },
    "BuildDBWait": {
      "Type": "Wait",
      "Seconds": 15,
      "Next": "GetBuildDBStatus"
    },
    "FailBuildDB": {
      "Type": "Fail",
      "Cause": "Database Build Failed",
      "Error": "Error"
    },
    "InitializeBaselineData": {
      "Type": "Pass",
      "Next": "LoadBaselineData",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "action": "load_baseline_data",
          "sql_statement": [
            "begin transaction;",
            "truncate table {0}.customer;",
            "insert into {0}.customer
(c_customer_sk,c_customer_id,c_current_addr_sk,c_first_name,c_last_name)",
            "values",
            "(7550,'AAAAAAAAOHNBAAAA',9264662,'Michelle','Deaton'),",
            "(37079,'AAAAAAAAHNAJAAAA',13971208,'Michael','Simms'),",
            "(40626,'AAAAAAAACLOJAAAA',1959255,'Susan','Ryder'),",
            "(2142876,'AAAAAAAAMJCLACAA',7644556,'Justin','Brown');",
            "analyze {0}.customer;",
            "--",
            "truncate table {0}.customer_address;",
            "insert into {0}.customer_address
(ca_address_sk,ca_address_id,ca_state,ca_zip,ca_country)",
            "values",
            "(13971208,'AAAAAAAAIAPCFNAA','NE','63451','United States'),",
            "(7644556,'AAAAAAAAMIFKEHAA','SD','58883','United States'),",
```

```
            "(9264662,'AAAAAAAAGBOFNIAA','CA','99310','United States');",
            "analyze {0}.customer_address;",
            "--",
            "truncate table {0}.item;",
            "insert into {0}.item
(i_item_sk,i_item_id,i_rec_start_date,i_rec_end_date,i_current_price,i_category,i_product_name
            "values",

"(3417,'AAAAAAAAIFNAAAAA','1997-10-27',NULL,14.29,'Electronics','ationoughtesepri
'),",
            "(9615,'AAAAAAAAOIFCAAAA','1997-10-27',NULL,9.68,'Home','antioughtcallyn
st'),",
            "(3693,'AAAAAAAAMGOAAAAA','2001-03-12',NULL,2.10,'Men','prin
stcallypri'),",

"(3630,'AAAAAAAAMCOAAAAA','2001-10-27',NULL,2.95,'Electronics','barpricallypri'),",

"(16506,'AAAAAAAAIHAEAAAA','2001-10-27',NULL,3.85,'Home','callybaranticallyought'),",

"(7866,'AAAAAAAAILOBAAAA','2001-10-27',NULL,12.60,'Jewelry','callycallyeingation');",
            "--",
            "analyze {0}.item;",
            "truncate table {0}.date_dim;",
            "insert into {0}.date_dim (d_date_sk,d_date_id,d_date,d_day_name)",
            "values",
            "(2450521,'AAAAAAAAJFEGFCAA','1997-03-13','Thursday'),",
            "(2450749,'AAAAAAAANDFGFCAA','1997-10-27','Monday'),",
            "(2451251,'AAAAAAAADDHGFCAA','1999-03-13','Saturday'),",
            "(2451252,'AAAAAAAAEDHGFCAA','1999-03-14','Sunday'),",
            "(2451981,'AAAAAAAANAKGFCAA','2001-03-12','Monday'),",
            "(2451982,'AAAAAAAAOAKGFCAA','2001-03-13','Tuesday'),",
            "(2452210,'AAAAAAAACPKGFCAA','2001-10-27','Saturday'),",
            "(2452641,'AAAAAAAABKMGFCAA','2003-01-01','Wednesday'),",
            "(2452642,'AAAAAAAACKMGFCAA','2003-01-02','Thursday');",
            "--",
            "analyze {0}.date_dim;",
            "--  commit and End transaction",
            "commit;",
            "end transaction;"
          ]
        }
      }
    },
    "LoadBaselineData": {
```

```
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetBaselineData",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetBaselineData": {
      "Type": "Task",
      "Next": "CheckBaselineData",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckBaselineData": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailLoadBaselineData"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "ParallelizeDimensionLoadJob"
        }
      ],
      "Default": "BaselineDataWait"
    },
    "BaselineDataWait": {
      "Type": "Wait",
      "Seconds": 20,
      "Next": "GetBaselineData"
    },
    "FailLoadBaselineData": {
      "Type": "Fail",
      "Cause": "Load Baseline Data Failed",
      "Error": "Error"
```

```
    },
    "ParallelizeDimensionLoadJob": {
      "Type": "Parallel",
      "Next": "InitializeSalesFactLoadJob",
      "ResultPath": "$.status",
      "Branches": [
        {
          "StartAt": "InitializeCustomerAddressDimensionLoadJob",
          "States": {
            "InitializeCustomerAddressDimensionLoadJob": {
              "Type": "Pass",
              "Next": "ExecuteCustomerAddressDimensionLoadJob",
              "Result": {
                "input": {
                  "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
                  "redshift_database": "dev",
                  "redshift_user": "awsuser",
                  "redshift_schema": "tpcds",
                  "action": "load_customer_address",
                  "sql_statement": [
                    "begin transaction;",
                    "/* Create a staging table to hold the input data. Staging table is
created with BACKUP NO option for faster inserts and also data temporary */",
                    "drop table if exists {0}.stg_customer_address;",
                    "create table if not exists {0}.stg_customer_address",
                    "(ca_address_id    varchar(16)  encode zstd",
                    ",ca_state         varchar(2)   encode zstd",
                    ",ca_zip           varchar(10)  encode zstd",
                    ",ca_country       varchar(20)  encode zstd",
                    ")",
                    "backup no",
                    "diststyle even;",
                    "/* Ingest data from source */",
                    "insert into {0}.stg_customer_address
(ca_address_id,ca_state,ca_zip,ca_country)",
                    "values",
                    "('AAAAAAAACFBBAAAA','NE','','United States'),",
                    "('AAAAAAAAGAEFAAAA','NE','61749','United States'),",
                    "('AAAAAAAAPJKKAAAA','OK','','United States'),",
                    "('AAAAAAAAMIHGAAAA','AL','','United States');",
                    "/* Perform UPDATE for existing data with refreshed attribute
values */",
                    "update {0}.customer_address",
                    "   set ca_state = stg_customer_address.ca_state,",
```

```
                    "       ca_zip = stg_customer_address.ca_zip,",
                    "       ca_country = stg_customer_address.ca_country",
                    "  from {0}.stg_customer_address",
                    " where customer_address.ca_address_id =
 stg_customer_address.ca_address_id;",
                    "/* Perform insert for new rows  */",
                    "insert into {0}.customer_address",
                    "(ca_address_sk",
                    ",ca_address_id",
                    ",ca_state",
                    ",ca_zip",
                    ",ca_country",
                    ")",
                    "with max_customer_address_sk as",
                    "(select max(ca_address_sk) max_ca_address_sk",
                    "from {0}.customer_address)",
                    "select row_number() over (order by
 stg_customer_address.ca_address_id) + max_customer_address_sk.max_ca_address_sk as
 ca_address_sk",
                    ",stg_customer_address.ca_address_id",
                    ",stg_customer_address.ca_state",
                    ",stg_customer_address.ca_zip",
                    ",stg_customer_address.ca_country",
                    "from {0}.stg_customer_address,",
                    "max_customer_address_sk",
                    "where stg_customer_address.ca_address_id not in (select
 customer_address.ca_address_id from {0}.customer_address);",
                    "/* Commit and End transaction */",
                    "commit;",
                    "end transaction;"
                ]
              }
            }
          },
          "ExecuteCustomerAddressDimensionLoadJob": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
            "TimeoutSeconds": 180,
            "HeartbeatSeconds": 60,
            "Next": "GetCustomerAddressDimensionLoadStatus",
            "InputPath": "$",
            "ResultPath": "$"
          },
```

```
            "GetCustomerAddressDimensionLoadStatus": {
              "Type": "Task",
              "Next": "CheckCustomerAddressDimensionLoadStatus",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "InputPath": "$",
              "ResultPath": "$.status"
            },
            "CheckCustomerAddressDimensionLoadStatus": {
              "Type": "Choice",
              "Choices": [
                {
                  "Variable": "$.status",
                  "StringEquals": "FAILED",
                  "Next": "FailCustomerAddressDimensionLoad"
                },
                {
                  "Variable": "$.status",
                  "StringEquals": "FINISHED",
                  "Next": "CompleteCustomerAddressDimensionLoad"
                }
              ],
              "Default": "CustomerAddressWait"
            },
            "CustomerAddressWait": {
              "Type": "Wait",
              "Seconds": 5,
              "Next": "GetCustomerAddressDimensionLoadStatus"
            },
            "CompleteCustomerAddressDimensionLoad": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "End": true
            },
            "FailCustomerAddressDimensionLoad": {
              "Type": "Fail",
              "Cause": "ETL Workflow Failed",
              "Error": "Error"
            }
```

```
          }
        },
        {
          "StartAt": "InitializeItemDimensionLoadJob",
          "States": {
            "InitializeItemDimensionLoadJob": {
              "Type": "Pass",
              "Next": "ExecuteItemDimensionLoadJob",
              "Result": {
                "input": {
                  "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
                  "redshift_database": "dev",
                  "redshift_user": "awsuser",
                  "redshift_schema": "tpcds",
                  "action": "load_item",
                  "sql_statement": [
                    "begin transaction;",
                    "/* Create a staging table to hold the input data. Staging table is
created with BACKUP NO option for faster inserts and also data temporary */",
                    "drop table if exists {0}.stg_item;",
                    "create table if not exists {0}.stg_item",
                    "(i_item_id        varchar(16) encode zstd",
                    ",i_rec_start_date date encode zstd",
                    ",i_rec_end_date   date encode zstd",
                    ",i_current_price  numeric(7,2) encode zstd",
                    ",i_category       varchar(50) encode zstd",
                    ",i_product_name   varchar(50) encode zstd",
                    ")",
                    "backup no",
                    "diststyle even;",
                    "/* Ingest data from source */",
                    "insert into {0}.stg_item",

"(i_item_id,i_rec_start_date,i_rec_end_date,i_current_price,i_category,i_product_name)",
                    "values",

"('AAAAAAAAABJBAAAA','2000-10-27',NULL,4.10,'Books','ationoughtesecally'),",
                    "('AAAAAAAAOPKBAAAA','2001-10-27',NULL,4.22,'Books','ableoughtn
stcally'),",
                    "('AAAAAAAAHGPAAAAA','1997-10-27',NULL,29.30,'Books','priesen
stpri'),",

"('AAAAAAAAICMAAAAA','2001-10-27',NULL,1.93,'Books','eseoughtoughtpri'),",
```

```
 "('AAAAAAAAGPGBAAAA','2001-10-27',NULL,9.96,'Books','bareingeinganti'),",
                    "('AAAAAAAANBEBAAAA','1997-10-27',NULL,2.25,'Music','n
 steseoughtanti'),",

 "('AAAAAAAACLAAAAAA','2001-10-27',NULL,1.71,'Home','bareingought'),",

 "('AAAAAAAAOBBDAAAA','2001-10-27',NULL,5.55,'Books','callyationantiableought');",
                    "/
*********************************************************************************************
                    "** Type 2 is maintained for i_current_price column.",
                    "** Update all attributes for the item when the price is not
 changed",
                    "** Sunset existing active item record with current i_rec_end_date
 and insert a new record when the price does not match",

 "*********************************************************************************************
                    "update {0}.item",
                    "   set i_category = stg_item.i_category,",
                    "       i_product_name = stg_item.i_product_name",
                    "  from {0}.stg_item",
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price = stg_item.i_current_price;",
                    "insert into {0}.item",
                    "(i_item_sk",
                    ",i_item_id",
                    ",i_rec_start_date",
                    ",i_rec_end_date",
                    ",i_current_price",
                    ",i_category",
                    ",i_product_name",
                    ")",
                    "with max_item_sk as",
                    "(select max(i_item_sk) max_item_sk",
                    "   from {0}.item)",
                    "select row_number() over (order by stg_item.i_item_id) +
 max_item_sk as i_item_sk",
                    "      ,stg_item.i_item_id",
                    "      ,trunc(sysdate) as i_rec_start_date",
                    "      ,null as i_rec_end_date",
                    "      ,stg_item.i_current_price",
                    "      ,stg_item.i_category",
                    "      ,stg_item.i_product_name",
```

```
                    "  from {0}.stg_item, {0}.item, max_item_sk",
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price <> stg_item.i_current_price;",
                    "/* Sunset penultimate records that were inserted as type 2 */",
                    "update {0}.item",
                    "   set i_rec_end_date = trunc(sysdate)",
                    "  from {0}.stg_item",
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price <> stg_item.i_current_price;",
                    "/* Commit and End transaction */",
                    "commit;",
                    "end transaction;"
                  ]
                }
              }
            },
            "ExecuteItemDimensionLoadJob": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "Next": "GetItemDimensionLoadStatus",
              "InputPath": "$",
              "ResultPath": "$"
            },
            "GetItemDimensionLoadStatus": {
              "Type": "Task",
              "Next": "CheckItemDimensionLoadStatus",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "InputPath": "$",
              "ResultPath": "$.status"
            },
            "CheckItemDimensionLoadStatus": {
              "Type": "Choice",
              "Choices": [
                {
                  "Variable": "$.status",
                  "StringEquals": "FAILED",
```

```
                  "Next": "FailItemDimensionLoad"
                },
                {
                  "Variable": "$.status",
                  "StringEquals": "FINISHED",
                  "Next": "CompleteItemDimensionLoad"
                }
              ],
              "Default": "ItemWait"
            },
            "ItemWait": {
              "Type": "Wait",
              "Seconds": 5,
              "Next": "GetItemDimensionLoadStatus"
            },
            "CompleteItemDimensionLoad": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "End": true
            },
            "FailItemDimensionLoad": {
              "Type": "Fail",
              "Cause": "ETL Workflow Failed",
              "Error": "Error"
            }
          }
        }
      ]
    },
    "InitializeSalesFactLoadJob": {
      "Type": "Pass",
      "Next": "ExecuteSalesFactLoadJob",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "snapshot_date": "2003-01-02",
          "action": "load_sales_fact",
          "sql_statement": [
```

```
            "begin transaction;",
            "/* Create a stg_store_sales staging table */",
            "drop table if exists {0}.stg_store_sales;",
            "create table {0}.stg_store_sales",
            "(sold_date             date encode zstd",
            ",i_item_id             varchar(16) encode zstd",
            ",c_customer_id         varchar(16) encode zstd",
            ",ca_address_id         varchar(16) encode zstd",
            ",ss_ticket_number      integer encode zstd",
            ",ss_quantity           integer encode zstd",
            ",ss_net_paid           numeric(7,2) encode zstd",
            ",ss_net_profit         numeric(7,2) encode zstd",
            ")",
            "backup no",
            "diststyle even;",
            "/* Ingest data from source */",
            "insert into {0}.stg_store_sales",

"(sold_date,i_item_id,c_customer_id,ca_address_id,ss_ticket_number,ss_quantity,ss_net_paid,ss_
            "values",

"('2003-01-02','AAAAAAAAIFNAAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,5046.37,150

"('2003-01-02','AAAAAAAAIFNAAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,2103.72,-12

"('2003-01-02','AAAAAAAAILOBAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,959.10,-130

"('2003-01-02','AAAAAAAAILOBAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1403191,13,962.65,-475

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1201746,17,111.60,-241

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1201746,17,4013.02,-11

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAMJCLACAA','AAAAAAAAMIFKEHAA',1201746,17,2689.12,-55

"('2003-01-02','AAAAAAAAMGOAAAAA','AAAAAAAAMJCLACAA','AAAAAAAAMIFKEHAA',193971,18,1876.89,-556
            "/* Delete any rows from target store_sales for the input date for
idempotency */",
            "delete from {0}.store_sales where ss_sold_date_sk in (select d_date_sk
from {0}.date_dim where d_date='{1}');",
            "/* Insert data from staging table to the target table */",
            "insert into {0}.store_sales",
            "(ss_sold_date_sk",
            ",ss_item_sk",
```

```
            ",ss_customer_sk",
            ",ss_addr_sk",
            ",ss_ticket_number",
            ",ss_quantity",
            ",ss_net_paid",
            ",ss_net_profit",
            ")",
            "select date_dim.d_date_sk ss_sold_date_sk",
            "      ,item.i_item_sk ss_item_sk",
            "      ,customer.c_customer_sk ss_customer_sk",
            "      ,customer_address.ca_address_sk ss_addr_sk",
            "      ,ss_ticket_number",
            "      ,ss_quantity",
            "      ,ss_net_paid",
            "      ,ss_net_profit",
            "  from {0}.stg_store_sales as store_sales",
            "  inner join {0}.date_dim on store_sales.sold_date = date_dim.d_date",
            "  left join {0}.item on store_sales.i_item_id = item.i_item_id and
 item.i_rec_end_date is null",
            "  left join {0}.customer on store_sales.c_customer_id =
 customer.c_customer_id",
            "  left join {0}.customer_address on store_sales.ca_address_id =
 customer_address.ca_address_id;",
            "/* Drop staging table */",
            "drop table if exists {0}.stg_store_sales;",
            "/* Commit and End transaction */",
            "commit;",
            "end transaction;"
          ]
        }
      }
    },
    "ExecuteSalesFactLoadJob": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetSalesFactLoadStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetSalesFactLoadStatus": {
      "Type": "Task",
```

```
      "Next": "CheckSalesFactLoadStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckSalesFactLoadStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailSalesFactLoad"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "SalesETLPipelineComplete"
        }
      ],
      "Default": "SalesWait"
    },
    "SalesWait": {
      "Type": "Wait",
      "Seconds": 5,
      "Next": "GetSalesFactLoadStatus"
    },
    "FailSalesFactLoad": {
      "Type": "Fail",
      "Cause": "ETL Workflow Failed",
      "Error": "Error"
    },
    "ClusterUnavailable": {
      "Type": "Fail",
      "Cause": "Redshift cluster is not available",
      "Error": "Error"
    },
    "SalesETLPipelineComplete": {
      "Type": "Pass",
      "Next": "ValidateSalesMetric",
      "Result": {
        "input": {
```

```
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "snapshot_date": "2003-01-02",
          "action": "validate_sales_metric",
          "sql_statement": [
            "select 1/count(1) from {0}.store_sales where ss_sold_date_sk in (select
 d_date_sk from {0}.date_dim where d_date='{1}')"
          ]
        }
      }
    },
    "ValidateSalesMetric": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetValidateSalesMetricStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetValidateSalesMetricStatus": {
      "Type": "Task",
      "Next": "CheckValidateSalesMetricStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckValidateSalesMetricStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailSalesMetricValidation"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
```

```
          "Next": "DataValidationComplete"
        }
      ],
      "Default": "SalesValidationWait"
    },
    "SalesValidationWait": {
      "Type": "Wait",
      "Seconds": 5,
      "Next": "GetValidateSalesMetricStatus"
    },
    "FailSalesMetricValidation": {
      "Type": "Fail",
      "Cause": "Data Validation Failed",
      "Error": "Error"
    },
    "DataValidationComplete": {
      "Type": "Pass",
      "Next": "InitializePauseCluster"
    },
    "InitializePauseCluster": {
      "Type": "Pass",
      "Next": "PauseCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "pause"
        }
      }
    },
    "PauseCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "PauseClusterWait",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus",
      "Catch": [
        {
          "ErrorEquals": [
            "States.ALL"
          ],
          "Next": "ClusterPausedComplete"
```

```
        }
      ]
    },
    "InitializeCheckPauseCluster": {
      "Type": "Pass",
      "Next": "GetStateOfPausedCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "status"
        }
      }
    },
    "GetStateOfPausedCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "IsClusterPaused",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus"
    },
    "IsClusterPaused": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "available",
          "Next": "InitializePauseCluster"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "paused",
          "Next": "ClusterPausedComplete"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "unavailable",
          "Next": "ClusterUnavailable"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "resuming",
```

```
          "Next": "PauseClusterWait"
        }
      ]
    },
    "PauseClusterWait": {
      "Type": "Wait",
      "Seconds": 720,
      "Next": "InitializeCheckPauseCluster"
    },
    "ClusterPausedComplete": {
      "Type": "Pass",
      "End": true
    }
  }
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください[統合サービスの IAM ポリシー](#)。

IAM の例

サンプルプロジェクトで生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:111122223333:function:CFN36-RedshiftDataApi-
AIDACKCEVSQ6C2EXAMPLE",
                "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

エラー処理で Step Functions と AWS Batch を使用する

このサンプルプロジェクトでは、Step Functions を使用して、エラー処理機能を備えたステートマシンを使用して AWS Batch ジョブを実行する方法を示します。

このプロジェクトでは、Step Functions は、ステートマシンを使用して、AWS Batch ジョブを同期的に呼び出します。その後、ジョブが成功または失敗するまで待機し、ジョブが成功または失敗するときにメッセージを含む Amazon SNS トピックを送信します。

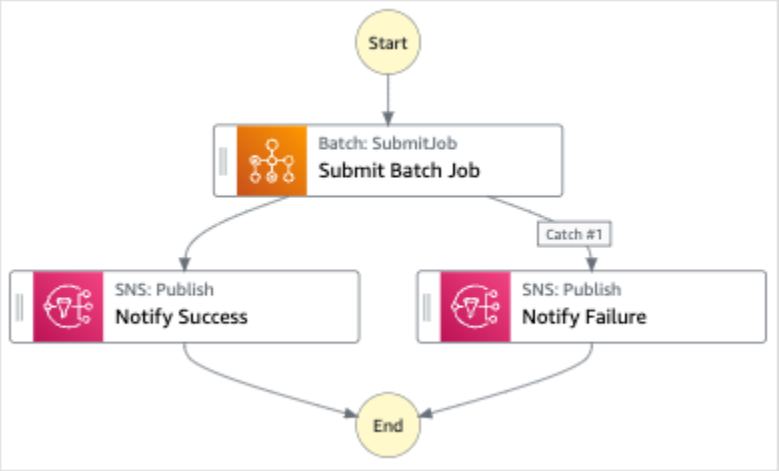
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Manage a batch job`** と入力し、返された検索結果から [バッチジョブの管理] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS Batch ジョブ
   - 1 つの Amazon SNS トピック
   - AWS Step Functions ステートマシン
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[バッチジョブの管理] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

     > **Important**
     >
     > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

   - [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

     > **Tip**
     >
     > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

      > **Note**
      >
      > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

   2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

      [デモの実行] を選択した場合、実行入力を入力する必要はありません。

> **Note**
>
> デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをリソースに直接渡すことにより、Amazon SNS AWS Batch と統合します。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し、サービス API `Parameters` に渡すことによって Amazon SNS AWS Batch を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "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-west-2:123456789012:job-queue/
BatchJobQueue-123456789abcdef",
        "JobDefinition": "arn:aws:batch:us-west-2:123456789012:job-definition/
BatchJobDefinition-123456789abcdef:1"
      },
      "Next": "Notify Success",
      "Retry": [
        {
          "ErrorEquals": [
            "States.ALL"
          ],
          "IntervalSeconds": 30,
          "MaxAttempts": 2,
          "BackoffRate": 1.5
        }
      ],
      "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-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
      },
      "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-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
      },
      "End": true
    }
  }
```

```
}
```

IAM の例

サンプルプロジェクトによって生成されたこのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

Example **BatchJobNotificationAccessPolicy**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
```

```
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

AWS Batch ジョブのファンアウト

このサンプルプロジェクトは、Step Functions マッピング AWS Batch の状態を使用してジョブをファンアウトする方法を示しています。

このプロジェクトでは、Step Functions はステートマシンを使用して Lambda 関数を呼び出して簡単な前処理を行い、 AWS Batch その状態を使用して複数のジョブをparallel 呼び出します。マッピング

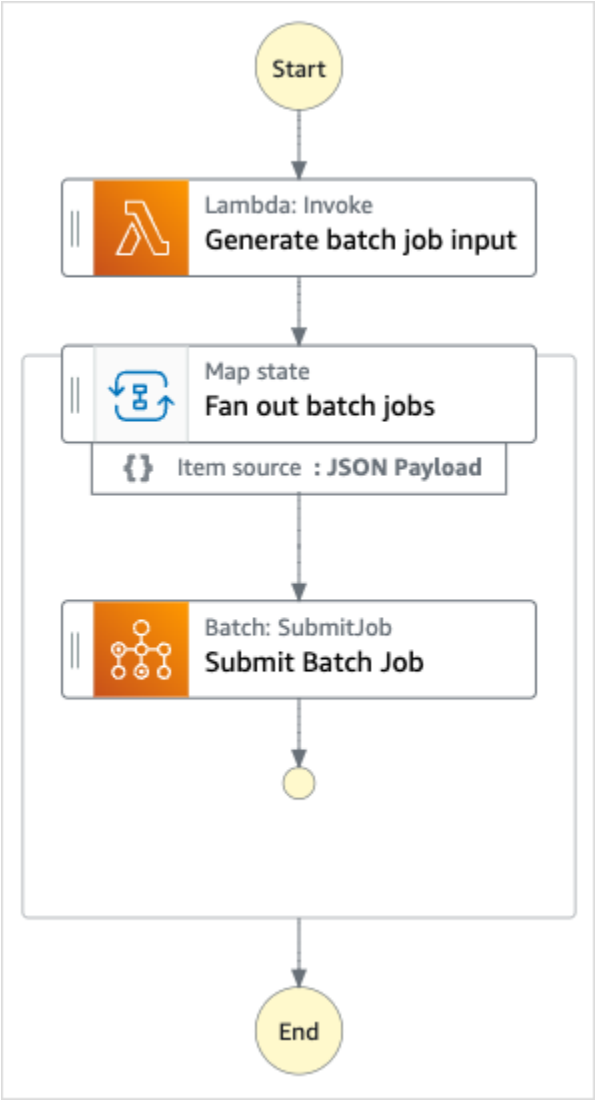
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Fan out a batch job`** と入力し、返された検索結果から [バッチジョブをファンアウトする] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Lambda 関数
   - AWS Batch ジョブキュー
   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下のイメージは、[バッチジョブをファンアウトする] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをリソースに直接渡すことにより、Amazon SNS AWS Batch と統合します。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し、サービス API `Parameters` に渡すことによって Amazon SNS AWS Batch を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "An example of the Amazon States Language for fanning out AWS Batch job",
  "StartAt": "Generate batch job input",
  "TimeoutSeconds": 3600,
  "States": {
    "Generate batch job input": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<GENERATE_BATCH_JOB_INPUT_LAMBDA_FUNCTION_NAME>"
      },
      "Next": "Fan out batch jobs"
    },
    "Fan out batch jobs": {
      "Comment": "Start multiple executions of batch job depending on pre-processed
 data",
      "Type": "Map",
      "End": true,
      "ItemsPath": "$",
      "Parameters": {
        "BatchNumber.$": "$$.Map.Item.Value"
      },
      "Iterator": {
        "StartAt": "Submit Batch Job",
        "States": {
          "Submit Batch Job": {
            "Type": "Task",
            "Resource": "arn:aws:states:::batch:submitJob.sync",
            "Parameters": {
              "JobName": "BatchJobFanOut",
              "JobQueue": "<BATCH_QUEUE_ARN>",
              "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>"
            },
            "End": true
          }
        }
      }
    }
  }
```

```
}
```

IAM の例

サンプルプロジェクトによって生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

Example **BatchJobFanOutAccessPolicy**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Example **InvokeGenerateBatchJobMapLambdaPolicy**

```
{
    "Statement": [
        {
```

```
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:StepFunctionsSample-BatchJobFa-
GenerateBatchJobMap-444455556666",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

AWS Batch Lambda 付き

このサンプルプロジェクトでは、Step Functions AWS Lambda を使用して関数を使用してデータを前処理し、 AWS Batch ジョブをオーケストレーションする方法を示します。

このプロジェクトでは、Step Functions はステートマシンを使用して Lambda 関数を呼び出して、AWS Batch ジョブが送信される前にシンプルな前処理を行います。前のジョブの結果または成功に応じて、複数のジョブが呼び出されることがあります。

ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

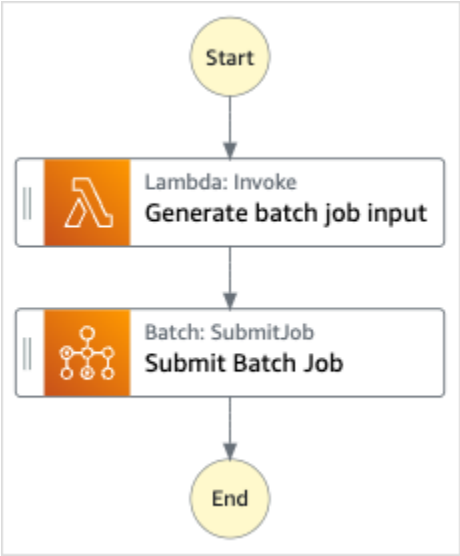
1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`Batch job with Lambda`** と入力し、返された検索結果から [Lambda を使用したバッチジョブ] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - Lambda 関数
   - AWS Batch ジョブ

- AWS Step Functions ステートマシン。
- 関連 AWS Identity and Access Management (IAM) ロール

以下のイメージは、[Lambda を使用したバッチジョブ] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

- [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

  Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

  > **Important**
  >
  > ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

  > **Tip**
  >
  > サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

  準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

  これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

  サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

  > **Important**
  >
  > CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

   1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

> **Note**
>
> Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

   > **Note**
   >
   > デプロイしたデモプロジェクトに事前入力された実行入力データが含まれている場合は、その入力を使用してステートマシンを実行します。

3. [実行のスタート] を選択します。
4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

ステートマシンのコード例

このサンプルプロジェクトのステートマシンは、パラメータをリソースに直接渡すことにより、Amazon SNS AWS Batch と統合します。

このステートマシンの例を見て、Step Functions `Resource` がフィールドの Amazon リソースネーム (ARN) に接続し、サービス API `Parameters` に渡すことによって Amazon SNS AWS Batch を制御する方法を確認してください。

AWS 他のサービスを制御する方法の詳細については AWS Step Functions 、「」を参照してください。AWS Step Functions 他のサービスとの併用

```
{
  "Comment": "An example of the Amazon States Language for using batch job with pre-
processing lambda",
  "StartAt": "Generate batch job input",
  "TimeoutSeconds": 3600,
  "States": {
    "Generate batch job input": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.batch_input",
      "Parameters": {
        "FunctionName": "<GENERATE_BATCH_JOB_INPUT_LAMBDA_FUNCTION_NAME>"
      },
      "Next": "Submit Batch Job"
    },
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobFanOut",
        "JobQueue": "<BATCH_QUEUE_ARN>",
        "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>",
        "Parameters.$": "$.batch_input"
      },
      "End": true
    }
  }
}
```

IAM の例

サンプルプロジェクトによって生成されたこれらのサンプル AWS Identity and Access Management (IAM) ポリシーには、ステートマシンと関連リソースを実行するのに必要な最小限の権限が含まれています。IAM ポリシーで必要なアクセス許可のみを含めることをお勧めします。

Example **BatchJobWithLambdaAccessPolicy**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
```

```
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012: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:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Example **InvokeGenerateBatchJobMapLambdaPolicy**

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:StepFunctionsSample-BatchWithL-
GenerateBatchJobMap-444455556666",
```

```
            "Effect": "Allow"
        }
    ]
}
```

Step Functions AWS を他のサービスで使用する場合の IAM の設定方法については、を参照してください統合サービスの IAM ポリシー。

Amazon Bedrock で AI プロンプトチェーンを実行する

このサンプルプロジェクトでは、Amazon Bedrock と統合して AI プロンプトチェーンを実行する方法を示しています。このサンプルプロジェクトでは、Amazon Bedrock を使用して高品質のチャットボットを構築する方法を示しています。このプロジェクトでは、いくつかのプロンプトを連結し、指定された順序で解決します。これらのプロンプトを連鎖させることで、高度に精選された応答を提供するために使用される言語モデルの能力が強化されます。

このサンプルプロジェクトは、 AWS ステートマシンとサポートリソースを作成し、関連するIAM権限を設定します。このサンプルプロジェクトでは、Step Functions のステートマシンと Amazon Bedrock の最適化されたサービスの統合について学んだり、独自のプロジェクトの出発点として使用する方法について説明します。

トピック

- AWS CloudFormation テンプレートと追加リソース
- 前提条件
- ステップ 1: ステートマシンを作成してリソースをプロビジョニングする
- ステップ 2: ステートマシンを実行する

AWS CloudFormation テンプレートと追加リソース

このサンプルプロジェクトをデプロイするには、CloudFormation テンプレートを使用します。このテンプレートは、以下のリソースをユーザーに作成します。 AWS アカウント

- Step Functions ステートマシン。
- ステートマシンの実行ロール。このロールは、 AWS のサービス Amazon BedrockInvokeModelステートマシンがその他やアクションなどのリソースにアクセスするために必要な権限を付与します。

前提条件

このサンプルプロジェクトでは Cohere Command の大規模言語モデル (LLM) を使用しています。このサンプルプロジェクトを正常に実行するには、Amazon Bedrock コンソールからこの LLM へのアクセスを追加する必要があります。モデルアクセスを追加するには、次の操作を行います。

1. Amazon Bedrock コンソールを開きます。
2. ナビゲーションペインで、[モデルアクセス] を選択します。
3. [モデルアクセスの管理] を選択します。
4. [Cohere] のチェックボックスを選択します。
5. [アクセスをリクエスト] を選択します。[Cohere] モデルの [アクセス状態] には、[アクセスが付与されました] と表示されます。

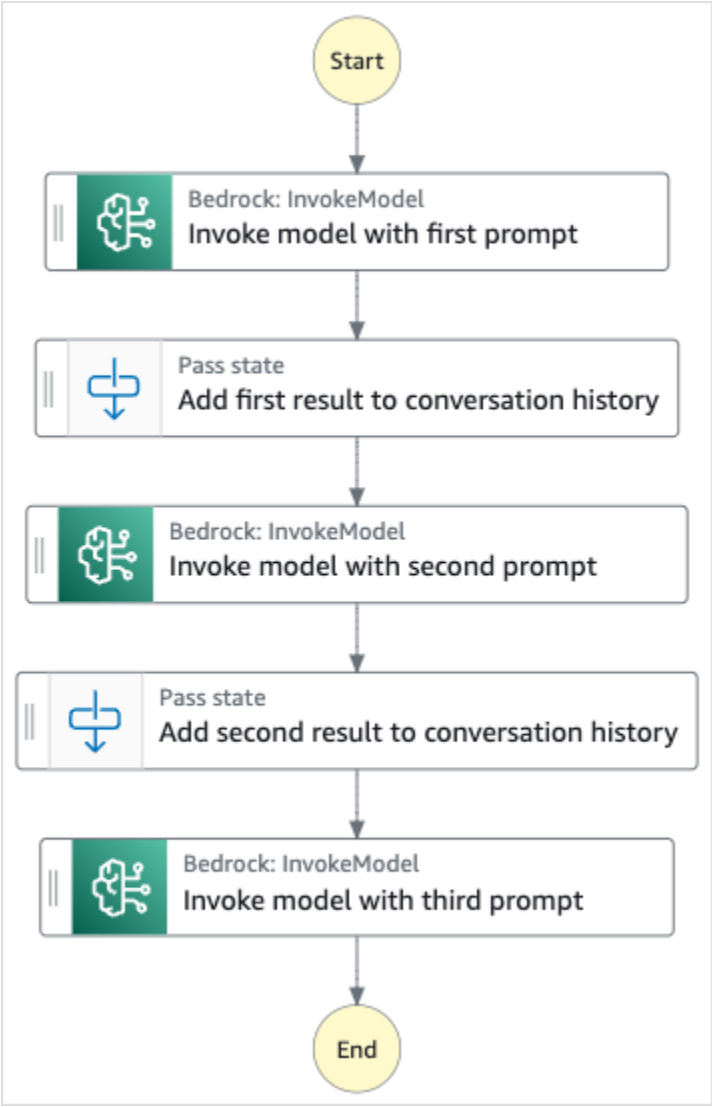
ステップ 1: ステートマシンを作成してリソースをプロビジョニングする

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. 検索ボックスに **`bedrock`** と入力し、返された検索結果から [Bedrock で AI プロンプトチェイニングを実行] を選択します。
3. [次へ] を選択して続行します。
4. Step Functions には、 AWS のサービス 選択したサンプルプロジェクトで使用されているものが一覧表示されます。サンプルプロジェクトのワークフローグラフも表示されます。 AWS アカウント このプロジェクトを自分のプロジェクトにデプロイするか、独自のプロジェクトを構築するための出発点として使用してください。進める方法に応じて、[デモの実行] または [その上に構築する] を選択します。

   このサンプルプロジェクトは、以下のリソースをデプロイします。

   - AWS Step Functions ステートマシン。
   - 関連 AWS Identity and Access Management (IAM) ロール

   以下の図は、[Bedrock で AI プロンプトチェイニングを実行] サンプルプロジェクトのワークフローグラフを示しています。



5. [テンプレートの使用] を選択して選択を続行します。
6. 次のいずれかを行います。

   - [その上に構築する] を選択した場合、Step Functions は選択したサンプルプロジェクトのワークフロープロトタイプを作成します。Step Functions は、ワークフロー定義にリストされているリソースをデプロイしません。

     Workflow Studio の デザインモード では、[State browser] (状態ブラウザ) から状態をドラッグアンドドロップして、ワークフロープロトタイプの構築を続行できます。または、VS Code と同様の統合コードエディタを提供する コードモード に切り替えて、Step Functions コンソール内のステートマシンの Amazon ステートメント言語 (ASL) 定義を更新してください。Workflow Studio を使用してステートマシンを構築する方法の詳細については、「Workflow Studio を使用する」を参照してください。

> **Important**
>
> ワークフローを実行する前に、サンプルプロジェクトで使用されているリソースのプレースホルダー Amazon リソースネーム (ARN) を必ず更新してください。

- [デモの実行] を選択した場合、Step Functions AWS CloudFormation AWS はテンプレートを使用してそのテンプレートにリストされているリソースをにデプロイする読み取り専用のサンプルプロジェクトを作成します。 AWS アカウント

> **Tip**
>
> サンプルプロジェクトのステートマシン定義を表示するには、[コード] を選択します。

準備できたら、[デプロイと実行] を選択してサンプルプロジェクトをデプロイし、リソースを作成します。

これらのリソースおよび関連する IAM 許可が作成されるまで、最大 10 分かかることがあります。リソースのデプロイ中に CloudFormation Stack ID リンクを開いて、どのリソースがプロビジョニングされているかを確認できます。

サンプルプロジェクトのすべてのリソースが作成されると、新しいサンプルプロジェクトが [ステートマシン] ページに表示されます。

> **Important**
>
> CloudFormation テンプレートで使用される各サービスには標準料金が適用される場合があります。

ステップ 2: ステートマシンを実行する

1. [ステートマシン] ページで、サンプルプロジェクトを選択します。
2. サンプルプロジェクトページで、[実行を開始] を選択します。
3. [実行を開始] ダイアログボックスで、以下の操作を行います。

1. (オプション) 実行を識別するには、[名前] ボックスに名前を指定します。デフォルトでは、Step Functions は自動的に一意の実行名を生成します。

   > **Note**
   >
   > Step Functions では、ステートマシン、実行、アクティビティ、ラベルに、ASCII 以外の文字を含む名前を作成できます。これらの非ASCII名は、Amazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

2. (オプション) [入力] ボックスに、JSON 形式の入力値を入力してワークフローを実行します。

   [デモの実行] を選択した場合、実行入力を入力する必要はありません。

3. [実行のスタート] を選択します。

4. Step Functions コンソールから実行 ID のタイトルが付いたページが表示されます。このページは、[実行の詳細] ページと呼ばれます。このページでは、実行の進行中または完了後に実行結果を確認できます。

   実行結果を確認するには、[グラフビュー] で個々の状態を選択し、ステップの詳細 ペインの個々のタブを選択すると、入力、出力、定義などの各状態の詳細がそれぞれ表示されます。[実行の詳細] ページに表示できる実行情報の詳細については、「[実行の詳細] ページ - インターフェイスの概要」を参照してください。

クォータ

AWS Step Functions 特定の期間の API アクションの数や定義できるステートマシンの数など、特定のステートマシンパラメータのサイズにクォータを設定します。これらのクォータは、正しく設定されていないステートマシンによりシステムのすべてのリソースが消費されないようにするために設計されていますが、その多くはハードクォータではありません。

サービスクォータの引き上げをリクエストするには、次のいずれかを行います。

- https://console.aws.amazon.com/servicequotas/home で Service Quotas コンソールを使用します。Service Quotas コンソールを使用したクォータ引き上げのリクエストの詳細については、「Service Quotas ユーザーガイド」の「クォータ引き上げのリクエスト」を参照してください。
- AWS Management Console で提供されるリソースの割り当てを地域ごとに増やすことをリクエストするには、Support センターページを使用してください。 AWS Step Functions 詳細については、「AWS 全般のリファレンス」の「AWS の Service Quotas」を参照してください。

> **Note**
>
> ステートマシン実行の特定のステージまたはアクティビティの実行に長い時間がかかる場合は、タイムアウトイベントを発生させるようにステートマシンのタイムアウトを設定できます。

トピック

- 一般的なクォータ
- アカウントに関連するクォータ
- HTTP タスクに関連するクォータ
- 状態のスロットリングに関連するクォータ
- API アクションのスロットリングに関連するクォータ
- ステートマシンの実行に関連するクォータ
- タスクの実行に関連するクォータ
- バージョンとエイリアスに関連するクォータ
- タグ付けに関連する制限

一般的なクォータ

クォータ	説明
Step Functions の名前	ステートマシン、実行、およびアクティビティタスク名は 80 文字以下にする必要があります。 AWS これらの名前はアカウントと地域で一意である必要があり、次のものを含んではいけません。 • 空白 • ワイルドカード文字 (? *) • 角かっこ (< > { } [ ]) • 特殊文字 (: ; , \ | ^ ~ $ # % & ` ") • 制御文字 (\\u0000 - \\u001f または \\u007f - \\u009f) ステートマシンのタイプが [Express] の場合、ステートマシンを複数回実行する際に同じ名前を指定できます。Step Functions では、複数の実行が同じ名前であっても、Express ステートマシンの実行ごとに一意の実行 ARN を生成します。 Step Functions では、ステートマシン、実行、アクティビティの名前、および ASCII 以外の文字を含むラベルを作成できます。これらの非ASCII名は、Amazonでは機能しません。CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。

アカウントに関連するクォータ

リソース	デフォルトのクォータ	引き上げ可能
登録済みステートマシンの最大数	10,000	25,000
登録済みアクティビティの最大数	10,000	15,000
最大リクエストサイズ	リクエストあたり 1 MB。これは Step Functions API リクエストあたりの合計データサイズで、リクエストヘッダーおよびその他すべての関連リクエストデータを含みます。	ハードクォータ
アカウントあたりの最大オープン実行数	各 AWS リージョンの各 AWS アカウント で 1,000,000 回の実行。これを超えると、`ExecutionLimitExceeded` エラーが発生します。これは Express Workflow には適用されません。	百万
オープンマップ実行の最大数 オープンマップ実行とは、開始されてはいるがまだ完了していないマップランのことです。スケジュールされたマップランは、開いているマップランの合計数がデフォルトのクォータである 1000 MapRunStartedを下回るまでイベントで待機します。	1,000 このクォータは分散マップ状態に適用されます。	ハードクォータ
マップ実行の最大 redrives。	1,000	ハードクォータ

リソース	デフォルトのクォータ	引き上げ可能
	このクォータは分散マップ状態に適用されます。	

HTTP タスクに関連するクォータ

HTTP タスクは、Step Functions サービスの帯域幅を維持するため、トークンのバケットスキームを使用してスロットリングされます。次の表は、HTTP タスクのバケットサイズとリフィルレートを示しています。

リソース	バケットサイズ	補充レート/秒
HTTP タスク	300	300

次の表に HTTP タスクの所要時間のクォータを示します。

リソース	デフォルトのクォータ
HTTP タスクの所要時間 HTTP タスクの所要時間とは、HTTP タスクが HTTP リクエストを送信してレスポンスを受信するまでにかかる時間のことです。	60 秒 これはハードクォータであり、変更できません。

状態のスロットリングに関連するクォータ

Step Functions 状態の移行は、サービス帯域幅を維持するため、トークンのバケットスキームを使用してスロットリングされます。標準ワークフロー と Express ワークフローには、状態遷移スロットリングが異なります。標準ワークフロー はソフトクォータであり、引き上げることができます。

> **Note**
>
> StateTransitionサービスメトリクスのスロットリングは Amazon と同様に報告されますExecutionThrottled。 CloudWatch詳細については、メトリクスをご覧ください。ExecutionThrottled CloudWatch

	Standard		Express	
サービスメトリクス	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
StateTransition — 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)	5,000	5,000	無制限	無制限
StateTransition — その他のすべてのリージョン	800	800	無制限	無制限

API アクションのスロットリングに関連するクォータ

一部の Step Functions API アクションは、サービスの帯域幅を維持するため、トークンのバケットスキームを使用してスロットリングされます。これらのクォータはソフトクォータであり、引き上げることができます。

> **Note**
>
> スロットリングクォータはアカウント、リージョンごとです。 AWS AWS Step Functions バケットサイズとリフィル率の両方がいつでも増加する可能性があります。

	Standard		Express	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
`StartExecution` — 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)	1,300	300	6,000	6,000
`StartExecution` — その他のすべてのリージョン	800	150	6,000	6,000

API に関連するクォータ TestState

次の表は API TestState で利用できるクォータの一覧です。

API 名	クォータ	引き上げ可能
TestState	1 秒あたりのトランザクション (TPS) は 1	ハードクォータ

その他のクォータ

これらのクォータはソフトクォータであり、引き上げることができます。

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
`CreateActivity`	100	1	100	1

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
CreateStateMachine	100	1	100	1
DeleteActivity	100	1	100	1
DeleteStateMachine	100	1	100	1
DescribeActivity	200	1	200	1
DescribeExecution	300	15	250	10
DescribeStateMachine	200	20	200	20
DescribeStateMachineForExecution	200	1	200	1
GetActivityTask	3,000	500	1,500	300
GetExecutionHistory	400	20	400	20
ListActivities	100	10	100	5
ListExecutions	200	5	100	2

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
ListStateMachines	100	5	100	5
ListTagsForResource	100	1	100	1
SendTaskFailure	3,000	500	1,500	300
SendTaskHeartbeat	3,000	500	1,500	300
SendTaskSuccess	3,000	500	1,500	300

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
StartSync Execution	同期 Express 実行 API コールは、既存のアカウント容量制限には影響しません。Step Functions は、オンデマンドで容量を提供し、持続的なワークロードを使って、自動的に拡張します。容量が利用可能になるまで、ワークロードの急増をスロットリングできます。 スロットリングが発生した場合は、しばらくしてからもう一度試してください。同期 Express ワークフローの詳細については、「同期および非同期 Express ワークフロー」を参照してください。			

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名	バケットサイズ	補充レート/秒	バケットサイズ	補充レート/秒
StopExecution	1,000	200	500	25
TagResource	200	1	200	1
UntagResource	200	1	200	1
UpdateStateMachine	100	1	100	1

ステートマシンの実行に関連するクォータ

次の表では、ステートマシンの実行に関連するクォータについて説明しています。ステートマシンの実行クォータは、実行履歴の保持時間のクォータを除き、変更できないハードクォータです。

Quota	規格	Express
最大実行時間	1 年。実行が最大 1 年を超えると、States.Timeout エラーで失敗し、メトリクスが出力されます。ExecutionsTimedOut CloudWatch	5 分。実行が最大 5 分を超えて実行されると、States.Timeout エラーで失敗し、メトリクスが出力されます。ExecutionsTimedOut CloudWatch
実行履歴の最大サイズ	1 つのステートマシンの実行履歴の 25,000 件のイベント。実行履歴がこのクォータに達すると実行は失敗します。これを回避するには、履歴のクォータに到達しないようにする を参照してください	無制限。

Quota	規格	Express
最大実行アイドル時間	1 年 (最大実行時間によって制限されます)	5 分 (最大実行時間によって制約される)
実行履歴の保持時間	実行が終了してから 90 日後。この期間後は、実行履歴の取得や表示はできません。Step Functions が保持するクローズした実行の数にはこれ以上のクォータはありません。 コンプライアンス、組織、または規制の要件を満たすために、クォータリクエストを送信することによって実行履歴の保持期間を 30 日に短縮できます。そのためには、を使用して新しいケースを作成してください。 AWS Support Center Console 保存期間を 30 日に短縮する変更は、リージョンのアカウントレベルで適用されます。	実行履歴を表示するには、Amazon CloudWatch Logs 口ギングを設定する必要があります。詳細については、「CloudWatch Logs を使用した口グ記録」を参照してください。
実行 redrivable 期間 Redrivable 期間とは、特定の標準ワークフロー実行を redrive することができる期間のことです。この期間は、ステートマシンが実行を完了した日から始まります。	14 日間。 このハードクォータは分散マップ状態に適用されます。	Redrive は現在、Express ワークフローではサポートされていません。

タスクの実行に関連するクォータ

次の表は、タスク実行に関連するクォータについて説明し、変更できないハードクォータです。

Quota	規格	Express
最大タスク実行時間	1 年 (最大実行時間によって制限されます)	5 分 (最大実行時間によって制約される)
Step Functions がキューにタスクを保持する最大時間	1 年 (最大実行時間によって制限されます)	5 分 (最大実行時間によって制約される)
Amazon リソースネーム (ARN) あたりのアクティビティポーラー最大数	ARN あたりに `GetActivityTask` を呼び出すポーラー数: 1,000 このクォータを超えるとこのエラーが表示されます。「アクティビティタスクで同時にポーリングするワーカーの最大数に達しました」	Express ワークフローには適用されません。
タスク、状態、実行の最大の入力または出力サイズ	UTF-8 でエンコードされた文字列としての 256 KB のデータ。このクォータは、タスクのスケジュール、状態の入力、または実行の開始時に、タスク (アクティビティ、Lambda 関数、または統合サービス)、状態または実行出力、入力データに影響します。	UTF-8 でエンコードされた文字列としての 256 KB のデータ。このクォータは、タスクのスケジュール、状態の入力、または実行の開始時に、タスク (アクティビティ、Lambda 関数、または統合サービス)、状態または実行出力、入力データに影響します。

バージョンとエイリアスに関連するクォータ

リソース	デフォルトのクォータ
公開されているステートマシンバージョンの最大数	ステートマシンあたり 1000。 このソフトリミットの引き上げをリクエストするには、AWS Management Console の [サポートセンター] ページを使用してください。
ステートマシンエイリアスの最大数	ステートマシンあたり 1000。 このソフトリミットの引き上げをリクエストするには、AWS Management Console の [サポートセンター] ページを使用してください。

タグ付けに関連する制限

Step Functions リソースにタグ付けの場合は、次の制限に注意してください。

> **Note**
>
> タグ付けの制限は、他のクォータと違って増やすことはできません。

制限	説明
リソースあたりのタグの最大数	50
キーの最大長	UTF-8 で 128 文字の Unicode 文字
値の最大長	UTF-8 で 256 文字の Unicode 文字
プレフィックスの制限	aws: プレフィックスは使用専用なので、 AWS タグ名や値には使用しないでください。このプレフィックスが含まれるタグの名前または値は編集または削除できません。このプレフィック

制限	説明
	スの付いたタグは、リソースあたりのタグ数のクォータにカウントされません。
文字の制限	タグに使用できるのは、Unicode 文字、数字、空白、または _ . : / = + - @ のみです。

AWS Step Functionsでのログ記録とモニタリング

ログ記録とモニタリングは、Step Functions および AWS ソリューションの信頼性、可用性、およびパフォーマンスを維持する上で重要な役割を果たします。Step Functions で使用できるツールが複数あります。

> **Tip**
>
> サンプルワークフローを AWS アカウント デプロイしてワークフロー実行のメトリクス、ログ、トレースをモニタリングする方法を学ぶには、「AWS Step Functions ワークショップ」の「モジュール 12 – オブザーバビリティ」を参照してください。

トピック

- を使用した Step Functions のモニタリング CloudWatch
- EventBridge Step Functions 実行ステータス変更の (CloudWatch イベント)
- AWS CloudTrail を使用して Step Functions をログ記録
- CloudWatch Logs を使用したログ記録
- AWS X-Ray および Step Functions
- AWS User Notificationsと使用するAWS Step Functions

を使用した Step Functions のモニタリング CloudWatch

モニタリングは、 AWS Step Functions および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害をデバッグできるように、使用する AWS サービスからモニタリングデータを収集する必要があります。ただし、Step Functions のモニタリングをスタートする前に、以下の質問に回答するモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?

- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、通常の パフォーマンスのベースラインを環境に確立します。これを行うには、さまざまな時間帯に、さまざまな負荷条件でパフォーマンスを測定します。Step Functions をモニタリングするには、モニタリングデータの履歴を保存することを検討します。このデータを、最新のパフォーマンスデータと比較するベースラインとして使用し、通常のパフォーマンスのパターンやパフォーマンスの異常を検出して、問題への対応を検討することができます。

例えば、Step Functions を使用すると、ハートビートタイムアウトが原因で失敗したアクティビティまたは AWS Lambda タスクの数をモニタリングできます。パフォーマンスが確立したベースラインを下回った場合、ハートビート間隔を変更する必要が生じることがあります。

ベースラインを確立するには、少なくとも、次のメトリクスをモニタリングする必要があります。

- `ActivitiesStarted`
- `ActivitiesTimedOut`
- `ExecutionsStarted`
- `ExecutionsTimedOut`
- `LambdaFunctionsStarted`
- `LambdaFunctionsTimedOut`

以下のセクションでは、Step Functions が Amazon に提供するメトリクスについて説明します CloudWatch。これらのメトリクスを使用してステートマシンおよびアクティビティを追跡し、しきい値のアラームを設定できます。を使用してメトリクスを表示できます AWS Management Console。

時間間隔を報告するメトリクス

Step Functions CloudWatch メトリクスの一部は時間間隔 で、常にミリ秒単位で測定されます。これらのメトリクスは通常、ステートマシン、アクティビティ、Lambda 関数タイムアウトを設定できる実行段階に対応しており、わかりやすい名前が付いています。

例えば、`ActivityRunTime` メトリクスは、アクティビティが実行をスタートしてから完了するまでの時間を測定します。同じ期間のタイムアウト値を設定できます。

CloudWatch コンソールでは、時間間隔メトリクスの表示統計として平均を選択すると、最良の結果が得られます。

カウントを報告するメトリクス

一部の Step Functions CloudWatch メトリクスでは、結果がカウント として報告されます。例えば、ExecutionsFailed はステートマシンの実行が失敗した回数を記録します。

Step Functions は、ステートマシンの実行ごとに 2 つのExecutionsStartedメトリクスを出力します。これにより、ステートマシンの実行ごとに SampleCount ExecutionsStartedメトリクスの 統計に 2 の値が表示されます。 SampleCount 統計にはExecutionStarted=1、実行が完了したExecutionStarted=0日時が表示されます。

> **Tip**
>
> CloudWatch コンソールでカウントを報告するメトリクスの表示統計として Sum を選択することをお勧めします。

実行メトリクス

AWS/States 名前空間には、すべての Step Functions 実行に関する以下のメトリクスが含まれます。これらは、リージョンのアカウント全体に適用されるディメンションレスメトリクスです。

メトリクス	説明
OpenExecutionCount	現在開いている実行の概数 — アカウントで現在進行中のワークフロー。 その目的は、ワークフローがいつ最大実行制限に近づいているかを把握し、標準ワークフローRedriveExecution の StartExecution または を呼び出す際のExecutionLimitExceededエラーを回避することです。 メトリクスはアクティブなワークフロー状態の遷移に依存するため、低いレベルでは、予測が観測された実行中のワークフロー数と一致しない場合があります。
OpenExecutionLimit	オープン実行の最大数。詳細については、「アカウントに関連するクォータ」を参照してください。 この制限は Express ワークフローには適用されません。

バージョンまたはエイリアスを持つステートマシンの実行メトリクス

バージョン または エイリアス を使用してステートマシンの実行を実行すると、Step Functions は次のメトリクスを出力します。ExecutionThrottled メトリクスは、スロットリングされた実行の場合にのみ出力されます。これらのメトリクスには、特定のステートマシンを識別StateMachineArnするための が含まれます。

メトリクス	説明
ExecutionTime	実行の開始から終了までの間隔をミリ秒単位で表します。
ExecutionThrottled	スロットリングされたStateEntered イベントと再試行の数。これは StateTransition のスロットリングに関連しています。詳細については、「状態のスロットリングに関連するクォータ」を参照してください。
ExecutionsAborted	中止または終了した実行の数。
ExecutionsFailed	失敗した実行の数。
ExecutionsStarted	開始された実行の数。
ExecutionsSucceeded	正常に完了した実行の数。
ExecutionsTimedOut	何らかの理由でタイムアウトした実行の数。

Express ワークフローの実行メトリクス

AWS/States 名前空間には、以下の Step Functions Express Workflows 実行のメトリクスが含まれます。

メトリクス	説明
ExpressExecutionMemory	Express ワークフローによって消費されるメモリの合計。
ExpressExecutionBilledDuration	Express ワークフローに課金される期間。

メトリクス	説明
ExpressExecutionBilledMemory	Express ワークフローに課金される消費メモリ量。

標準ワークフローの Redrive 実行メトリック

redrive ステートマシンを実行すると、Step Functions は次のメトリクスを出力します。

すべての redriven 実行で、Executions* メトリクスが出力されます。例えば、redriven実行が中止されたとします。この実行では、RedrivenExecutionsAborted と ExecutionsAborted の両方で、0 以外のデータポイントが出力されます。

メトリクス	説明
ExecutionsRedriven	redriven 実行の数。
RedrivenExecutionsAborted	キャンセルまたは終了されたredriven実行の数。
RedrivenExecutionsTimedOut	何らかの理由でタイムアウトしたredriven実行の数。
RedrivenExecutionsSucceeded	正常に完了したredriven実行の数。
RedrivenExecutionsFailed	失敗したredriven実行の数。

Step Functions 実行メトリクスのディメンション

ディメンション	説明
StateMachineArn	当該の実行に関するステートマシンの Amazon リソースネーム (ARN)

バージョンを使用した実行のディメンション

ディメンション	説明
StateMachineArn	バージョンによって実行が開始されたステートマシンの Amazon リソースネーム (ARN)。
Version	実行を開始するために使用されるステートマシンのバージョン。

エイリアスを使用した実行のディメンション

ディメンション	説明
StateMachineArn	エイリアスによって実行が開始されたステートマシンの Amazon リソースネーム (ARN)。
Alias	実行を開始するために使用されるステートマシンのエイリアス。

バージョンとエイリアスのリソース数メトリクス

AWS/States 名前空間には、ステートマシンのバージョンとエイリアスの数の、以下のメトリクスが含まれます。

メトリクス	説明
AliasCount	ステートマシン用に作成されたエイリアスの数。 ステートマシンごとに最大 100 個のエイリアスを作成できます。
VersionCount	ステートマシンに対して発行されたバージョンの数。 ステートマシンのバージョンを最大 1000 個公開できます。

バージョンとエイリアスのリソース数メトリクスのディメンション

ディメンション	説明
ResourceArn	バージョンまたはエイリアスを持つステートマシンの Amazon リソースネーム (ARN)。

アクティビティのメトリクス

AWS/States 名前空間には、以下の Step Functions アクティビティのメトリクスが含まれます。

メトリクス	説明
ActivityRunTime	アクティビティの開始から終了までの間隔をミリ秒単位で表します。
ActivityScheduleTime	アクティビティがスケジュール状態のままであるミリ秒単位の間隔。
ActivityTime	アクティビティがスケジュールされてから終了するまでのミリ秒単位の間隔。
ActivitiesFailed	失敗したアクティビティの数。
ActivitiesHeartbeatTimedOut	ハートビートタイムアウトが原因でタイムアウトしたアクティビティの数。
ActivitiesScheduled	スケジュールされたアクティビティの数。
ActivitiesStarted	開始されたアクティビティの数。
ActivitiesSucceeded	正常に完了したアクティビティの数。
ActivitiesTimedOut	閉鎖時にタイムアウトしたアクティビティの数。

Step Functions アクティビティメトリクスのディメンション

ディメンション	説明
ActivityArn	アクティビティの ARN。

Lambda 関数メトリクス

AWS/States 名前空間には、Step Functions Lambda 関数の以下のメトリクスが含まれます。

メトリクス	説明
LambdaFunctionRunTime	Lambda 関数の開始から終了までの間隔をミリ秒単位で表します。
LambdaFunctionScheduleTime	Lambda 関数がスケジュール状態のままであるミリ秒単位の間隔。
LambdaFunctionTime	Lambda 関数がスケジュールされてから閉じられるまでの間隔をミリ秒単位で表します。
LambdaFunctionsFailed	失敗した Lambda 関数の数。
LambdaFunctionsScheduled	スケジュールされた Lambda 関数の数。
LambdaFunctionsStarted	開始された Lambda 関数の数。
LambdaFunctionsSucceeded	正常に完了した Lambda 関数の数。
LambdaFunctionsTimedOut	クローズ時にタイムアウトする Lambda 関数の数。

Step Functions Lambda 関数メトリクスのディメンション

ディメンション	説明
LambdaFunctionArn	Lambda 関数の ARN。

> **Note**
>
> Lambda 関数メトリックスは、Lambda 関数 ARN を Resource フィールドに指定するタスク状態に対して発行されます。"Resource": "arn:aws:states:::lambda:invoke" を使用するタスクの状態は、代わりにサービス統合メトリックを発行します。詳細については、「Step Functions で Lambda を呼び出す」を参照してください。

サービス統合メトリクス

AWS/States 名前空間には、Step Functions サービス統合に対する次のメトリクスが含まれます。詳細については、「AWS Step Functions 他のサービスとの併用」を参照してください。

メトリクス	説明
ServiceIntegration RunTime	サービスタスクの開始から終了までの間隔をミリ秒単位で表します。
ServiceIntegration ScheduleTime	サービスタスクがスケジュール状態のままであるミリ秒単位の間隔。
ServiceIntegration Time	サービスタスクがスケジュールされてから閉じられるまでの間隔をミリ秒単位で表します。
ServiceIntegration sFailed	失敗したサービスタスクの数。
ServiceIntegration sScheduled	スケジュールされたサービスタスクの数。

メトリクス	説明
ServiceIntegrationsStarted	開始されたサービスタスクの数。
ServiceIntegrationsSucceeded	正常に完了したサービスタスクの数。
ServiceIntegrationsTimedOut	閉鎖時にタイムアウトしたサービスタスクの数。

Step Functions サービス統合メトリクスのディメンション

ディメンション	説明
ServiceIntegrationResourceArn	統合されたサービスのリソースの ARN。

サービスメトリクス

AWS/States 名前空間には、Step Functions サービスの次のメトリクスが含まれます。

メトリクス	説明
ThrottledEvents	スロットリングされたリクエストの数。
ProvisionedBucketSize	1 秒あたりの利用可能なリクエスト数。
ProvisionedRefillRate	バケットに許可される 1 秒あたりのリクエスト数。
ConsumedCapacity	1 秒あたりのリクエスト数。

Step Functions サービスメトリックのディメンション

ディメンション	説明
ServiceName	データをフィルタリングして状態遷移メトリクスを表示します。

API のメトリクス

AWS/States 名前空間には、Step Functions API のため、以下のメトリクスが含まれます。

メトリクス	説明
ThrottledEvents	スロットリングされたリクエストの数。
ProvisionedBucketSize	1 秒あたりの利用可能なリクエスト数。
ProvisionedRefillRate	バケットに許可される 1 秒あたりのリクエスト数。
ConsumedCapacity	1 秒あたりのリクエスト数。

Step Functions API メトリクスのディメンション

ディメンション	説明
APIName	指定された API 名の API にデータをフィルタリングします。

ベストエフォート型の CloudWatch メトリクス配信

CloudWatch メトリクスは、ベストエフォートで配信されます。

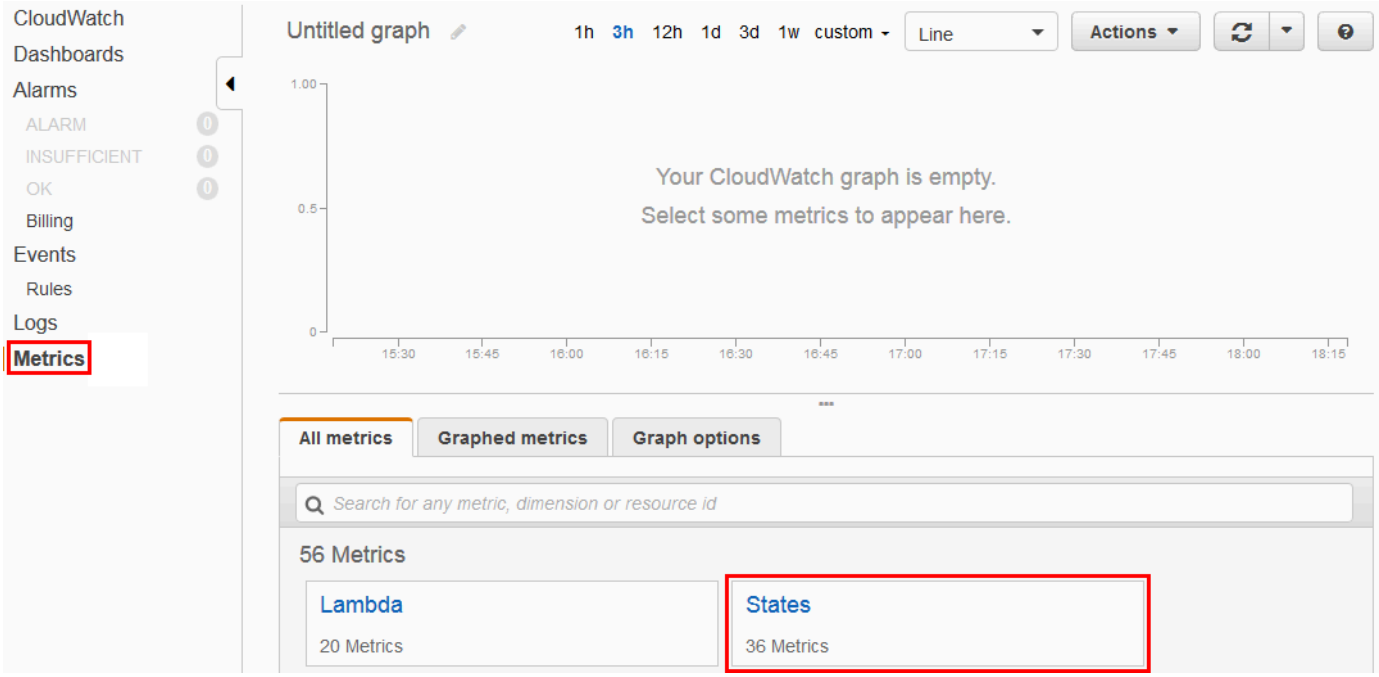
メトリクスの完全性や適時性は保証されません。特定のリクエストのデータポイントが、リクエストが実際に処理されたよりも後のタイムスタンプで返される場合があります。1 分間のデータポイント

は、 を通じて使用可能になる前に遅れたり CloudWatch、まったく配信されなかったりすることがあります。 CloudWatchリクエストメトリクスを使用すると、ステートマシンの実行をほぼリアルタイムで把握できます。これは、実行に関するメトリクスのすべてを完全に報告するためのものではありません。

この機能はベストエフォート型であるため、請求情報とコスト管理ダッシュボードで利用できるレポートには、実行メトリクスに表示されない 1 つ以上のアクセスリクエストが含まれることがあります。

Step Functions メトリクスの表示

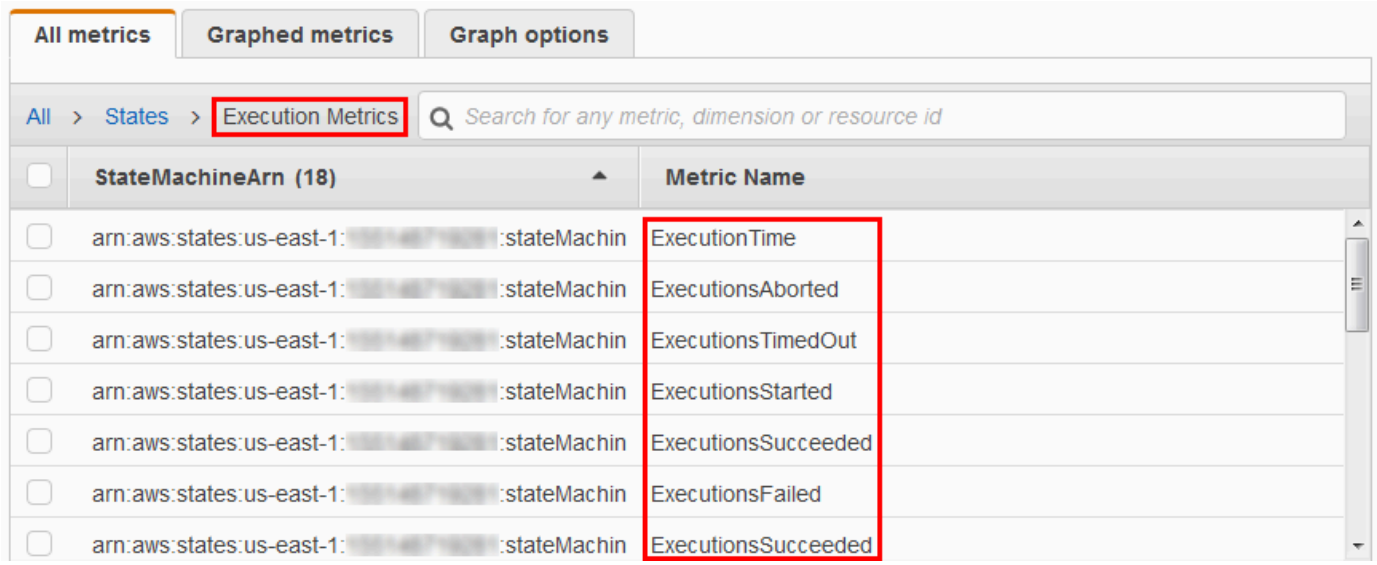
1. にサインイン AWS Management Console し、 CloudWatch コンソールを開きます。
2. [Metrics] (メトリクス) を選択し、[All Metrics] (すべてのメトリクス) タブで、[States] (状態) を選択します。



最近実行した場合、最大 4 タイプのメトリクスが表示されます。

- 実行メトリクス
- アクティビティ関数メトリクス
- Lambda 関数メトリクス
- サービス統合メトリクス

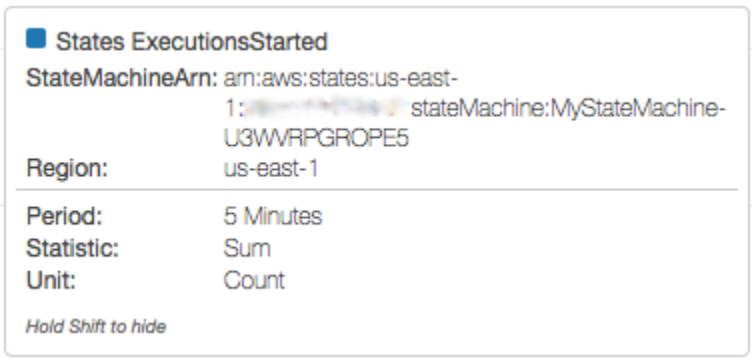
3. メトリクスタイプを選択してメトリクスのリストを表示します。



- メトリクス名または でメトリクスをソートするにはStateMachineArn、列見出しを使用します。
- メトリクスのグラフを表示するには、リストでメトリクスの横にあるボックスをオンにします。グラフパラメータは、グラフビューの上にある時間範囲コントロールを使用して変更できます。

  相対値または絶対値 (日時を指定) を使用してカスタム時間範囲を選択できます。ドロップダウンリストを使用して値を線、積み上げ領域、または数字 (値) として表示することもできます。
- グラフの詳細を表示するには、グラフの下に表示されるメトリクス色コードにマウスカーソルを合わせます。

  メトリクスの詳細が表示されます。



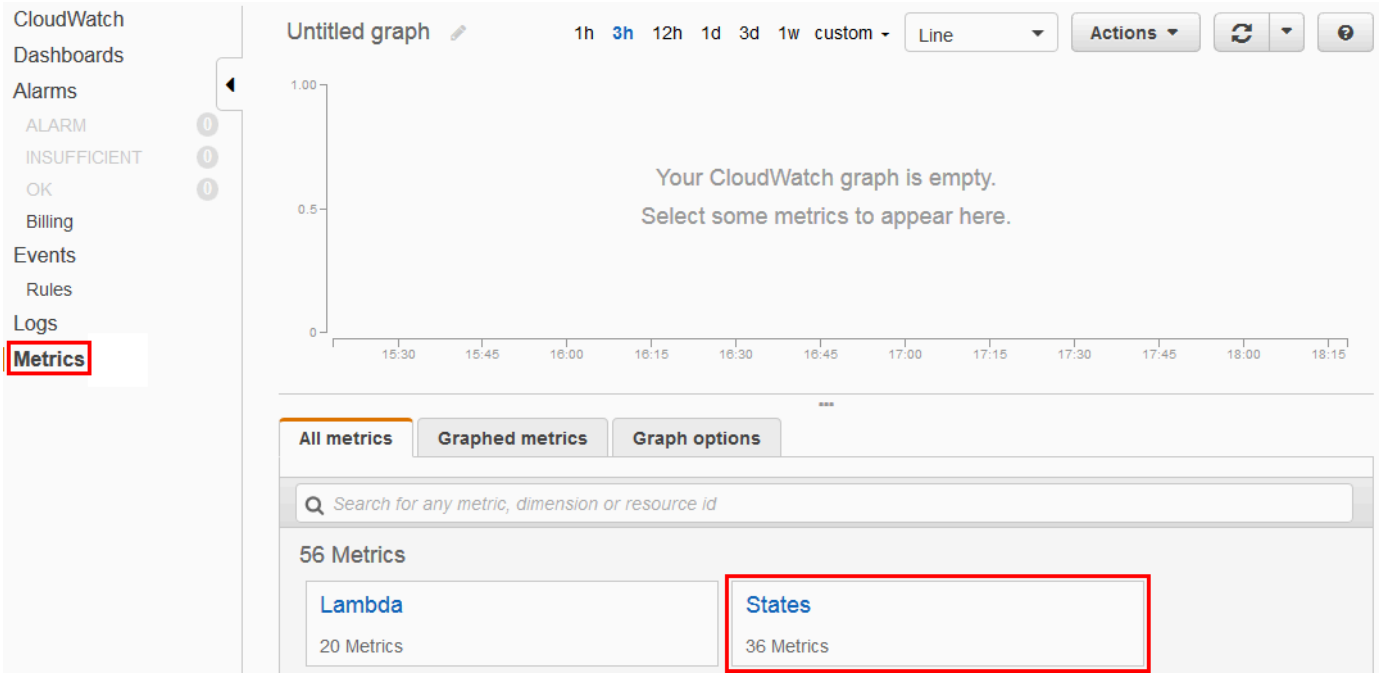
CloudWatch メトリクスの使用の詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch メトリクスの使用」を参照してください。 CloudWatch

Step Functions アラームの設定

Amazon CloudWatch アラームを使用してアクションを実行できます。例えば、アラームのしきい値に到達するのがいつか知りたい場合、`StateMachinesFailed` メトリクスが一定のしきい値を超えたときに Amazon SNS トピックに通知を送信したり、メールを送信したりするようにアラームを設定できます。

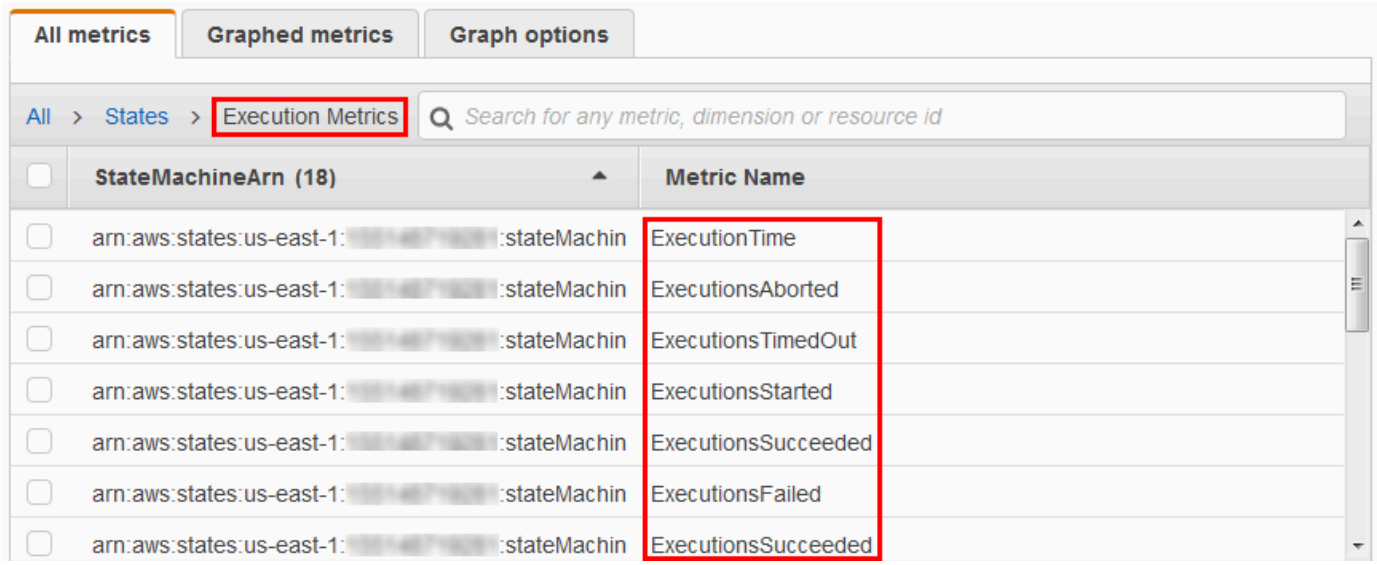
メトリクスにアラームを設定するには

1. にサインイン AWS Management Console し、 CloudWatch コンソールを開きます。
2. [Metrics] (メトリクス) を選択し、[All Metrics] (すべてのメトリクス) タブで、[States] (状態) を選択します。



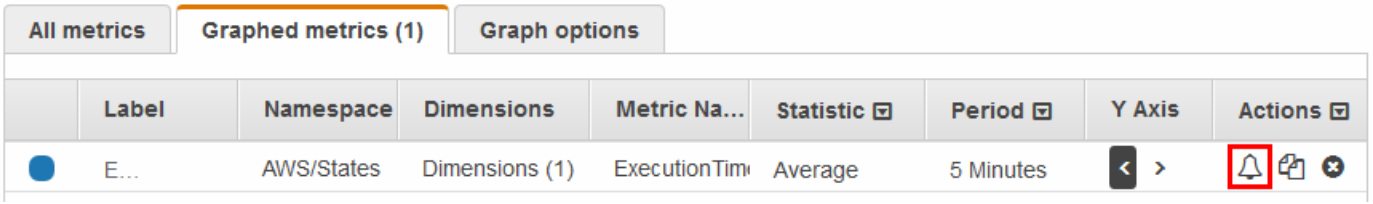
   最近実行した場合、最大 4 タイプのメトリクスが表示されます。

   - 実行メトリクス
   - アクティビティ関数メトリクス
   - Lambda 関数メトリクス
   - サービス統合メトリクス
3. メトリクスタイプを選択してメトリクスのリストを表示します。

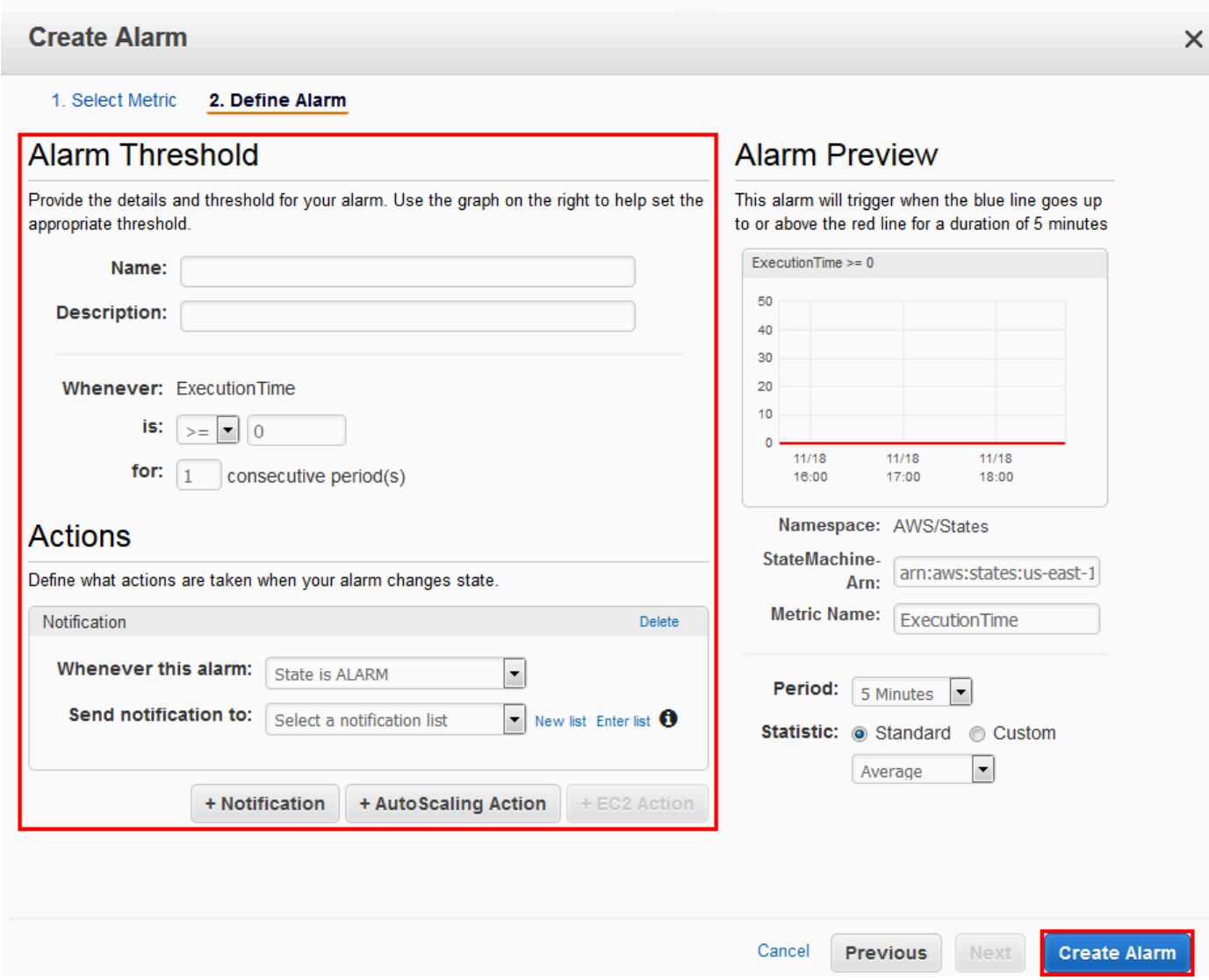


4. メトリクスを選択し、[Graphed metrics] (グラフ化したメトリクス) を選択します。
5. リストのメトリクスの横にある

   を選択します。



[Create Alarm] (アラームの作成) ページが表示されます。



6. [Alarm threshold] (アラームしきい値) と [Actions] (アクション) に値を入力し、[Create Alarm] (アラームの作成) を選択します。

CloudWatch アラームの設定と使用の詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch アラームの作成」を参照してください。 CloudWatch

EventBridge Step Functions 実行ステータス変更の (CloudWatch イベント)

Amazon EventBridge は、 AWS リソースの状態の変化に応答できる AWS のサービスです。例えば、次の 2 つの方法 EventBridge を使用して、Step Functions 標準ワークフローの実行ステータスの変更に で応答できます。

- Step Functions ステートマシンの実行ステータスが変更されたときに発生するイベントに対応するように EventBridge ルールを設定できます。これにより、DescribeExecution API を使用して継続的にポーリングすることなく、ワークフローをモニタリングできます。ステートマシンの実行の変更に基づいて、 EventBridge ターゲットを使用して、新しいステートマシンの実行の開始、 AWS Lambda 関数の呼び出し、Amazon Simple Notification Service (Amazon SNS) トピックへのメッセージの発行などを行うことができます。
- Step Functions ステートマシンを のターゲットとして設定することもできます EventBridge。これにより、他の AWS のサービスからのイベントに応答して、Step Functions ワークフローの実行をトリガーできます。

  詳細については、「Amazon ユーザーガイド EventBridge 」を参照してください。

ただし、Express ワークフローは にイベントを発行しません EventBridge。Express ワークフローの実行をモニタリングするには、 CloudWatch ログを使用できます。これを行うには、ステートマシンの [実行の詳細] ページで [モニタリング] タブと [ロギング] タブを選択します。モニタリングタブでは、実行期間、実行エラー、請求済みメモリ などのイベントの CloudWatch メトリクスを表示できます。[ログ記録] タブでは、最近のログとログ記録設定を確認できます。

> **Tip**
>
> Express ワークフローの例を にデプロイ AWS アカウント し、Express ワークフローをモニタリングする方法については、 AWS Step Functions ワークショップの「Monitoring Express Workflows module」を参照してください。

EventBridge ペイロード

EventBridge イベントの定義には入力プロパティを含めることができます。一部のイベントでは EventBridge、イベントの定義に出力プロパティを含めることもできます。

- に送信されたエスケープされた入力とエスケープされた出力の合計が 248KB EventBridge を超える場合、入力は除外されます。同様に、エスケープされた出力が 248 KB を超える場合、出力は除外されます。これはイベントクォータの結果です EventBridge。
- `inputDetails` と `outputDetails` プロパティでペイロードが切り捨てられているかどうかを判断できます。詳細については、`CloudWatchEventsExecutionDataDetails` データを参照してください。
- 標準ワークフローでは、 を使用して完全な入出力を確認できますDescribeExecution。

- `DescribeExecution` は、Express ワークフローでは使用できません。完全な入出力を表示したい場合は、Express ワークフローを標準ワークフローでラップできます。もう 1 つの方法は、Amazon S3 の ARN を使用することです。ARN 使用の詳細については、the section called “ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する” を参照してください。

トピック

- Step Functions イベントの例
- EventBridge コンソール EventBridge で Step Functions イベントを にルーティングする

Step Functions イベントの例

Step Functions が にイベントを送信する例を次に示します EventBridge。

トピック

- 実行のスタート
- 実行の成功
- 実行の失敗
- 実行のタイムアウト
- 実行の中断

いずれの場合も、イベントデータの `detail` セクションに、`DescribeExecution` API と同じ情報が含まれています。`status` フィールドは、イベント送信時の実行ステータス (放出されたイベントに応じて`RUNNING`、`SUCCEEDED`、`FAILED`、`TIMED_OUT`、`ABORTED` のいずれか) を示します。

実行のスタート

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
```

```
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws::states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "RUNNING",
        "startDate": 1551225271984,
        "stopDate":  null,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

実行の成功

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "SUCCEEDED",
        "startDate": 1547148840101,
        "stopDate": 1547148840122,
        "input": "{}",
```

```
        "inputDetails": {
             "included": true
        },
        "output": "\"Hello World!\"",
        "outputDetails": {
             "included": true
        }
    }
}
```

実行の失敗

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "FAILED",
        "startDate": 1551225146847,
        "stopDate": 1551225151881,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

実行のタイムアウト

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "TIMED_OUT",
        "startDate": 1551224926156,
        "stopDate": 1551224927157,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

実行の中断

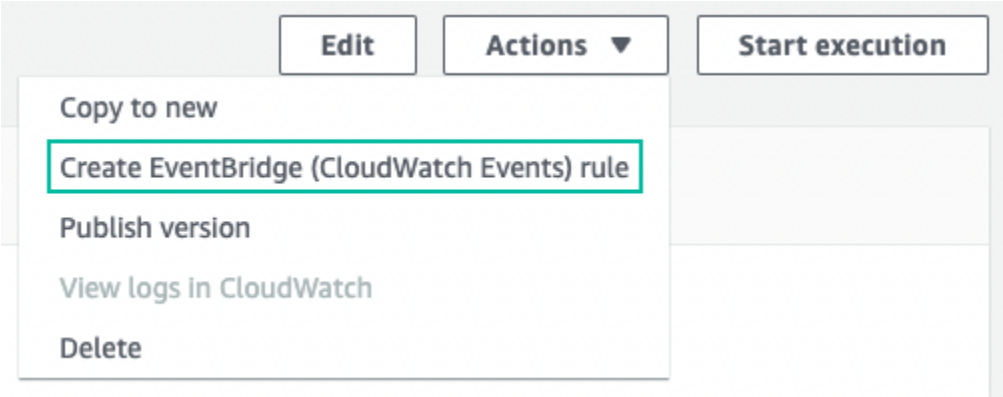
```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
```

```
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "ABORTED",
        "startDate": 1551225014968,
        "stopDate": 1551225017576,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

EventBridge コンソール EventBridge で Step Functions イベントを にルーティングする

以下の手順に従って、特定の Step Functions ステートマシンが正常に実行を完了したときに Step Functions ステートマシンの実行をトリガーする方法を学習してください。Amazon EventBridge コンソールを使用して、実行をトリガーするステートマシンを指定します。

1. ステートマシンの詳細 ページで、アクション を選択し、作成 EventBridge（ CloudWatch イベント) ルール を選択します。



または、https://console.aws.amazon.com/events/ で EventBridge コンソールを開きます。ナビゲーションペインの [バス] で、[ルール] を選択します。

2. [ルールの作成] を選択します。これにより、[ルールの詳細を定義] ページが開きます。
3. ルールの [名前] (例: *StepFunctionsEventRule*) を入力し、(オプション) ルールの [説明] を入力します。
4. [イベントバス] と [ルールタイプ] は、デフォルト設定のままにします。
5. [次へ] を選択します。これにより、[イベントパターンを構築] ページが開きます。
6. イベントソース で、AWS イベントまたは EventBridge パートナーイベントのデフォルト選択を維持します。
7. [サンプルイベント] セクションと [作成方法] セクションはデフォルトのままにします。
8. [イベントパターン] で、次のいずれかを実行します。

   a. [イベントソース] ドロップダウンリストでは、AWS のサービスのデフォルトの選択のままにします。

   b. [AWS のサービス] の場合、リストから [Step Functions] を選択します。

   c. [イベントタイプ] ドロップダウンリストから、[Step Functions の実行ステータスの変更] を選択します。

   d. (オプション) 特定のステータス、ステートマシンの Amazon リソースネーム (ARN)、または実行 ARN を設定します。この手順では、[特定のステータス] を選択し、ドロップダウンリストから[SUCCEEDED] を選択します。
9. [次へ] を選択します。これにより、[ターゲットを選択] ページが開きます。
10. [ターゲットタイプ] では、デフォルトの [AWS のサービス] を選択したままにします。
11. ターゲットを選択ドロップダウンリストから、 AWS サービスを選択します。例えば、Lambda 関数を起動、または Step Functions ステートマシンを実行できます。この手順では、[Step Functions ステートマシン] を選択します。
12. [ステートマシン] ドロップダウンリストから、ステートマシンを選択します。
13. [実行ロール] では、デフォルトの [この特定のリソースに対して新しいロールを作成する] のままにします。
14. [次へ] を選択します。これにより、[タグを設定] ページが開きます。
15. [次へ] をもう一度選択します。これにより、[レビューして作成] ページが開きます。
16. ルールの詳細を確認し、ルールの作成 を選択します。

ルールが作成され、ルールページが表示され、すべての Amazon EventBridge ルールが一覧表示されます。

AWS CloudTrail を使用して Step Functions を口グ記録

Step Functions は、Step Functions 内のユーザー、ロールまたは AWS のサービスによって実行されたアクションのレコードを提供するサービスである AWS CloudTrail と統合されています。CloudTrail は、 Step Functions コンソールからのコールとコードコールから Step Functions API へのコールなど、Step Functions の API コールをイベントとしてキャプチャします。

追跡を作成する場合は、Step Functions のイベントなど、Amazon Simple Storage Service (Amazon S3) バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。

CloudTrail で収集された情報を使用して、Step Functions に対するリクエスト、リクエスタの IP アドレス、リクエスト者、リクエスト日時などの追加詳細を確認できます。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrail における Step Functions 情報

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。Step Functions でアクティビティが発生すると、そのアクティビティは、[Event history] (イベント履歴) にある他の AWS のサービスとともに、 CloudTrail イベントに記録されます。

最近のイベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

Step Functions のイベントなど、AWS アカウントのイベントの継続的なレコードについては、追跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべての AWS リージョンに適用されます。追跡は AWS パーティション内のすべてのリージョンからのイベントを口グに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、その他の AWS のサービスを設定して、CloudTrail 口グで収集したイベントデータをより詳細に分析し、それに基づく対応を行うことができます。

詳細については、次を参照してください。

- 追跡を作成するための概要
- CloudTrail のサポート対象サービスと統合
- Amazon SNS の CloudTrail の通知の設定
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと ユーザー認証情報のどちらを使用して送信されたか
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか
- リクエストが、別の AWS のサービスによって送信されたかどうか

詳細については、CloudTrail userIdentity エレメントを参照してください。

例:Step Functions ログファイルエントリ

「トレイル」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、単一か複数のログエントリがあります。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

CreateActivity

次の例は、`CreateActivity` アクションを示す CloudTrail ログエントリです。

```
{
    "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": "AWS Internal",
    "userAgent": "AWS Internal",
    "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-4cf4-8523-37
        "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

次の例は、CreateStateMachine アクションを示す CloudTrail ログエントリです。

```
{
    "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": "AWS Internal",
    "userAgent": "AWS Internal",
    "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\": {           \"Type\": \"Pass\",           \"End\": true       }   }}"
    },
    "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"
}
```

DeleteActivity

次の例は、DeleteActivity アクションを示す CloudTrail ログエントリです。

```
{
      "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": "AWS Internal",
      "userAgent": "AWS Internal",
      "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

次の例は、DeleteStateMachine アクションを示す CloudTrail ログエントリです。

```
{
    "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": "AWS Internal",
    "userAgent": "AWS Internal",
    "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

次の例は、StartExecution アクションを示す CloudTrail ログエントリです。

```
{
    "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": "AWS Internal",
    "userAgent": "AWS Internal",
    "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

次の例は、StopExecutionアクションを示す CloudTrail ログエントリです。

```
{
    "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": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:testUser.2016-10-27-18-17-00.337b3344-83:testUser.2016-10-27-18-1
    },
    "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"
}
```


CloudWatch Logs を使用したログ記録

標準ワークフロー では、AWS Step Functions に実行履歴が記録されますが、オプションで Amazon CloudWatch Logs へのログ記録を設定することもできます。

Express ワークフローは、標準ワークフローとは異なり AWS Step Functions に実行履歴を記録しません。Express ワークフローの実行履歴と結果を表示するには、Amazon CloudWatch Logs へのログ記録を設定する必要があります。ログを発行しても、実行がブロックまたは遅くなることはありません。

> **Note**
>
> ログ記録を設定すると、CloudWatch Logs 料金が適用され、提供されたログのレートで請求されます。詳細については、「CloudWatch の料金」ページの [ログ] タブにある [Vended Logs] を参照してください。

のログ記録の設定

Step Functions コンソールを使用して標準ワークフローを作成する場合、CloudWatch Logs へのログ記録を有効化する設定は行われません。Step Functions コンソールを使用して作成された Express ワークフローは、デフォルトで CloudWatch Logs へのログ記録を有効にするように設定されます。

Express ワークフローでは、Step Functions は CloudWatch Logs に必要な AWS Identity and Access Management (IAM) ポリシーを持つロールを作成できます。API、CLI、または AWS CloudFormation を使用して標準ワークフローまたは Express ワークフローを作成する場合、 ではデフォルトでログ記録が有効にならず、またロールに必要な許可があることを確認する必要があります。

コンソールからスタートした各実行について、Step Functions は CloudWatch Logs へのリンクを提供し、その実行に固有のログイベントを取得するための正しいフィルターが設定されています。

ログ記録を設定するには、CreateStateMachine または UpdateStateMachine を使用するときに、LoggingConfiguration パラメータを渡すことができます。CloudWatch Logs Insights を使用すると CloudWatch Logs でデータをさらに分析できます。詳細については、CloudWatch Logs Insights を使用したログデータの分析を参照してください。

CloudWatch Logs ペイロード

実行履歴イベントは、入力プロパティまたは出力プロパティを定義に含めることができるかもしれません。CloudWatch Logs に送信されるエスケープ入力またはエスケープ出力が 248 KB を超えると、CloudWatch Logs クォータの結果として切り捨てられます。

- `inputDetails` と `outputDetails` パイプラインをレビューして、ペイロードが切り捨てられたかどうかを確定できます。詳細については、`HistoryEventExecutionDataDetails` データを参照してください。
- 標準ワークフローs では、`GetExecutionHistory` を使用して完全な実行履歴を表示できます。
- `GetExecutionHistory` は、Express ワークフローでは使用できません。フル入出力を表示したいのであれば、Amazon S3 ARN を使用できます。詳細については「the section called “ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する”」を参照してください。

CloudWatch Logs にログ記録するための IAM ポリシー

CloudWatch Logs にログ記録するための適切な許可を持つ、ステートマシンの実行 IAM ロールを、以下の例に示されるように設定する必要もあります。

IAM ポリシーの例

以下は、アクセス許可を設定するために使用できるポリシーの例です。次の例に示すように、CreateLogDelivery や DescribeLogGroups などの CloudWatch API アクションは、Amazon CloudWatch Logs で定義されるリソースタイプをサポートしていないため、Resource フィールドには * を指定する必要があります。詳細については、「Amazon CloudWatch Logs で定義されるアクション」を参照してください。

- CloudWatch リソースの情報については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch Logs がリソースとオペレーションをログに記録する」を参照してください。
- CloudWatch Logs にログを送信するよう設定するために必要なアクセス許可については、「CloudWatch Logs に送信されるログ」セクションの「ユーザーアクセス許可」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:CreateLogStream",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutLogEvents",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

CloudWatch Logs にアクセスできない

CloudWatch Logs にアクセスできない場合は、次のことを行っているかを確認してください。

1. CloudWatch Logs にログを記録するための適切なアクセス許可を持つ、ステートマシン実行 IAM ロールを設定した。

CreateStateMachine リクエストまたは UpdateStateMachine リクエストを使用している場合は、前の例に示されているように、アクセス許可を持つ roleArn パラメータに IAM ロールが指定されていることを確認してください。

2. CloudWatch Logs リソースポリシーが CloudWatch Logs リソースポリシーの 5120 文字の制限を超えていないことを確認した。

   文字数制限を超えた場合は、CloudWatch Logs リソースポリシーから不要なアクセス権限を削除するか、ロググループ名にプレフィックス /aws/vendedlogs を付けてください。これにより、リソースポリシーに文字を付加しなくてもロググループにアクセス許可が付与されます。Step Function コンソールでロググループを作成すると、ロググループ名に /aws/vendedlogs/states のプレフィックスが付きます。詳細については「Amazon CloudWatch Logs リソースポリシーのサイズ制限」を参照してください。

ログレベル

OFF、ALL、ERROR、または FATAL のいずれかを選択します。OFF に設定すると、イベントタイプはログに記録されません。ALL に設定すると、すべてのイベントタイプがログに記録されます。ERROR および FATAL については、次の表を参照してください。

これらのログレベルに基づいて Express ワークフローを実行したときに表示される実行データの詳細については、「コンソールでの Standard ワークフローと Express ワークフローの実行」を参照してください。

イベントタイプ	ALL	ERROR	FATAL	OFF
ChoiceStateEntered	✓			
ChoiceStateExited	✓			
ExecutionAborted	✓	✓	✓	
ExecutionFailed	✓	✓	✓	
ExecutionStarted	✓			

イベントタイプ	ALL	ERROR	FATAL	OFF
Execution Succeeded	✓			
Execution TimedOut	✓	✓	✓	
FailStateEntered	✓	✓		
LambdaFunctionFailed	✓	✓		
LambdaFunctionScheduled	✓			
LambdaFunctionScheduleFailed	✓	✓		
LambdaFunctionStarted	✓			
LambdaFunctionStartFailed	✓	✓		
LambdaFunctionSucceeded	✓			
LambdaFunctionTimedOut	✓	✓		
MapIterationAborted	✓	✓		
MapIterationFailed	✓	✓		
MapIterationStarted	✓			

イベントタイプ	ALL	ERROR	FATAL	OFF
MapIterationSucceeded	✓			
MapRunAborted	✓	✓		
MapRunFailed	✓	✓		
MapStateAborted	✓	✓		
MapStateEntered	✓			
MapStateExited	✓			
MapStateFailed	✓	✓		
MapStateStarted	✓			
MapStateSucceeded	✓			
ParallelStateAborted	✓	✓		
ParallelStateEntered	✓			
ParallelStateExited	✓			
ParallelStateFailed	✓	✓		
ParallelStateStarted	✓			

イベントタイプ	ALL	ERROR	FATAL	OFF
ParallelStateSucceeded	✓			
PassStateEntered	✓			
PassStateExited	✓			
SucceedStateEntered	✓			
SucceedStateExited	✓			
TaskFailed	✓	✓		
TaskScheduled	✓			
TaskStarted	✓			
TaskStartFailed	✓	✓		
TaskStateAborted	✓	✓		
TaskStateEntered	✓			
TaskStateExited	✓			
TaskSubmitFailed	✓	✓		
TaskSubmitted	✓			
TaskSucceeded	✓			
TaskTimedOut	✓	✓		

イベントタイプ	ALL	ERROR	FATAL	OFF
WaitState Aborted	✓	✓		
WaitState Entered	✓			
WaitStateExited	✓			

AWS X-Ray および Step Functions

AWS X-Ray を使用して、ステートマシンのコンポーネントの視覚化、パフォーマンスのボトルネックの特定、およびエラーの原因となったリクエストのトラブルシューティングを行うことができます。ステートマシンはトレースデータを X-Ray に送信し、X-Ray はデータを処理してサービスマップと検索可能なトレースサマリーを生成します。

ステートマシンで X-Ray を有効にすると、X-Ray が利用可能なすべての AWS リージョンで Step Functions で実行されるリクエストをトレースできます。これにより、Step Functions リクエスト全体の詳細な概要が提供されます。Step Functions は、トレース ID がアップストリームサービスによって渡されない場合でも、ステートマシンを実行するためにトレースを X-Ray に送信します。X-Ray サービスマップを使用して、X-Ray と統合されているすべての AWS サービスを含む、リクエストのレイテンシーを表示できます。また、サンプリングルールを設定することで、X-Ray で記録するリクエストとサンプリングレートを条件に応じて指定できます。

ステートマシン用 X-Ray が有効にならず、アップストリームサービスがトレース ID を渡さない場合、Step Functions はステートマシン実行用トレースを X-Ray 送信しません。ただし、アップストリームサービスによってトレース ID が渡されると、Step Functions はステートマシン実行のため、トレースを X-Ray に送信します。

両方がサポートされているリージョンでは、Step Functions AWS X-Ray で を使用できます。X-RayとStep Funcion のリージョンサポートに関する情報については、Step Functions と X-Ray エンドポイントとクォータページを参照してください。

> **X-Ray 関数と Step Functions 結合クォータ**
>
> 最大 7 日間、トレースにデータを追加し、30 日前のトレースデータと、X-Ray がトレースデータを格納する時間の長さに関するクエリを行うことができます。トレースは X-Ray

クォータの対象となります。その他のクォータに加えて、X-Ray は Step Functions ステートマシンに最小保証トレースサイズ 100 KB を提供します。X-Ray に 100 KB を超えるトレースデータが提供される場合、フリーズトレースとなる可能性があります。X-Ray の他のクォータの詳細については、X-Ray エンドポイントとクォータ ページの Service Quotas セクションを参照してください。

Important

Step Functions は、分散マップ状態によって開始された子ワークフロー実行の X-Ray トレースをサポートしていません。このような実行では、トレースドキュメントのサイズ制限を簡単に超える可能性があるためです。

トピック

- セットアップと設定
- 概念
- Step Functions サービス統合と X-Ray
- X-Ray コンソールの表示
- Step Functions の X-Ray トレース情報を表示
- トレース
- サービスマップ
- セグメントとサブセグメント
- 分析
- 構成
- トレースマップまたはサービスマップにデータがない場合はどうなりますか。

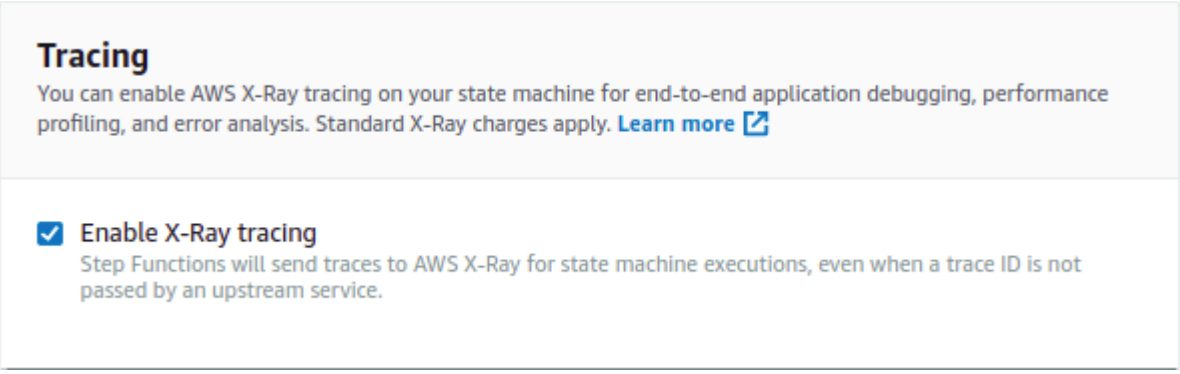
セットアップと設定

ステートマシンの作成時に X-Ray トレースを有効にする

[Specify details] (詳細の指定) ページの [Enable X-Ray tracing] (X-Ray によるトレースを有効にする) を選択して、新しいステートマシンの作成時に X-Ray トレースを有効にできます。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [作成方法を選択] ページで、適切なオプションを選択してステートマシンを作成します。[サンプルプロジェクトを実行] を選択する場合、ステートマシンの作成時に X-Ray トレースを有効にすることはできません。ステートマシンの作成後に X-Ray トレースを有効にする必要があります。既存のステートマシンで X-Ray を有効にする方法の詳細については、「既存のステートマシンで X-Ray を有効にする」を参照してください。

   [次へ] を選択します。
3. [詳細の指定] ページで、ステートマシンを設定します。
4. [Enable X-Ray tracing] (X-Ray によるトレースを有効にする) を選択します。



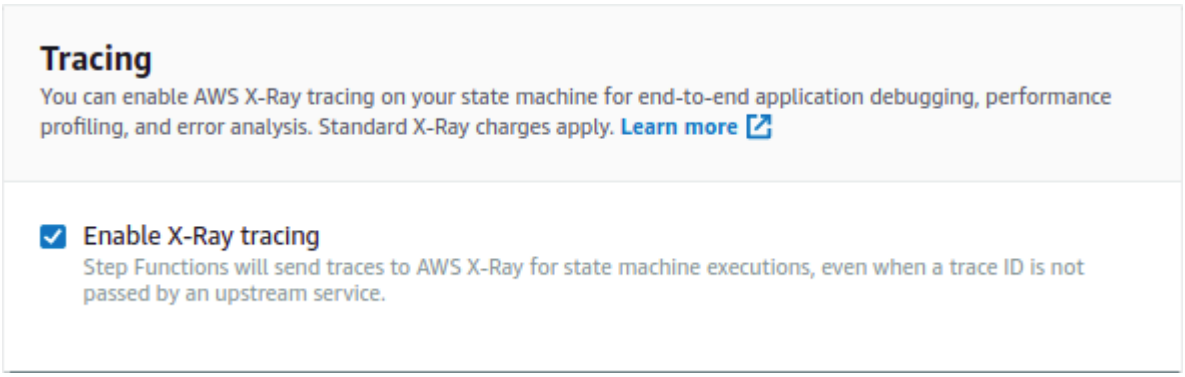
   Step Functions ステートマシンは、ステートマシンの実行のために X-Ray にトレースを送信します。

> **Note**
>
> 既存の IAM ロールの使用を選択する場合は、X-Ray の書き込みを許可を確実に行う必要があります。必要なアクセス許可の詳細については、「X-Ray の IAM ポリシー」を参照してください。

既存のステートマシンで X-Ray を有効にする

既存のステートマシンで X-Ray を有効にする:

1. Step Functions コンソールで、トレースを有効にするステートマシンを選択します。
2. [編集] を選択します。
3. [Enable X-Ray tracing] (X-Ray によるトレースを有効にする) を選択します。



追加の変更が必要になる可能性があることを知らせる通知が表示されます。

> **Note**
>
> 既存のステートマシンで X-Ray を有効にする場合、トレースを実行できるだけのアクセス権限を十分に付与する IAM ポリシーが備わるように調整が必要です。手動で追加するか、生成できます。詳細については、の IAM ポリシー AWS X-Ray の IAM ポリシーのセクションを参照してください。

4. (オプション) X-Ray 権限が含まれるように、ステートマシンの新しいロールを自動生成します。
5. [保存] を選択します。

Step Functions 用 X-Ray トレースを設定

X-Ray トレースを有効にしてステートマシンを初めて実行すると、X-Ray tracing. AWS X-Ray does のデフォルトの設定値を使用して、アプリケーションに送信されるすべてのリクエストのデータが収集されるわけではありません。代わりに、統計的に有意な数のリクエストのデータを収集します。デフォルトでは、毎秒、最初のリクエストを記録し、追加リクエストの 5% を記録します。1 秒あたり 1 つのリクエストがリザーバです。これにより、サービスがリクエストを処理している限り、毎秒少なくとも 1 つのトレースが記録されます。5% は、リザーバサイズを超えて追加リクエストがサンプリングされるレートです。

開始時にサービス料がかからないように、デフォルトのサンプリングレートは控えめになっています。デフォルトのサンプリングルールを変更し、サービスまたはリクエストのプロパティに基づいてサンプリングを適用する追加のルールを設定するように X-Ray を設定できます。

例えば、サンプリングを無効にして、状態を変更したり、 または AWS アカウント トランザクションを処理したりする呼び出しのすべてのリクエストを追跡できます。バックグラウンドポーリング、

ヘルスチェック、接続保守などの大量の読み取り専用呼び出しには、低いレートでサンプリングを行い、発生した問題に気づけるデータを得ることができます。

ステートマシンのサンプリングルールを設定:

1. X-Ray コンソール に移動します。
2. [Sampling] (サンプリング) を選択します。
3. ルールを作成するには、[Create sampling rule] (サンプリングルールの作成) を選択します。

   ルールを編集するには、ルールの名前を選択します。

   ルールを削除するには、ルールを選択し、[Actions] (アクション) メニューを使用して削除します。

名前や優先度など、既存のサンプリングルールの一部は変更できません。代わりに、既存のルールを追加またはクローンを作成し、望みの変更を行い、新しいルールを活用します。

X-Ray サンプリングルールの詳細および各種パラメータの設定方法については、X-Ray コンソールにおけるサンプリングルールの設定を参照してください。

アップストリームサービスの統合

エクスプレス、同期、標準ワークフローなどの Step Functions ワークフローの実行をアップストリームサービスと統合するには、`traceHeader` を設定する必要があります。API Gateway で HTTP API を使用している場合、これは自動的に行われます。ただし、Lambda 関数や SDK を使用している場合は、`StartExecution` または `StartSyncExecution` API 呼び出しで `traceHeader` を自分で設定する必要があります。

`traceHeader` 形式を `\p{ASCII}#` として指定する必要があります。また、Step Functions が同じトレース ID を使用できるようにするには、形式を `Root={TRACE_ID};Sampled={1 or 0}` として指定する必要があります。Lambda 関数を使用している場合は、`TRACE_ID` を現在のセグメントのトレース ID に置き換え、サンプリングモードが true の場合は、Sampled フィールドを `1` に設定し、サンプリングモードが false の場合は `0` に設定します。トレース ID をこの形式で指定すると、完全なトレースが確実に得られます。

以下は、`traceHeader` を指定する方法を示すために Python で記述された例です。

```
state_machine = config.get_string_paramter("STATE_MACHINE_ARN")
    if (xray_recorder.current_subsegment() is not None and
        xray_recorder.current_subsegment().sampled) :
```

```
        trace_id = "Root={};Sampled=1".format(
            xray_recorder.current_subsegment().trace_id
        )
    else:
        trace_id = "Root=not enabled;Sampled=0"
    LOGGER.info("trace %s", trace_id)

    # execute it
    response = states.start_sync_execution(
        stateMachineArn=state_machine,
        input=event['body'],
        name=context.aws_request_id,
        traceHeader=trace_id
    )
    LOGGER.info(response)
```

概念

X-Ray コンソール

AWS X-Ray コンソールでは、アプリケーションが提供するリクエストのサービスマップとトレースを表示できます。ステートマシンで有効になっているときに、X-Ray が収集した詳細情報を閲覧するため、コンソールにアクセスできます。

ステートマシン実行のために X-Ray コンソールにアクセスする方法に関する情報は、X-Ray コンソールの表示 を参照してください。

X-Ray コンソールの詳細については、X-Ray コンソールのドキュメントを参照してください。

セグメント、サブセグメント、およびトレース

ステートマシンへのリクエストに関する情報をセグメントが記録します。これには、ステートマシンが実行する作業などの情報が含まれており、ダウンストリームコールに関する情報があるサブセグメントを含む可能性があります。

トレースでは、1 つのリクエストで生成されたセグメントをすべて収集します。

サンプリング

効率的にトレースを行ってアプリケーションが処理するリクエストの代表的なサンプルを提供するため、X-Ray によってサンプリングアルゴリズムが適用され、トレースするリクエストが決定されます。これは、サンプリングルールを編集すれば、変更できます。

メトリクス

ステートマシンの場合、X-Ray は呼び出し時間、状態遷移時間、Step Functions 全実行時間、およびこの実行時間の分散を計測します。この情報には、X-Ray コンソールを通じてアクセスできます。

分析

AWS X-Ray Analytics コンソールは、トレースデータを解釈するためのインタラクティブなツールです。現在のトレースセットに関連付けられているグラフとメトリクスおよびフィールドのパネルをクリックして、アクティブなデータセットをきめ細かく絞り込むことができます。これにより、ステートマシンのパフォーマンス、パフォーマンス問題のすばやい発見と特定方法を分析できます。

X-Ray 分析の詳細については、AWS X-Ray 「分析コンソールの操作」を参照してください。

Step Functions サービス統合と X-Ray

Step Functions と統合されている AWS サービスの中には、リクエストにトレースヘッダーを追加したり、X-Ray デーモンを実行したり、サンプリング決定を行ったり、トレースデータを X-Ray にアップロードしたり AWS X-Ray することで、 との統合を提供するものがあります。その他は AWS X-Ray SDK を使用して計測する必要があります。X-Ray 統合をまだサポートしていないものもあります。Step Functions とのサービス統合を使用する場合は、完全なトレースデータを提供するために X-Ray 統合が必要です。

ネイティブ X-Ray サポート

ネイティブ X-Ray サポートとのサービス統合には、次のものがあります。

- Amazon Simple Notification Service
- Amazon Simple Queue Service
- AWS Lambda
- AWS Step Functions

必要な計器

X-Ray 計器が必要なサービス統合:

- Amazon Elastic Container Service
- AWS Batch

- AWS Fargate

クライアント側のトレースのみ

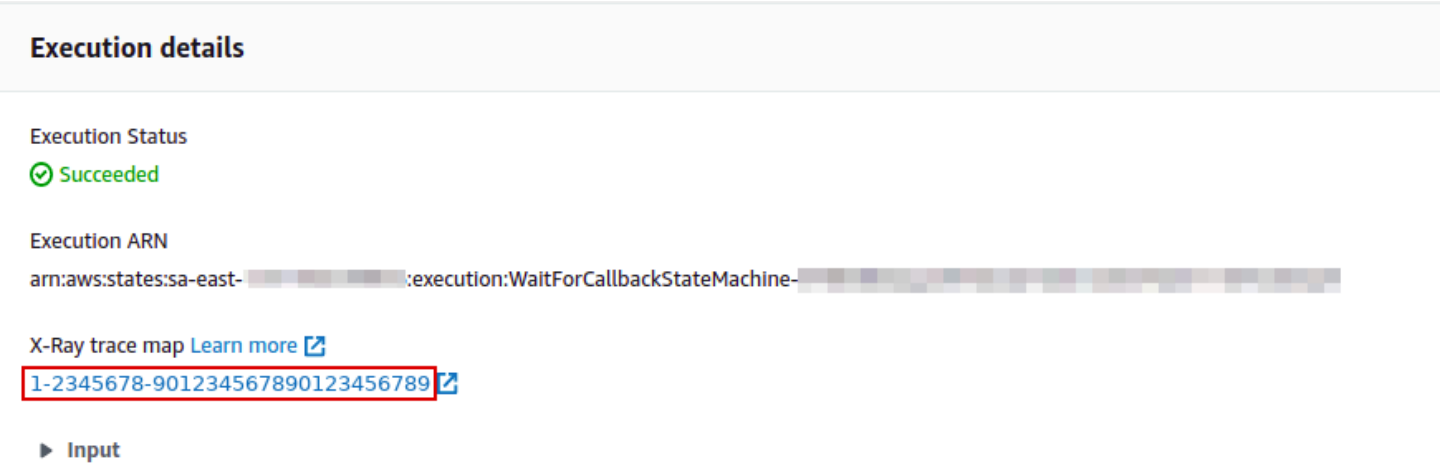
その他のサービス統合では、X-Ray のトレースはサポートされません。ただし、クライアント側のトレースは引き続き収集できます。

- Amazon DynamoDB
- Amazon EMR
- Amazon SageMaker
- AWS CodeBuild
- AWS Glue

X-Ray コンソールの表示

X-Ray は、サービスからセグメント形式でデータを受け取ります。X-Ray は、トレースへの共通リクエストを含むセグメントをグループ化します。X-Ray は、トレースを処理して、アプリケーションのビジュアル表現を提供するサービスグラフを生成します。

ステートマシンの実行をスタートしたら、[Execution details] (実行詳細) セクションで [X-Ray trace map] (X-Ray トレースマップ) を選択して X-Ray トレースを表示できます。



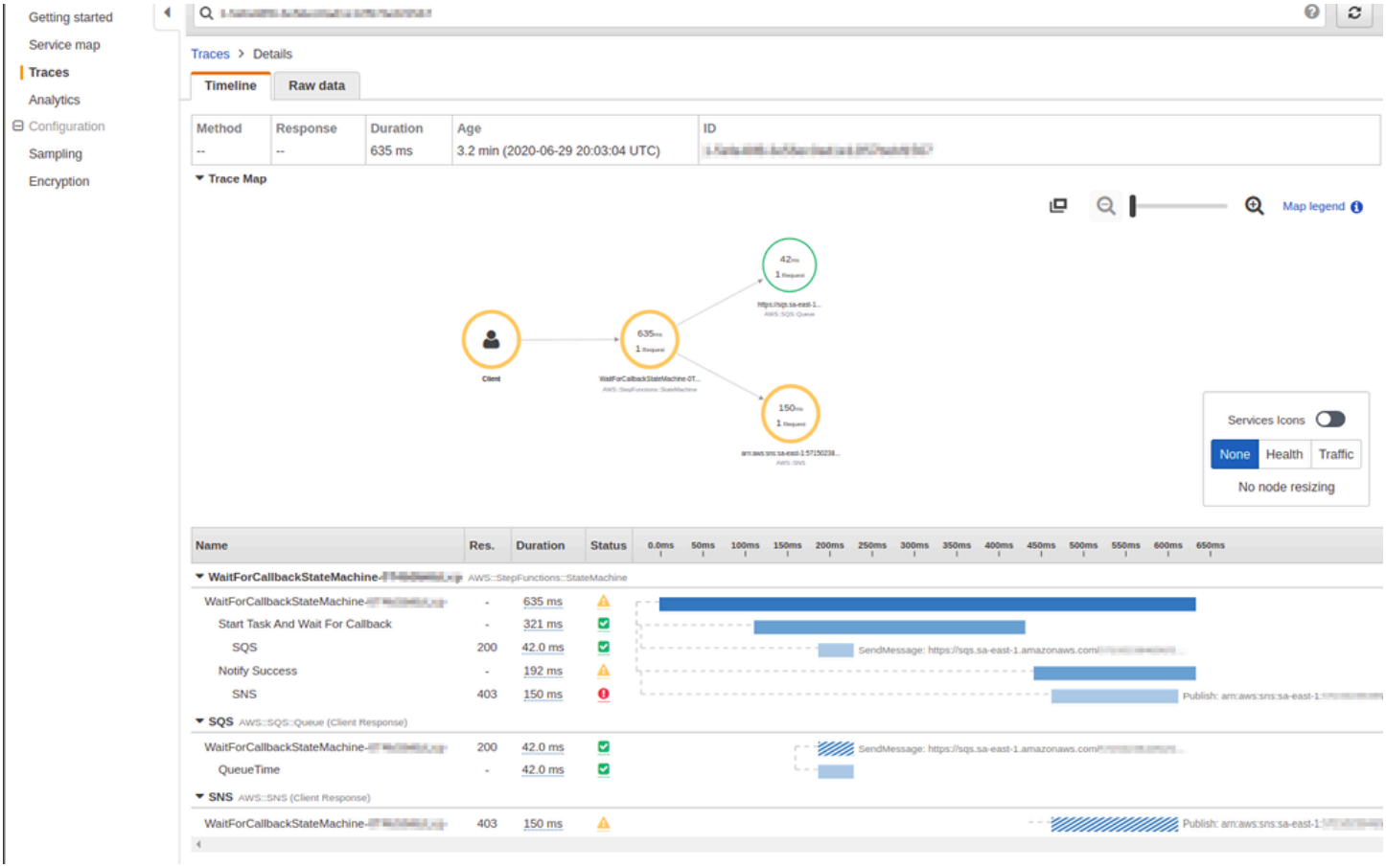
ステートマシンに対する X-Ray を有効にした後、X-Ray コンソールでその実行のトレース情報を表示できます。

Step Functions の X-Ray トレース情報を表示

次の手順は、X-Ray を有効にして実行した後にコンソールに表示できる情報の種類を示しています。コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda) サンプルプロジェクト用 X-Ray トレースが表示されます。

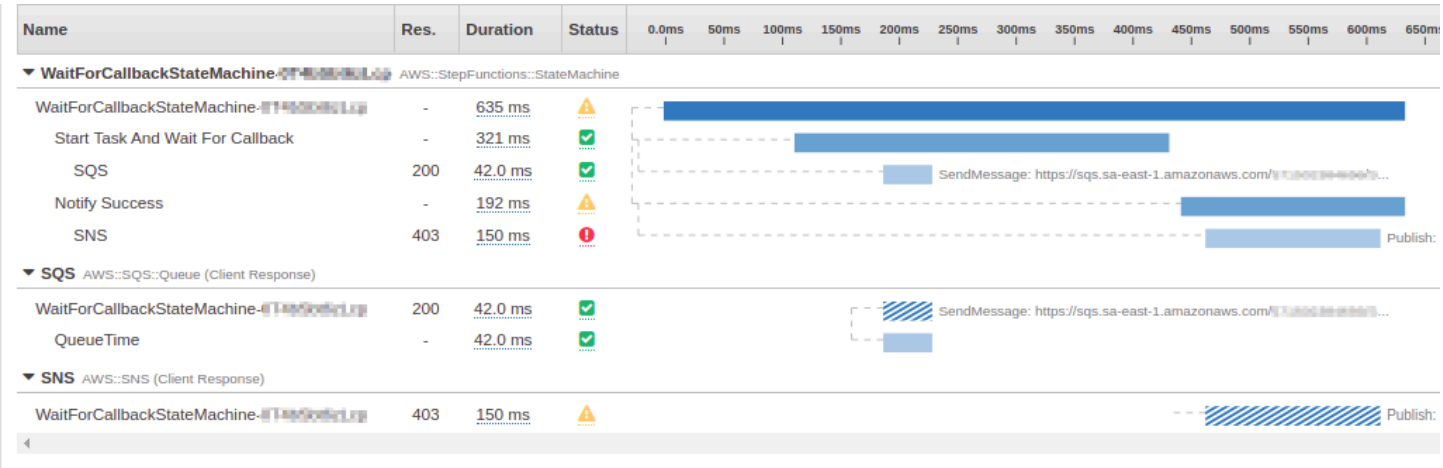
トレース

実行が完了した後、X-Ray コンソールに移動して、そこで X-Ray トレース ページが表示されます。これにより、サービスマップの概要と、ステートマシンのトレースおよびセグメント情報が表示されます。

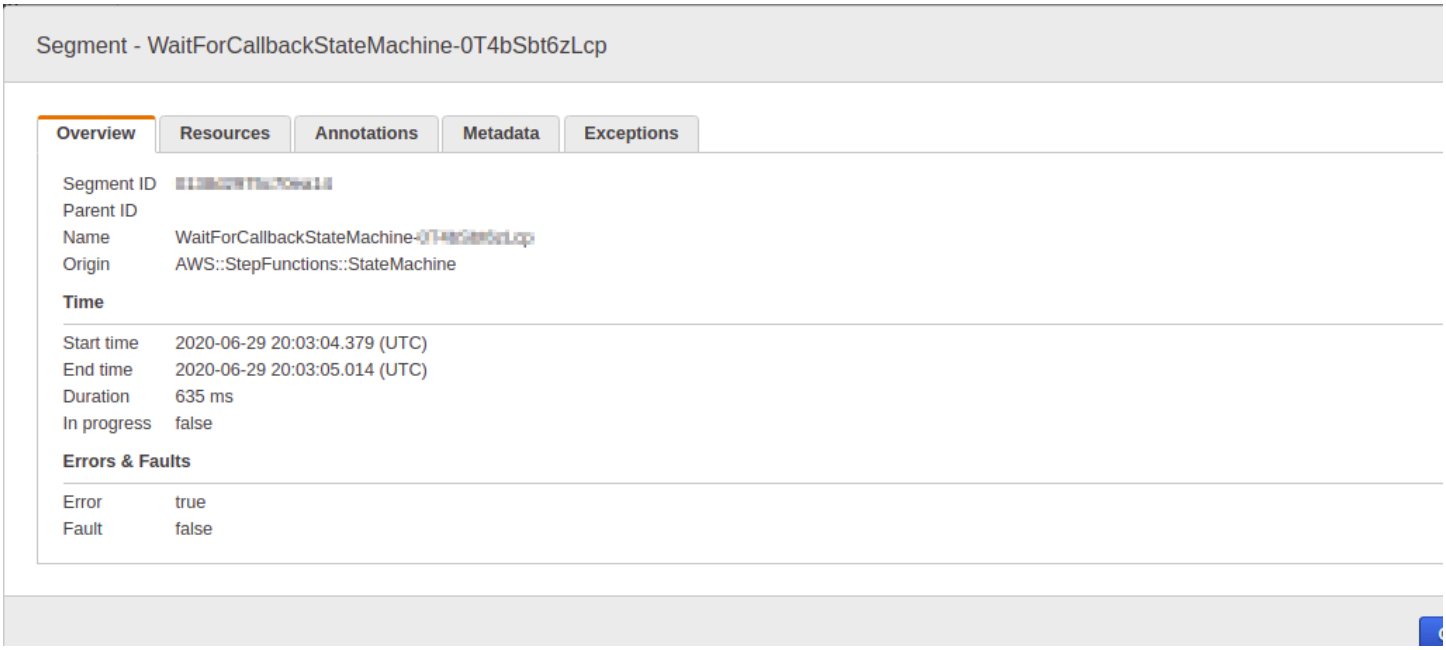


サービスマップ

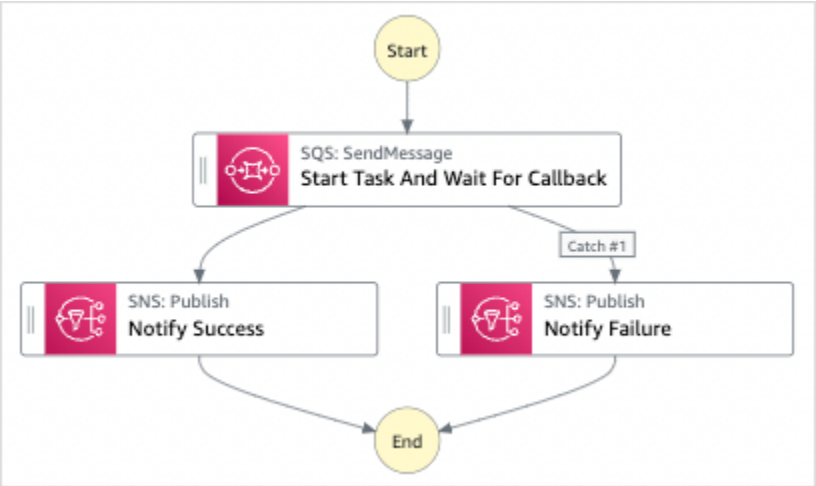
X-Ray コンソールのサービスマップは、エラーが発生しているサービス、高レイテンシーとの接続があるサービスを特定するか、不成功に終わったリクエストのトレースを識別する助けとなります。



トレースマップ上で、サービスノードを選択して、その接続を移動したリクエストを閲覧するため、そのノードまたは 2 つのノード間のエッジへのリクエストを閲覧できます。ここで、`WaitForCallBack` ノードが選択されており、その実行および応答ステータスに関する追加情報を表示できます。



X-Ray サービスマップとステートマシンとの相関関係を確認できます。X-Ray をサポートしていれば、Step Functions によって呼び出されるサービス統合ごとにサービスマップノードがあります。



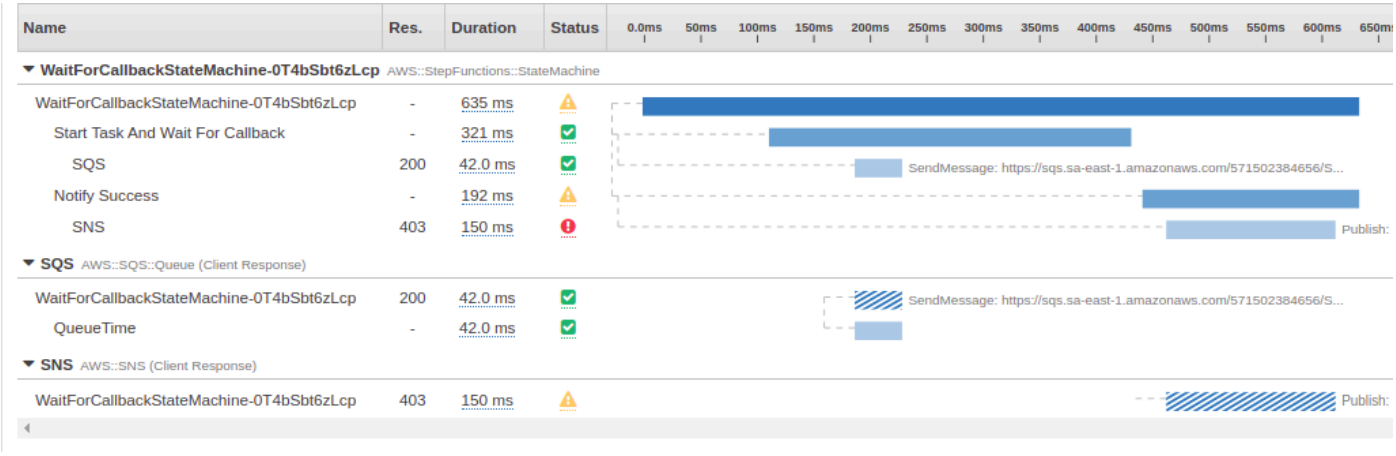
セグメントとサブセグメント

トレースは、1 つのリクエストで生成されたセグメントを収集します。各セグメントには、リソース名、リクエストの詳細、行った作業の詳細が含まれています。[Trace] (トレース) ページでは、セグメントと、拡大した場合は対応するサブセグメントが表示されます。セグメントまたはサブセグメントを選択して、その詳細を表示できます。

各タブを選択して、セグメントとサブセグメントの情報の表示方法を確認します。

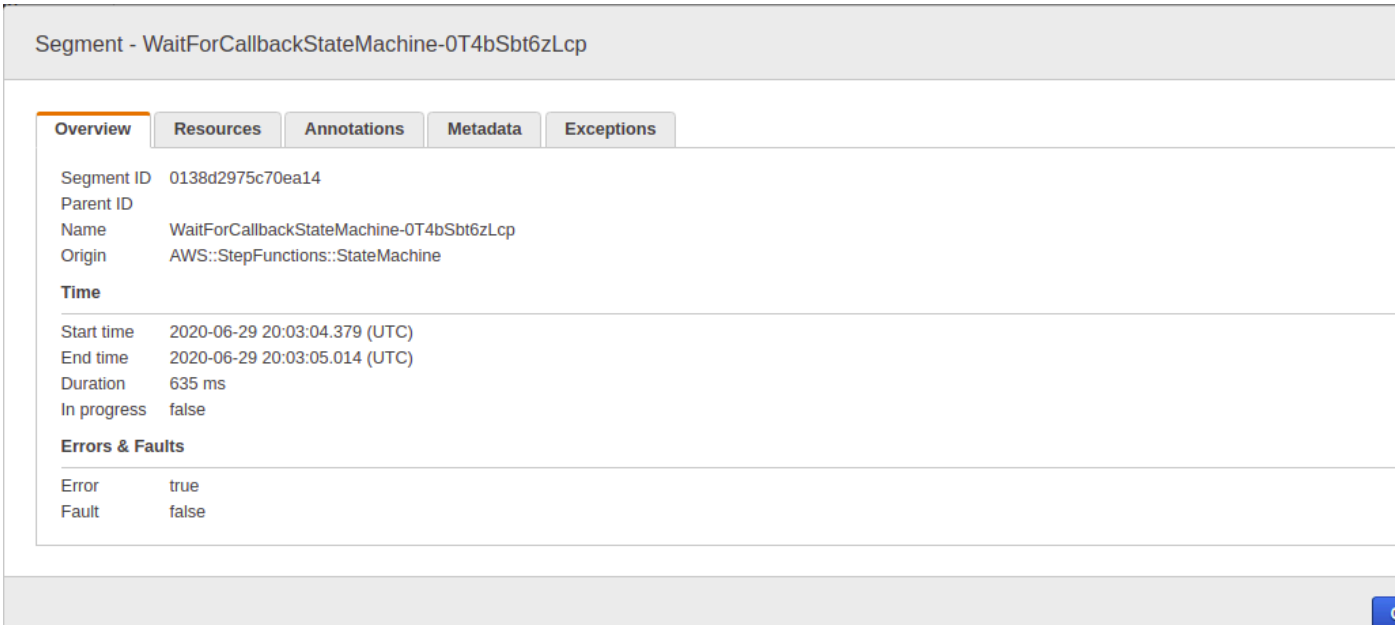
Overview of Segments

このステートマシンのセグメントとサブセグメントの概要。サービスマップ上のノードごとに別のセグメントがあります。



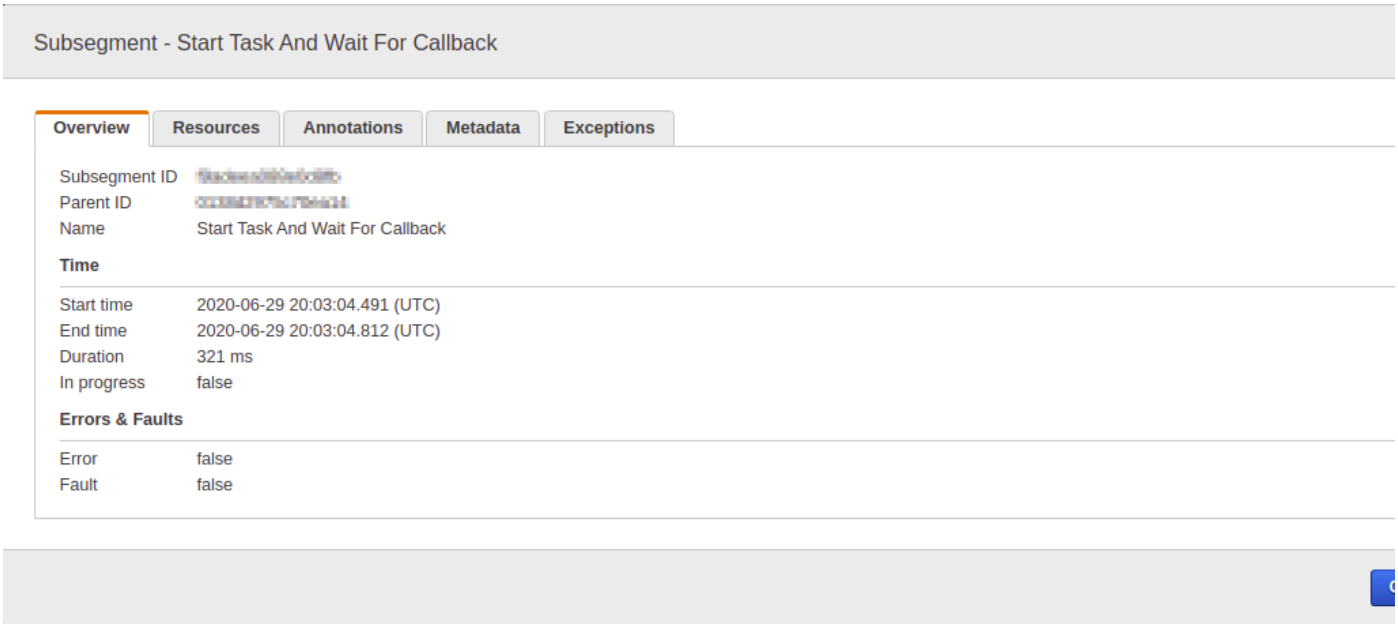
View segment detail

セグメントを選択すると、リソース名、リクエストの詳細、行った作業の詳細が含まれています。



View subsegment detail

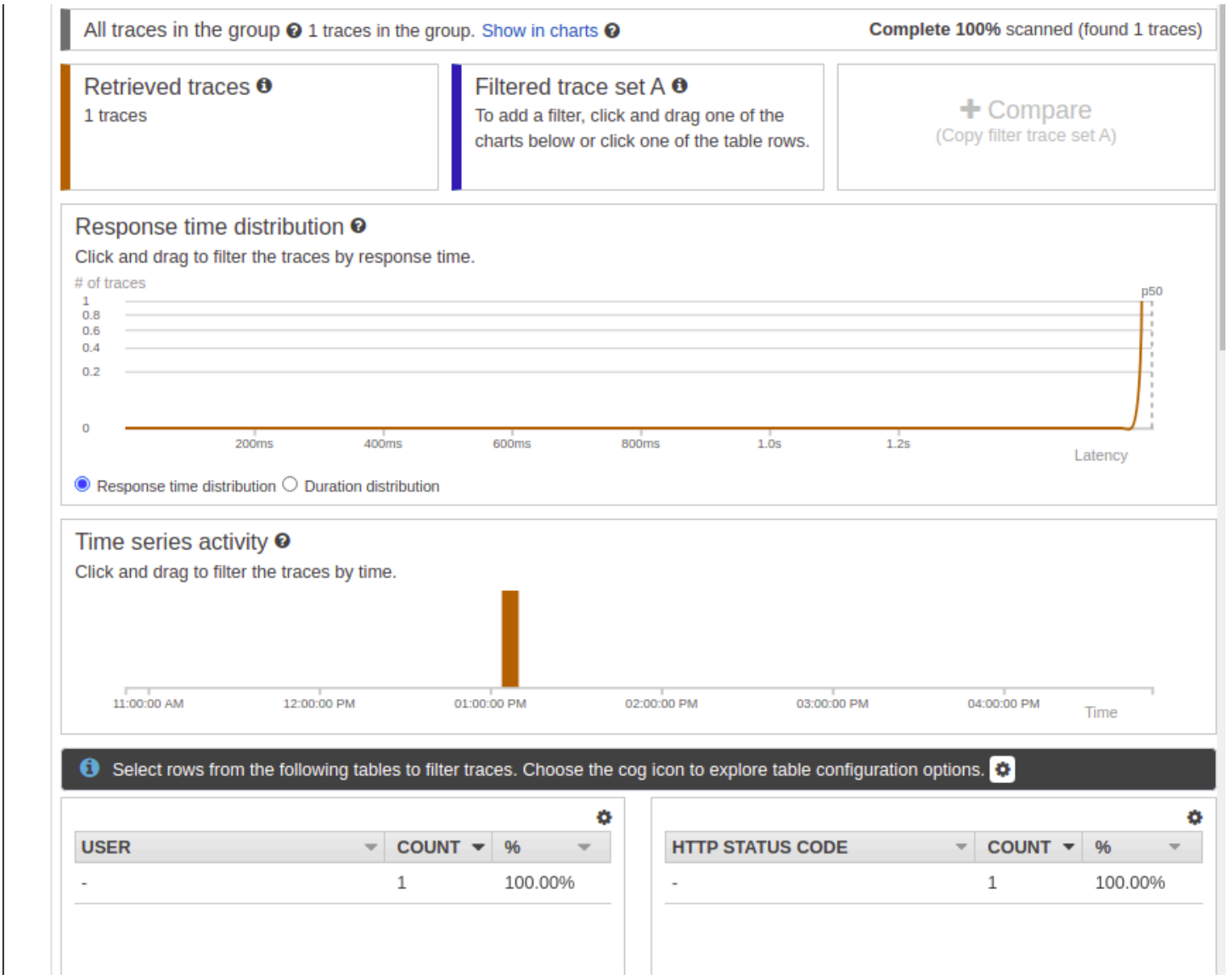
セグメントでは、完了した作業に関するデータをサブセグメントに分けることができます。サブセグメントを選択すると、より詳細なタイミング情報と詳細を表示できます。サブセグメントには、 AWS サービス、外部 HTTP API、または SQL データベースへの呼び出しに関する追加の詳細を含めることができます。



分析

AWS X-Ray Analytics コンソールは、トレースデータを解釈するためのインタラクティブなツールです。これを使用して、ステートマシンのパフォーマンスをより簡単に理解できます。このコンソールを使用すると、インタラクティブな応答時間グラフと時系列グラフを使用して、トレースを調査、分析、および視覚化できます。こうすると、パフォーマンスとレイテンシーの問題をすばやく選択しやすくなります。

現在のトレースセットに関連付けられているグラフとメトリクスおよびフィールドのパネルをクリックして、アクティブなデータセットをきめ細かく絞り込むことができます。

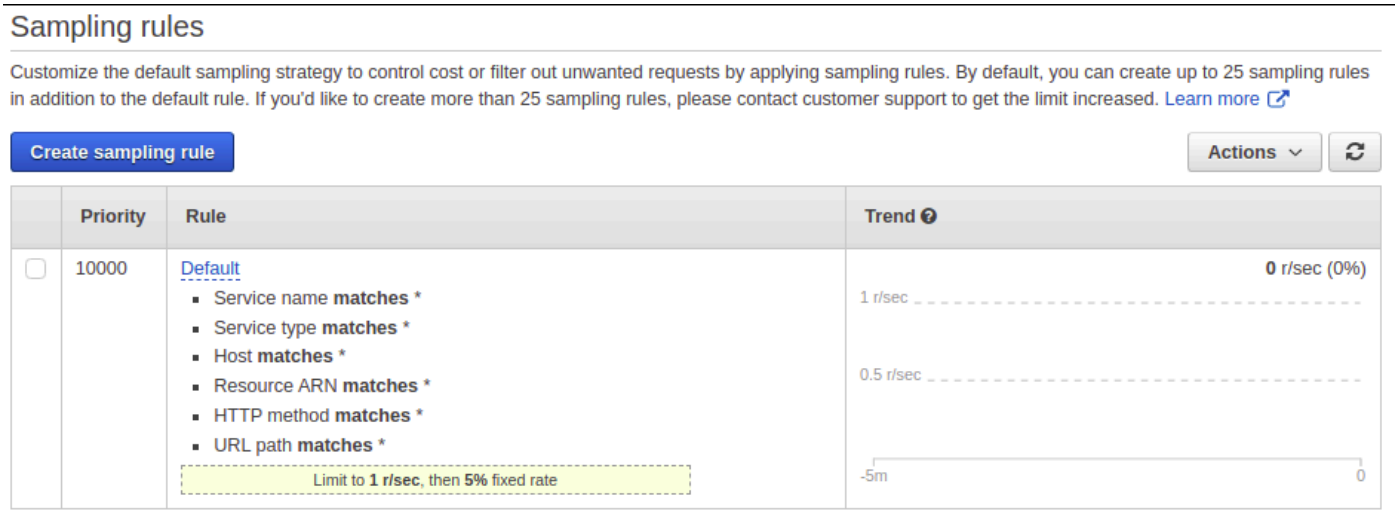


構成

X-Ray コンソールから、サンプリングと暗号化オプションを設定できます。

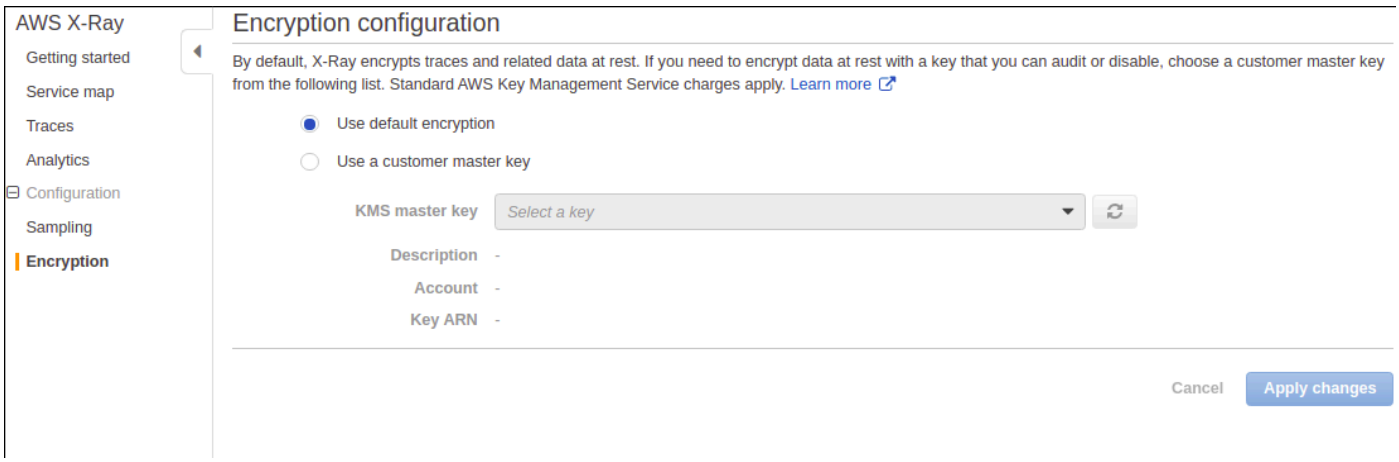
Sampling

[Sampling] (サンプリング) を選択して、サンプリングレートと設定の詳細を表示します。サンプリングルールを変更して記録するデータの量を制御し、特定の要件に合わせてサンプリング動作を変更できます。



Encryption

Encryption (暗号化) を選択して、暗号化設定を変更します。デフォルトの設定を使って、X-Ray が静止時のトレースと日付を暗号化できます。または、必要に応じてカスタマーマスターキーを選択できます。後の方法を選択した場合、スタンダード AWS KMS 料金が適用されます。



トレースマップまたはサービスマップにデータがない場合はどうなりますか。

X-Ray が有効なものの、X-Ray コンソールでデータが表示されない場合は、次の点をチェックしてください。

- IAM ロールを正しく設定して、X-Ray への書き込みが許可されるようにします。
- サンプリングルールでは、データのサンプリングが許可されています。
- 新しく作成または変更した IAM ロールが適用されるまでに短い遅延が生じる可能性があるため、数分後にトレースマップまたはサービスマップを再度チェックしてください。
- X-Ray トレースパネルに Data Not Found が表示された場合は、IAM アカウントの設定をチェックし、目的のリージョンで AWS Security Token Service が有効になっていることを確認します。詳細については、「IAM ユーザーガイド」の「 AWS STS での のアクティブ化と非アクティブ化 AWS リージョン」を参照してください。

AWS User Notificationsと使用するAWS Step Functions

AWS User Notifications を使用すると、AWS Step Functions イベントの通知を受ける配信チャネルを設定できます。指定したルールにイベントが一致すると、通知を受け取ります。イベントの通知は、E メール、AWS Chatbot チャット通知、AWS Console Mobile Application プッシュ通知など、複数のチャネルを通じて受け取ることができます。また、コンソール通知センターの通知を確認することもできます。User Notifications は集約をサポートしているため、特定のイベント中に受け取る通知の数を減らすことができます。

のセキュリティ AWS Step Functions

このセクションでは、 AWS Step Functions セキュリティと認証について説明します。

トピック

- におけるデータ保護 AWS Step Functions
- AWS Step Functionsにおけるアイデンティティとアクセスの管理
- ログ記録とモニタリング
- のコンプライアンス検証 AWS Step Functions
- レジリエンスとは AWS Step Functions
- のインフラストラクチャーセキュリティ AWS Step Functions
- の設定と脆弱性の分析 AWS Step Functions

Step Functions は IAM AWS を使用して他のサービスやリソースへのアクセスを制御します。IAM の仕組みの概要については、「IAM ユーザーガイド」の「アクセス管理の概要」を参照してください。セキュリティ認証情報の概要については、「Amazon Web Services 全般のリファレンス」の「AWS セキュリティ認証情報」を参照してください。

におけるデータ保護 AWS Step Functions

AWS のデータ保護には、https://aws.amazon.com/compliance/shared-responsibility-model/、 (責任分担モデル) が適用されます AWS Step Functions。このモデルで説明したように、 AWS は、 AWS クラウドすべてを支えるグローバルインフラストラクチャを保護する責任があります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護のため、 AWS アカウント 認証情報を保護し、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。こうすると、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、以下の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。

- SSL/TLS を使用してリソースと通信します。 AWS TLS 1.2、できれば TLS 1.3 が必要です。
- を使用して API とユーザーアクティビティのロギングを設定します。 AWS CloudTrail
- AWS 暗号化ソリューションと、 AWS のサービスその中に含まれるデフォルトのセキュリティコントロールをすべて使用してください。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介してアクセスするときに FIPS 140-2 で検証された暗号モジュールが必要な場合は、FIPS エンドポイントを使用してください。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや名前フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これには、コンソール、API AWS CLI、または AWS SDK AWS のサービス を使用して Step Functions やその他の機能を操作する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

での暗号化 AWS Step Functions

保管時の暗号化

Step Functions は、保管中のデータを常に暗号化します。 AWS Step Functions 入力されたデータは、透過的なサーバー側の暗号化を使用して保存時に暗号化されます。これは、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管時に暗号化することで、セキュリティを重視したアプリケーションを構築して、暗号化のコンプライアンスと規制の要件を満たすことができます。

転送中の暗号化

Step Functions は、他の統合された AWS のサービスとの間で転送中のデータを暗号化します (AWS Step Functions 他のサービスとの併用 を参照)。Step Functions と統合サービス間を通過するすべてのデータは、Transport layer Security (TLS) を使用して暗号化されています。

AWS Step Functionsにおけるアイデンティティとアクセスの管理

にアクセスするには、 AWS Step Functions AWS リクエストの認証に使用できる認証情報が必要です。これらの認証情報には、 AWS AWS 他のリソースからイベントデータを取得するなど、リソースにアクセスするための権限が必要です。次のセクションでは、AWS Identity and Access Management (IAM) と Step Functions を使用してリソースにアクセスできるユーザーを制御することにより、リソースをセキュリティで保護する方法について詳しく説明します。

AWS Identity and Access Management (IAM) は、 AWS のサービス AWS 管理者がリソースへのアクセスを安全に制御できるようにするものです。IAM 管理者は、誰を認証するか (サインインさせるか) と、誰に Step Functions リソースの使用を許可するか (アクセス許可を付与するか) を制御します。IAM AWS のサービス は追加料金なしで使用できるアプリです。

対象者

AWS Identity and Access Management (IAM) の使用方法は、Step Functions で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Step Functions サービスを使用する場合は、管理者から必要な許可と認証情報が与えられます。作業を実行するためにさらに多くの Step Functions 機能を使用する場合、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Step Functions の機能にアクセスできない場合は、「ID とアクセスのトラブルシューティング AWS Step Functions」を参照してください。

サービス管理者 - 社内の Step Functions リソースを担当している場合は、通常、Step Functions へのフルアクセス許可が付与されます。サービスのユーザーがどの Step Functions 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。企業における Step Functions での IAM の使用方法については、「IAM AWS Step Functions との連携方法」を参照してください。

IAM 管理者 – IAM 管理者には、Step Functions へのアクセスを管理するポリシーの作成方法の詳細を理解することが推奨されます。IAM で使用できる Step Functions アイデンティティベースのポリシーの例については、「の ID ベースのポリシーの例 AWS Step Functions」を参照してください。

アイデンティティを使用した認証

認証とは、ID AWS 認証情報を使用してサインインする方法です。IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (サインイン AWS) する必要があります。

ID ソースを通じて提供された認証情報を使用して、フェデレーション ID AWS としてサインインできます。 AWS IAM Identity Center フェデレーテッド ID の例としては、(IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google や Facebook の認証情報などがあります。フェデレーションアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。 AWS フェデレーションを使用してアクセスすると、間接的にロールを引き継ぐことになります。

ユーザーのタイプによっては、 AWS Management Console AWS またはアクセスポータルにサインインできます。へのサインインについて詳しくは AWS、『AWS サインイン ユーザーガイド』の「AWS アカウントにサインインする方法」を参照してください。

AWS プログラムでアクセスする場合は、認証情報を使用してリクエストに暗号署名するためのソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨方法を使用して自分でリクエストに署名する方法の詳細については、IAM ユーザーガイドの「 AWS API リクエストへの署名」を参照してください。

使用する認証方法を問わず、セキュリティ情報の提供を追加でリクエストされる場合もあります。たとえば、アカウントのセキュリティを強化するために多要素認証 (MFA) AWS を使用することを推奨しています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント root ユーザー

を作成するときは AWS アカウント、 AWS のサービス アカウント内のすべてのリソースに完全にアクセスできる 1 つのサインイン ID から始めます。この ID は AWS アカウント root ユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでサインインすることでアクセスされます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーション ID

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、ID AWS のサービス プロバイダーとのフェデレーションを使用して一時的な認証情報を使用してアクセスするように要求します。

フェデレーテッド ID とは、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、Identity Center ディレクトリのユーザー、または ID AWS のサービス ソースを通じて提供された認証情報を使用してアクセスする任意のユーザーです。 AWS Directory Serviceフェデレーテッド ID がアクセスすると AWS アカウント、そのユーザーがロールを引き受け、そのロールが一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成したり、独自のアイデンティティソース内のユーザーやグループに接続して同期したりして、すべてのアプリケーションで使用することができます。 AWS アカウント IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「IAM Identity Center とは？」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたはアプリケーションに対して特定の権限を持つ社内の AWS アカウント ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループ は、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、 AWS アカウント 特定の権限を持つ社内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。 AWS Management Console ロールを切り替えることで、の IAM ロールを一時的に引き受けることができます。 AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用してロールを引き受けることができます。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーションアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、ロールをプロキシとして使用する代わりに AWS のサービス、ポリシーをリソースに直接アタッチできるものもあります。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — AWS のサービス AWS のサービス他の機能を使用するものもあります。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) — IAM ユーザーまたはロールを使用してアクションを実行する場合 AWS、あなたはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS

は、 AWS のサービス を呼び出したプリンシパルの権限をリクエスト元と組み合わせて使用して AWS のサービス、ダウンストリームサービスにリクエストを行います。FASリクエストは、AWS のサービス サービスが他のユーザーとのやりとりやリソースとのやり取りを必要とするリクエストを受信したときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール — サービスにリンクされたロールは、にリンクされているサービスロールの一種です。 AWS のサービスサービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。 AWS アカウント サービスにリンクされたロールはに表示され、そのサービスが所有します。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されるアプリケーション — IAM ロールを使用して、EC2 インスタンスで実行され、 AWS API AWS CLI リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 AWS インスタンスにロールを割り当て、そのロールをそのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされるインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか」を参照してください。

ポリシーを使用したアクセスの管理

AWS ポリシーを作成して AWS ID またはリソースにアタッチすることで、アクセスを制御します。ポリシーとは、ID またはリソースに関連付けると権限を定義するオブジェクトです。 AWS AWS プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON AWS ドキュメントとして保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザは AWS Management Console、、 AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらに インラインポリシー または マネージドポリシー に分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。 AWS アカウント管理ポリシーには、 AWS 管理ポリシーと顧客管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザ、ロール、フェデレーティッドユーザ、またはを含めることができます。 AWS のサービス

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。IAM AWS の管理ポリシーをリソースベースのポリシーで使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

ACL をサポートするサービスの例としては AWS WAF、Amazon S3、、および Amazon VPC があります。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS あまり一般的ではないポリシータイプもサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる許可の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCP) — SCP は、組織または組織単位 (OU) の最大権限を指定する JSON ポリシーです。 AWS Organizations AWS Organizations は、 AWS アカウント 企業が所有する複数のものをグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、メンバーアカウントのエンティティ (各エンティティを含む) の権限を制限します。 AWS アカウントのルートユーザー Organizations と SCP の詳細については、AWS Organizations ユーザーガイド の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーティッドユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限される範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合

もあります。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。詳細については、 IAM ユーザーガイド の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。 AWS 複数のポリシータイプが関係している場合にリクエストを許可するかどうかを決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

アクセスコントロール

有効な認証情報があればリクエストを認証できますが、アクセス許可が付与されている場合を除き、Step Functions リソースの作成やアクセスはできません。たとえば、Step Functions ルールに関連付けられた Amazon Simple Notification Service (Amazon SNS) および Amazon Simple Queue Service (Amazon SQS) AWS Lambdaターゲットを呼び出すためのアクセス権限が必要です。

以下のセクションでは、Step Functions の許可の管理方法について説明します。

- ステートマシンの IAM ロールの作成
- 管理者以外のユーザー用の詳細な IAM 許可の作成
- Step Functions の Amazon VPC エンドポイント
- 統合サービスの IAM ポリシー
- 分散マップ状態を使用するための IAM ポリシー

Step Functions のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションには通常、関連する AWS API オペレーションと同

じ名前が付けられます。一致する API オペレーションのない 許可のみのアクション など、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連するオペレーションを実行するための許可を付与するポリシーで使用されます。

Step Functions アクションのリストを確認するには、「サービス認可リファレンス」の「AWS Step Functionsで定義されるリソース」を参照してください。

Step Functions のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
states
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "states:action1",
      "states:action2"
         ]
```

Step Functions でのアイデンティティベースのポリシーの例については、「の ID ベースのポリシーの例 AWS Step Functions」を参照してください。

Step Functions のポリシーリソース

ポリシーリソースに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシーの要素は、オブジェクトあるいはアクションが適用されるオブジェクトを指定します。ステートメントには、`Resource` または `NotResource`要素を含める必要があります。ベストプラクティスとしては、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、 リソースレベルの許可 と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

ACM リソースのタイプとその ARN のリストを確認するには、「サービス認可リファレンス」の「AWS Step Functionsで定義されるアクション」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「AWS Step Functionsで定義されるアクション」を参照してください。

Step Functions でのアイデンティティベースのポリシーの例については、「の ID ベースのポリシーの例 AWS Step Functions」を参照してください。

Step Functions の条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どの プリンシパル がどの リソース に対してどのような条件下 で アクション を実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition要素を指定する場合、または 1 つの Condition要素に複数のキーを指定する場合、 AWS では AND論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 AWS OR論理演算を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、 IAM ユーザーガイド の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS グローバル条件キーとサービス固有の条件キーをサポートします。 AWS すべてのグローバル条件キーを確認するには、IAM ユーザーガイドの「AWS グローバル条件コンテキストキー」を参照してください。

Step Functions の条件キーのリストを確認するには、「サービス認可リファレンス」の「AWS Step Functionsの条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Resources Defined by AWS Step Functions」を参照してください。

Step Functions でのアイデンティティベースのポリシーの例については、「の ID ベースのポリシーの例 AWS Step Functions」を参照してください。

Step Functions の ACL

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Step Functions と共に ABAC を使用

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいて権限を定義する認可戦略です。では AWS、これらの属性はタグと呼ばれます。IAM エンティティ (ユーザーまたはロール) AWS や多くのリソースにタグを付けることができます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。次に、プリンシパルのタグがアクセスを試行するリソースのタグと一致したときにオペレーションを許可するよう、ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys`の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値は Yes です。サービスが一部のリソースタイプに対してのみ 3 つの条件キーすべてをサポートする場合、値は Partial です。

ABAC の詳細については、IAM ユーザーガイドの「ABAC とは？」を参照してください。ABAC の設定手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

Step Functions での一時的な認証情報の使用

一時的な認証情報のサポート	Yes

AWS のサービス 一時的な認証情報を使用してサインインすると機能しないものもあります。 AWS のサービス 一時的な認証情報で機能するものなど、追加情報については、『IAM ユーザーガイド』の「IAM と連携する」を参照してくださいAWS のサービス 。

ユーザー名とパスワード以外の方法でにサインインすると、 AWS Management Console 一時的な認証情報が使用されることになります。たとえば、会社のシングルサインオン (SSO) AWS リンクを使用してアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、 IAM ユーザーガイド の「ロールへの切り替え (コンソール)」を参照してください。

または API を使用して一時的な認証情報を手動で作成できます。 AWS CLI AWS その後、その一時的な認証情報を使用してアクセスできます AWS。 AWS 長期アクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをおすすめします。詳細については、「IAM の一時的なセキュリティ認証情報」を参照してください。

Step Functions のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	Yes

IAM ユーザーまたはロールを使用してアクションを実行する場合 AWS、そのユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FASは、を呼び出したプリンシパルの権限と AWS のサービス、 AWS のサービス ダウンストリームサービスにリクエストを行うリクエストを組み合わせて使用します。FASリクエストは、 AWS のサービス サービスが他のユーザーとのやりとりやリソースとのやり取りを必要とするリクエストを受信したときにのみ行われます。この場合、両

方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Step Functions のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、Step Functions の機能が阻害される可能性があります。Step Functions が指示する場合以外は、サービスロールを編集しないでください。

Step Functions のサービスにリンクされたロール

サービスにリンクされたロールのサポート	いいえ

サービスにリンクされたロールは、にリンクされているサービスロールの一種です。 AWS のサービスサービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。AWS アカウント サービスにリンクされたロールはに表示され、そのサービスが所有します。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

サービスリンクロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の中から、サービスにリンクされたロール 列に Yesと記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

IAM AWS Step Functions との連携方法

IAM を使用して Shield へのアクセスを管理する前に、Shield で利用できる IAM の機能について説明します。

で使用できる IAM 機能 AWS Step Functions

IAM の機能	Step Functions のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	いいえ
ポリシーアクション	Yes
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	あり
サービスリンクロール	No

Step Functions AWS やその他のサービスがほとんどの IAM 機能でどのように機能するかを大まかに把握するには、IAM ユーザーガイドの「IAM AWS と連携するサービス」を参照してください。

の ID ベースのポリシーの例 AWS Step Functions

デフォルトでは、ユーザーおよびロールには、Step Functions リソースを作成または変更するアクセス許可がありません。また、、 AWS Command Line Interface (AWS CLI) AWS Management Console、API AWS を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらの JSON ポリシードキュメント例を使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

Step Functions が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「AWS Step Functionsのアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Step Functions コンソールを使用する。
- 自分の許可の表示をユーザーに許可する

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウントで誰かが Step Functions リソースを作成、アクセス、または削除できるかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースのポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください。

- AWS 管理ポリシーから始めて、最小権限の権限に移行する — ユーザーとワークロードへのアクセス権限の付与を開始するには、AWS 多くの一般的なユースケースで権限を付与する管理ポリシーを使用してください。これらのポリシーは、で利用できます。 AWS アカウント AWS ユースケースに固有のカスタマー管理ポリシーを定義して、権限をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定するときは、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、 最小特権アクセス許可 とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。サービスアクションがなどの特定の用途で使用された場合は AWS のサービス、条件を使用してサービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは

100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。

- 多要素認証 (MFA) が必要 — IAM ユーザーまたは root ユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA をオンにしてください。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

Step Functions コンソールを使用する。

AWS Step Functions コンソールにアクセスするには、最低限の権限が必要です。これらの権限により、内の Step Functions リソースの詳細を一覧表示および表示できる必要があります AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最低限のコンソール権限を与える必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Step Functions コンソールを使用できるようにするには、Step Functions *ConsoleAccess* *ReadOnly* AWS または管理ポリシーをエンティティにアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへの許可の追加」を参照してください。

自分の許可の表示をユーザーに許可する

この例では、ユーザーアイデンティティに添付されたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーを作成する方法を示します。このポリシーには、コンソールで、またはまたは API を使用してこのアクションをプログラムで完了するための権限が含まれています。 AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Step Functions のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について知るには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Step Functions のアイデンティティベースのポリシーの例

Step Functions でのアイデンティティベースのポリシーの例については、「の ID ベースのポリシーの例 AWS Step Functions」を参照してください。

Step Functions 内のリソースベースのポリシー

リソースベースのポリシーのサポート	なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、またはを含めることができます。 AWS のサービス

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス権限を付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

ステートマシンの IAM ロールの作成

AWS Step Functions コードを実行したり、 AWS リソースにアクセスしたりできる (AWS Lambda 関数の呼び出しなど)。セキュリティを維持するために、IAM ロールを使用してこれらのリソースへの Step Functions アクセスを許可する必要があります。

Step Functions チュートリアルこのガイドでは、 AWS ステートマシンを作成したリージョンで有効な、自動生成された IAM ロールを活用できます。ただし、ステートマシンの IAM ロールを独自に作成することもできます。

ステートマシンの IAM ポリシーを作成する場合、そのポリシーにはステートマシンに付与する許可を含める必要があります。 AWS 既存の管理ポリシーを例として使用することも、特定のニーズを満たすカスタムポリシーを最初から作成することもできます。詳細については、「IAM ユーザーガイド」 の「IAM ポリシーの作成」を参照してください。

ステートマシンの IAM ロールを独自に作成するには、このセクションの手順に従います。

この例では、Lambda 関数を呼び出す許可を持つIAM ロールを作成します。

Step Functions のロールの作成

1. IAM コンソールにサインインし、[ロール]、[ロールの作成] の順に選択します。
2. [信頼できるエンティティ] ページの [AWS サービス] で、リストから [Step Functions] を選択し、[次へ：許可] を選択します。
3. [アタッチされた許可ポリシー] ページで、[次へ：確認] を選択します。
4. [確認] ページで、`StepFunctionsLambdaRole` を [ロール名] に入力し、[ロールの作成] を選択します。

   ロールのリストで、IAM ロールが表示されます。

IAM 許可とポリシーの詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

サービス間の混乱した代理の防止

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より権限のあるエンティティにアクションの実行を強制できるセキュリティ上の問題です。では AWS、サービス間のなりすましによって、混乱した代理人問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すと

きに発生する場合もあります。この種のなりすましは、クロスアカウントおよびクロスサービスで発生するおそれがあります。呼び出し元サービスが操作され、それ自身のアクセス許可が利用されて、本来アクセス許可が付与されない形で、別のユーザーのリソースに働きかけることがあります。

代理人が混乱するのを防ぐために、 AWS アカウント内のリソースへのアクセス権が付与されているサービスプリンシパルを使用するすべてのサービスのデータを保護するのに役立つツールが用意されています。このセクションでは、特定のサービス間での混乱した代理人防止に焦点を当てていますが、このトピックの詳細については、IAM ユーザーガイドの「混乱した代理人問題」セクションを参照してください AWS Step Functions。

リソースポリシーの aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、Step Functions が別のサービスに付与する許可を、お使いのリソースに限定するようお勧めします。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

混乱した代理問題から保護するために最も効果的なのは、リソースの完全な ARN で aws:SourceArn グローバル条件コンテキストキーを使用する方法です。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、ARN の未知部分を表すワイルドカード文字 (*) を使って、グローバルコンテキスト条件キー aws:SourceArn を使用します。例えば arn:aws:states:*:111122223333:* です。

混乱した代理問題を防止するために、信頼できるポリシーの中で Step Functions と aws:SourceArn および aws:SourceAccount を使用する例を、以下に示します。

```
{
  "Version":"2012-10-17",
  "Statement":[
     {
        "Effect":"Allow",
        "Principal":{
           "Service":[
              "states.amazonaws.com"
           ]
        },
        "Action":"sts:AssumeRole",
        "Condition":{
           "ArnLike":{
              "aws:SourceArn":"arn:aws:states:us-east-1:111122223333:stateMachine:*"
           },
           "StringEquals":{
```

```
                "aws:SourceAccount":"111122223333"
            }
        }
    }
  ]
}
```

インラインポリシーをアタッチする

Task 状態の他のサービスを Step Functions で直接制御できます。制御する必要があるサービスの API アクションに Step Functions がアクセスできるように、インラインポリシーをアタッチします。

1. IAM コンソール を開いて [ロール] を選択し、Step Functions ロールを検索したら、そのロールを選択します。
2. [インラインポリシーの追加] を選択します。
3. ロールのポリシーを作成するには、[ビジュアルエディタ] か [JSON] タブを使用します。

AWS Step Functions AWS 他のサービスを制御する方法の詳細については、を参照してください。AWS Step Functions 他のサービスとの併用

> **Note**
>
> Step Functions コンソールで作成した IAM ポリシーの例については、統合サービスの IAM ポリシー を参照してください。

管理者以外のユーザー用の詳細な IAM 許可の作成

IAM のデフォルトの管理ポリシー (など) はReadOnly、 AWS Step Functions すべての種類のアクセス権限を網羅しているわけではありません。このセクションでは、これらさまざまなタイプのアクセス許可について説明し、設定例を示します。

Step Functions には、4 つのカテゴリの許可があります。ユーザーに許可するアクセスに応じたカテゴリのアクセス許可の使用によって、アクセスを制御できます。

サービスレベルのアクセス許可

特定のリソースで動作しない API コンポーネントに適用されます。

ステートマシンレベルのアクセス許可

特定のステートマシンで動作するすべての API コンポーネントに適用されます。

実行レベルのアクセス許可

特定の実行で動作するすべての API コンポーネントに適用されます。

アクティビティレベルのアクセス許可

特定のアクティビティまたはその特定のインスタンスで動作するすべての API コンポーネントに適用されます。

サービスレベルのアクセス許可

このアクセス許可レベルは、特定のリソースで動作しないすべての API アクションに適用されます。例えば、CreateStateMachine、CreateActivity、ListStateMachines、ListActivities などです。

```
{
  "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"
      ]
```

```
    }
  ]
}
```

ステートマシンレベルのアクセス許可

このアクセス許可レベルは、特定のステートマシンで動作するすべての API アクションに適用されます。DeleteStateMachine、DescribeStateMachine、StartExecution、ListExecutions などのリクエストの一部として、これらの API オペレーションにはステートマシンの Amazon リソースネーム (ARN) が必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeStateMachine",
        "states:StartExecution",
        "states:DeleteStateMachine",
        "states:ListExecutions",
        "states:UpdateStateMachine",
        "states:TestState",
        "states:RevealSecrets"
      ],
      "Resource": [
        "arn:aws:states:*:*:stateMachine:StateMachinePrefix*"
      ]
    }
  ]
}
```

実行レベルのアクセス許可

このアクセス許可レベルは、特定の実行で動作するすべての API アクションに適用されます。DescribeExecution、GetExecutionHistory、StopExecution などのリクエストの一部として、これらの API オペレーションには実行の ARN が必要です。

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:DescribeStateMachineForExecution",
        "states:GetExecutionHistory",
        "states:StopExecution"
      ],
      "Resource": [
        "arn:aws:states:*:*:execution:*:ExecutionPrefix*"
      ]
    }
  ]
}
```

アクティビティレベルのアクセス許可

このアクセス許可レベルは、特定のアクティビティまたはその特定のインスタンスで動作するすべての API アクションに適用されます。DeleteActivity、DescribeActivity、GetActivityTask、SendTaskHeartbeat などのリクエストの一部として、これらの API オペレーションにはアクティビティの ARN かインスタンスのトークンが必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeActivity",
        "states:DeleteActivity",
        "states:GetActivityTask",
        "states:SendTaskHeartbeat"
      ],
      "Resource": [
        "arn:aws:states:*:*:activity:ActivityPrefix*"
      ]
    }
  ]
}
```

AWS アカウント ワークフロー内の他のリソースへのアクセス

Step Functions は、 AWS アカウント ワークフロー内のさまざまな設定で設定されたリソースへのクロスアカウントアクセスを提供します。Step Functions サービスインテグレーションを使用すると、 AWS AWS のサービス リソースベースのポリシーやクロスアカウントコールをサポートしていない場合でも、あらゆるクロスアカウントリソースを呼び出すことができます。

たとえば、開発とテストという名前の 2 AWS アカウントつのリソースを同じ場所に所有していると仮定します。 AWS リージョンクロスアカウントアクセスを使用すると、「Development」アカウントのワークフローは、「Testing」アカウントで使用可能な Amazon S3 バケット、Amazon DynamoDB テーブル、Lambda 関数などのリソースにアクセスできます。

> **Important**
>
> IAM ロールとリソースベースのポリシーは、単一のパーティション内のアカウント間でのみアクセスを委任します。例えば、標準 aws パーティションの米国西部 (北カリフォルニア) と、aws-cn パーティションの中国 (北京) にアカウントがあるとします。中国 (北京) アカウントの Amazon S3 リソースベースのポリシーを使用して、標準 aws アカウントのユーザーにアクセスを許可することはできません。

クロスアカウントアクセスの詳細については、「IAM ユーザーガイド」の「クロスアカウントポリシーの評価論理」を参照してください。

AWS アカウント それぞれが独自のリソースを完全に制御できますが、Step Functions を使用すると、コードをカスタマイズしなくても、ワークフロー内のステップを再編成、交換、追加、または削除できます。これは、プロセスが変わったりアプリケーションが進化したりしても可能です。

また、ネストしたステートマシンの実行を呼び出して、さまざまなアカウントで利用可能にすることもできます。これにより、ワークフローを効率的に分離して切り離すことができます。異なるアカウントの別の Step Functions ワークフローにアクセスするワークフローの中で、`.sync` サービス統合パターンを使用する場合、Step Functions のポーリングは自身の割り当てクオータを消費します。詳細については、「ジョブの実行 (.sync)」を参照してください。

> **Note**
>
> 現在、クロスリージョン AWS SDK AWS インテグレーションとクロスリージョンリソースアクセスは Step Functions では使用できません。

コンテンツ

- このトピックの主要概念
- クロスアカウントリソースの呼び出し
- チュートリアル: AWS クロスアカウントリソースへのアクセス
- .sync 統合パターンへのクロスアカウントアクセス

このトピックの主要概念

実行ロール

Step Functions がコードを実行したり、 AWS Lambda 関数の Invoke AWS アクションなどのリソースにアクセスしたりするために使用する IAM ロール。

サービス統合

Taskワークフロー内の状態内から呼び出すことができる AWS SDK 統合 API アクション。

ソースアカウント

ステートマシンを所有し、実行を開始したアン AWS アカウント 。

ターゲットアカウント

AWS アカウント An に対してクロスアカウント呼び出しを行います。

ターゲットロール

ターゲットアカウントが所有するリソースへの呼び出しを行う際に、ステートマシンが引き受けるターゲットアカウントの IAM ロール

ジョブの実行 (.sync)

AWS Batchなどのサービスの呼び出しに使用されるサービス統合パターン。またこれにより、Step Functions のステートマシンはジョブの完了まで待機してから次の状態に進みます。Step Functions の待機を指示するには、Task 状態の Resource フィールドに .sync サフィックスを追加します。

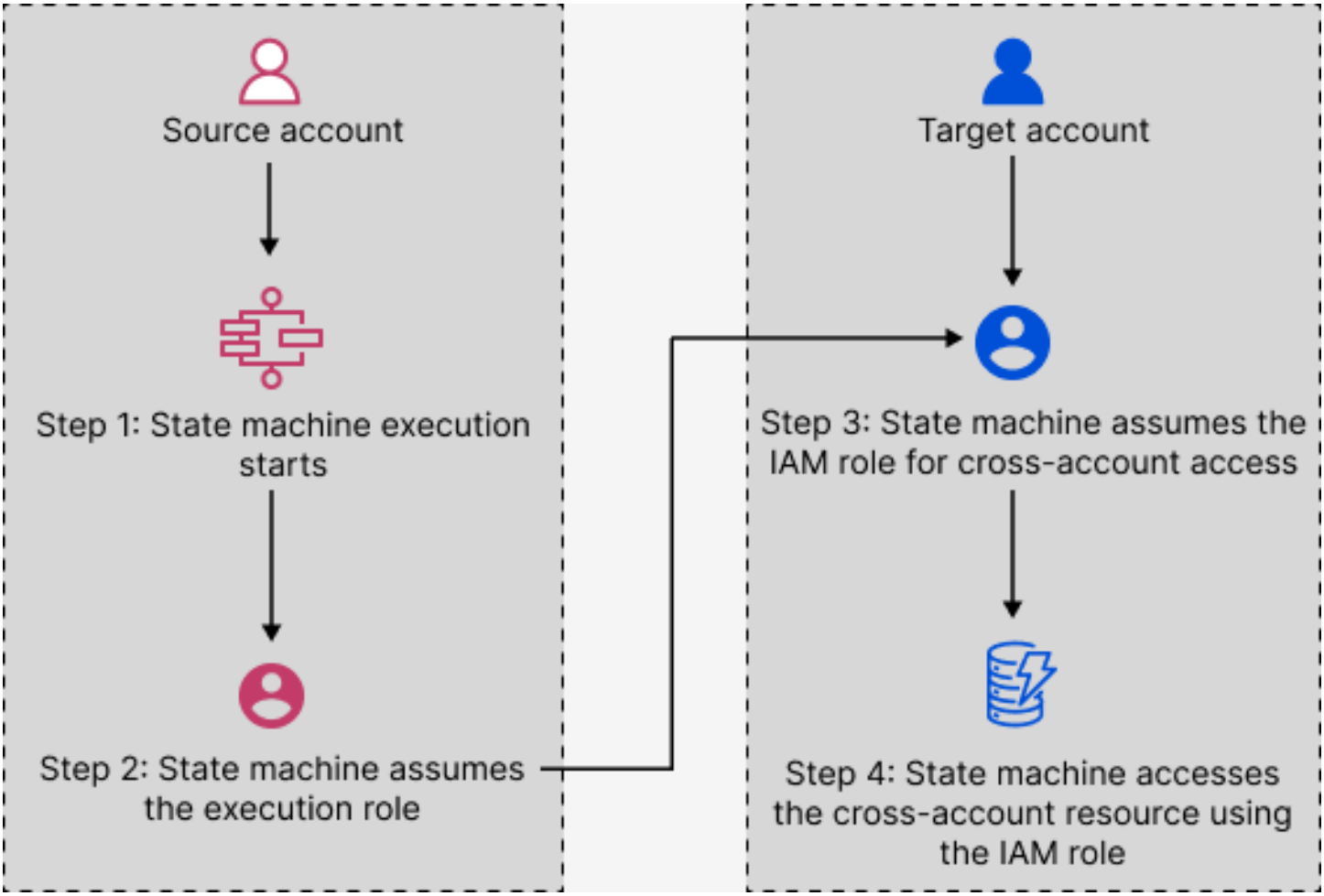
クロスアカウントリソースの呼び出し

ワークフローで、クロスアカウントリソースを呼び出すには次の操作を行います。

1. リソースを含むターゲットアカウントに IAM ロールを作成します。このロールは、ステートマシンを含むソースアカウントに、ターゲットアカウントのリソースにアクセスするアクセス許可を付与します。
2. Task ステートの定義で、クロスアカウントリソースを呼び出す前にステートマシンが引き受けるターゲット IAM ロールを指定します。
3. ターゲット IAM ロールの信頼ポリシーを変更して、ソースアカウントがこのロールを一時的に引き受けることができるようにします。信頼ポリシーには、ソースアカウントで定義したステートマシンの Amazon リソースネーム (ARN) を含める必要があります。また、 AWS リソースを呼び出すための適切な権限をターゲット IAM ロールに定義します。
4. ソースアカウントの実行ロールを更新して、ターゲット IAM ロールを引き受けるのに必要なアクセス許可を含めます。

例については「チュートリアル: AWS クロスアカウントリソースへのアクセス」を参照してください。

> **Note**
>
> 複数の AWS アカウントからリソースにアクセスするための IAM ロールを引き受けるようにステートマシンを設定できます。ただし、ステートマシンが一度に引き受けられる IAM ロールは、1 つだけです。



チュートリアル: AWS クロスアカウントリソースへのアクセス

Step Functions はクロスアカウントアクセスをサポートしており、異なる AWS アカウントで設定されたリソースの共有が可能です。このチュートリアルでは、「Production」というアカウントで定義されたクロスアカウント Lambda 関数にアクセスする手順について説明します。この関数は「Development」というアカウントのステートマシンから呼び出されます。このチュートリアルでは、「Development」アカウントを「ソースアカウント」、「Production」アカウントを「ターゲット IAM ロールを含むターゲットアカウント」と呼びます。

クロスアカウントの Lambda 関数を呼び出す前に、まず Task 状態の定義で、ステートマシンが引き受ける必要があるターゲット IAM ロールを指定します。次に、ターゲット IAM ロールの信頼ポリシーを変更して、ソースアカウントがターゲットロールを一時的に引き受けられるようにします。また、 AWS リソースを呼び出すには、ターゲット IAM ロールに適切な権限を定義します。最後に、ソースアカウントの実行ロールを更新して、ターゲットロールの引き受けに必要な許可を指定します。

複数の AWS アカウントからリソースにアクセスするための IAM ロールを引き受けるように、ステートマシンを設定できます。ただし、Task 状態の定義に基づいて、ステートマシンが一度に引き受けられる IAM ロールは、1 つだけです。

> **Note**
>
> 現在、クロスリージョン AWS SDK AWS インテグレーションとクロスリージョンリソースアクセスは Step Functions では使用できません。

コンテンツ

- 前提条件
- ステップ 1: ターゲットロールを指定するための Task 状態定義を更新する
- ステップ 2 : ターゲットロールの信頼ポリシーを更新する
- ステップ 3: 必要なアクセス許可をターゲットロールに追加する
- ステップ 4: 実行ロールに権限を追加してターゲットロールを引き受ける

前提条件

- このチュートリアルでは、クロスアカウントアクセスの設定方法を、Lambda 関数の例を使用して説明します。 AWS 他のリソースも使用できますが、必ず別のアカウントでリソースを設定してください。

  > **Important**
  >
  > IAM ロールとリソースベースのポリシーは、単一のパーティション内のアカウント間でのみアクセスを委任します。例えば、標準 aws パーティションの米国西部 (北カリフォルニア)と、aws-cn パーティションの中国 (北京) にアカウントがあるとします。中国 (北京) アカウントの Amazon S3 リソースベースのポリシーを使用して、標準 aws アカウントのユーザーにアクセスを許可することはできません。

- テキストファイルにクロスアカウントリソースの Amazon リソースネーム (ARN) をメモしておきます。このチュートリアルの後半で、この ARN をステートマシンの Task 状態定義に入力します。以下は、Lambda 関数の ARN の例です。

```
arn:aws:lambda:us-east-2:123456789012:function:functionName
```

- ステートマシンが引き受ける必要があるターゲット IAM ロールが作成できたことを確認してください。

ステップ 1: ターゲットロールを指定するための Task 状態定義を更新する

クロスアカウントの Lambda 関数を呼び出す前に、ワークフローの Task 状態で、ステートマシンが引き受ける必要がある `Credentials` フィールド (アイデンティティを含む) を追加します。

次の手順で、クロスアカウント Lambda 関数へのアクセス方法を説明します。Echo次の手順に従うと、 AWS どのリソースでも呼び出すことができます。

1. Step Functions コンソールを開き、[ステートマシンの作成] を選択します。
2. [オーサリング方法を選択] ページで [ワークフローを視覚的に設計] を選択し、すべてをデフォルトの選択のままにします。
3. Workflow Studio を開くには、[次へ] を選択します。
4. [アクション] タブで、Task 状態をキャンバスにドラッグアンドドロップします。これで、この Task 状態を使用しているクロスアカウントの Lambda 関数が呼び出されます。
5. [設定] タブで以下の操作を行います。

   a. 状態の名前を **`Cross-account call`** に変更します。

   b. [関数名] で [関数名を入力] を選択し、ボックスに Lambda 関数の ARN を入力します。例えば `arn:aws:lambda:us-east-2:111122223333:function:`*`Echo`* です。

   c. [IAM ロールの ARN を指定] でターゲットの IAM ロール ARN を指定します。例えば `arn:aws:iam::111122223333:role/LambdaRole` です。

   > **Tip**
   >
   > またはその状態の JSON 入力で、IAM ロールの ARN を含む既存の key-value ペアへの参照パスも指定できます。これを行うには、[状態入力からランタイムに IAM ロール ARN を取得する] を選択します。参照パスを使用して値を指定する例については、IAM ロール ARN として JSONPath を指定を参照してください。

6. [次へ] をクリックします。
7. [生成されたコードを確認] ページで [次へ] を選択します。
8. [ステートマシン設定を指定] ページで、名前、権限、ログレベルなど、新しいステートマシンの詳細を指定します。

9. [ステートマシンの作成] を選択します。
10. ステートマシンの IAM ロール ARN とステートマシン ARN をテキストファイルにメモしておきます。ターゲットアカウントの信頼ポリシーで、これらの ARN を指定する必要があります。

Task 状態の定義は、以下のようにします。

```
{
  "StartAt": "Cross-account call",
  "States": {
    "Cross-account call": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn": "arn:aws:iam::111122223333:role/LambdaRole"
      },
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:Echo",
      },
      "End": true
    }
  }
}
```

ステップ 2：ターゲットロールの信頼ポリシーを更新する

IAM ロールはターゲットアカウントに存在している必要があるため、ソースアカウントがこのロールを一時的に引き受けられるように、信頼ポリシーを変更する必要があります。さらに、ターゲットの IAM ロールを引き受けられるユーザーを制御できます。

信頼関係を作成すると、ソースアカウントのユーザーは AWS Security Token Service (AWS STS) AssumeRoleAPI オペレーションを使用できます。この操作は、 AWS ターゲットアカウントのリソースへのアクセスを可能にする一時的なセキュリティ認証情報を提供します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールのナビゲーションペインで、[ロール]を選択し、[検索] ボックスでターゲットの IAM ロールを検索します。例えば *LambdaRole* です。
3. [信頼関係] タブを選択します。
4. [信頼ポリシーを編集] を選択して、次のポリシーを貼り付けます。 AWS アカウント 番号と IAM ロール ARN を必ず置き換えてください。sts:ExternalId フィールドでは、ロールを引き

受けられるユーザーをさらに制御できます。ステートマシンの名前には AWS Security Token Service `AssumeRole` API がサポートする文字のみを含める必要があります。詳細については、AWS Security Token Service API AssumeRoleリファレンスのを参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/ExecutionRole"  // The source
 account's state machine execution role ARN
      },
      "Condition": {  // Control which account and state machine can assume the
 target IAM role

        "StringEquals": {
          "sts:ExternalId": "arn:aws:states:us-
east-1:123456789012:stateMachine:testCrossAccount"   //// ARN of the state machine
 that will assume the role.
        }
      }
    }
  ]
}
```

5. 以降のアクションに進むには、このウィンドウを開いたままで次のステップに進みます。

ステップ 3: 必要なアクセス許可をターゲットロールに追加する

IAM ポリシーでの許可により、特定のリクエストが許可されるか拒否されるかが決まります。ターゲット IAM ロールには、Lambda 関数を呼び出すための適切なアクセス許可が必要です。

1. [アクセス許可] タブを選択します。
2. [アクセス許可を追加]、[インラインポリシーを作成] の順に選択します。
3. [JSON] タブを選択し、既存の内容を次の許可に置き換えます。必ず Lambda 関数の ARN を置き換えてください。

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:Echo"  // The
 cross-account AWS resource being accessed
    }
  ]
}
```

4. [ポリシーを確認] を選択します。
5. [ポリシーを確認] ページで許可の名前を入力し、[ポリシーを作成] を選択します。

ステップ 4: 実行ロールに権限を追加してターゲットロールを引き受ける

Step Functions は、AssumeRoleすべてのクロスアカウントサービス統合のポリシーを自動的に生成するわけではありません。ステートマシンが 1 つ以上のターゲット IAM ロールを引き受けられるようにするには、必要な権限をステートマシンの実行ロールに追加する必要があります。 AWS アカウント

1. https://console.aws.amazon.com/iam/ の IAM コンソール内でステートマシンの実行ロールを開きます。これを実行するには:
   a. ステップ 1 でソースアカウント内に作成したステートマシンを開きます。
   b. [ステートマシンの詳細] ページで、[IAM ロールの ARN] を選択します。
2. [許可] タブで [アクセス許可を追加] を選択し、次に[インラインポリシーを作成] を選択します。
3. [JSON] タブを選択し、既存の内容を次の許可に置き換えます。必ず Lambda 関数の ARN を置き換えてください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/LambdaRole"  // The target role
 to be assumed
    }
  ]
}
```

4. [ポリシーを確認] を選択します。
5. [ポリシーを確認] ページで許可の名前を入力し、[ポリシーを作成] を選択します。

.sync 統合パターンへのクロスアカウントアクセス

ワークフローで `.sync` サービス統合パターンを使用すると、Step Functions は呼び出されたクロスアカウントリソースをポーリングして、タスクの完了を確認します。これにより、実際のタスク完了時間と Step Functions がタスク完了を認識する時間との間に若干の遅延が生じます。このポーリングループを完了させるには、ターゲットの IAM ロールが `.sync` の呼び出しに必要な許可を持つ必要があります。これを行うには、ターゲットの IAM ロールの信頼ポリシーで、ソースアカウントによる引き受けを許可する必要があります。さらに、ターゲットの IAM ロールにはポーリングループを完了させるための許可が必要です。

> **Note**
>
> 現在、`arn:aws:states:::states:startExecution.sync` はネストされた Express ワークフローをサポートしていません。代わりに `arn:aws:states:::aws-sdk:sfn:startSyncExecution` を使用します。

.sync 呼び出しのための信頼ポリシーの更新

次の例に示すように、ターゲットの IAM ロールの信頼ポリシーを更新します。`sts:ExternalId` フィールドでは、ロールを引き受けられるユーザーをさらに制御できます。ステートマシンの名前には AWS Security Token Service `AssumeRole` API がサポートする文字のみを含める必要があります。詳細については、AWS Security Token Service API AssumeRoleリファレンスのを参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::sourceAccountID:role/InvokeRole",
      },
      "Condition": {
        "StringEquals": {
```

```
          "sts:ExternalId": "arn:aws:states:us-
east-2:sourceAccountID:stateMachine:stateMachineName"
        }
      }
    }
  ]
}
```

.sync 呼び出しに必要な許可

ステートマシンに必要な許可を付与するには、ターゲット IAM ロールの必要な許可を更新します。詳細については、「the section called “統合サービスの IAM ポリシー”」を参照してください。サンプルポリシーの Amazon EventBridge 権限は必要ありません。例えばステートマシンを起動するには、以下の許可を追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:execution:stateMachineName:*"
      ]
    }
  ]
}
```

Step Functions の Amazon VPC エンドポイント

Amazon Virtual Private Cloud (Amazon VPC) AWS を使用してリソースをホストする場合、Amazon VPC AWS Step Functions とワークフロー間の接続を確立できます。公開インターネットと交差せずに、この Step Functions ワークフローとの接続を使用できます。Amazon VPC エンドポイントは、Standard ワークフロー、Express ワークフロー、同期 Express ワークフローでサポートされています。

Amazon VPC では、 AWS カスタム仮想ネットワークでリソースを起動できます。VPCを使用して、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

Amazon VPC を Step Functions に接続するには、まず VPC を他のサービスに接続できるインターフェイス VPC エンドポイントを定義する必要があります。 AWS このエンドポイントを使用すると、インターネットゲートウェイやネットワークアドレス変換 (NAT) インスタンス、または VPN 接続などを必要とせずに、信頼性の高いスケーラブルな方法で接続できるようになります。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

エンドポイントを作成する

、 AWS Command Line Interface (AWS CLI)、 AWS SDK、 AWS Step Functions API AWS Management Console、 AWS CloudFormationまたはを使用して VPC AWS Step Functions にエンドポイントを作成できます。

Amazon VPC コンソールまたは AWS CLIを使用して、エンドポイントを作成および設定する方法については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントの作成」を参照してください。

> **Note**
>
> エンドポイントを作成するとき、VPC の接続先のサービスとして Step Functions を指定します。Amazon VPC コンソールでは、 AWS サービス名はリージョンによって異なります。例えば、米国東部 (バージニア北部) を選択した場合、Standard ワークフロー と Express ワークフロー のサービス名は com.amazonaws.us-east-1.states であり、同期 Express ワークフロー のサービス名は com.amazonaws.us-east-1.sync-states です。

> **Note**
>
> プライベート DNS を通じて、SDK のエンドポイントをオーバーライドせずに VPC エンドポイントを使用できます。ただし、同期 Express ワークフロー用の SDK のエンドポイントをオーバーライドする場合は、`DisableHostPrefixInjection` 構成を `true` に設定する必要があります。例 (Java SDK V2):
>
> ```
> SfnClient.builder()
>   .endpointOverride(URI.create("https://vpce-{vpceId}.sync-states.us-
> east-1.vpce.amazonaws.com"))
>   .overrideConfiguration(ClientOverrideConfiguration.builder()
>
>  .advancedOptions(ImmutableMap.of(SdkAdvancedClientOption.DISABLE_HOST_PREFIX_INJECTION,
>  true))
>     .build())
>   .build();
> ```

を使用してエンドポイントを作成および設定する方法については AWS CloudFormation、ユーザーガイドの AWS:: EC2:: VpcEndpoint リソースを参照してください。AWS CloudFormation

Amazon VPC エンドポイントポリシー

Step Functions への接続アクセスを制御するには、Amazon VPC エンドポイントの作成時に AWS Identity and Access Management (IAM) エンドポイントポリシーをアタッチできます。複数のエンドポイントポリシーをアタッチすることで、複雑な IAM ルールを作成できます。詳細については、以下を参照してください。

- Step Functions 用 Amazon Virtual Private Cloud エンドポイントのポリシー
- 管理者以外のユーザー用の詳細な IAM 許可の作成
- VPC エンドポイントによるサービスのアクセス制御

Step Functions 用 Amazon Virtual Private Cloud エンドポイントのポリシー

Step Functions 用 Amazon VPC エンドポイントのポリシーを作成できます。このポリシーでは以下を指定します。

- アクションを実行できるプリンシパル。

- 実行可能なアクション。
- このアクションを実行できるリソース。

以下は、1 人の IAM ユーザーにステートマシンの作成を許可し、他のすべての IAM ユーザーからのステートマシン削除権限を拒否できる Amazon VPC エンドポイントのポリシーの例です。またこのサンプルポリシーでは、すべてのユーザーに対して実行許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "*Execution",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": "*"
        },
        {
            "Action": "states:CreateStateMachine",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": {
              "AWS": "arn:aws:iam::123456789012:user/MyUser"
            }
        },
        {
            "Action": "states:DeleteStateMachine",
            "Resource": "*",
            "Effect": "Deny",
            "Principal": "*"
        }
    ]
}
```

エンドポイントポリシーの作成の詳細については、以下を参照してください。

- 管理者以外のユーザー用の詳細な IAM 許可の作成
- VPC エンドポイントによるサービスのアクセス制御

統合サービスの IAM ポリシー

AWS Step Functions コンソールでステートマシンを作成すると、Step Functions はステートマシン定義で使用されるリソースに基づいて AWS Identity and Access Management (IAM) ポリシーを次のように作成します。

- ステートマシンが Optimized 統合の 1 つを使用している場合、ステートマシンに必要な許可とロールを持つポリシーが作成されます。
- ステートマシンが AWS SDK インテグレーションのいずれかを使用する場合、部分的な権限を持つ IAM ロールが作成されます。その後、IAM コンソールを使用して、欠落しているロールポリシーを追加できます。

以下は、ステートマシンの定義に基づき、Step Functions でポリシーが生成される様子を示す例です。*`[[resourceName]]`* などのサンプルコードの項目は、ステートマシンの定義に表示されている静的リソースに置き換えられています。複数の静的リソースがある場合は、IAM ロールの各エントリが存在します。

動的リソースと静的リソース

静的リソースは、ステートマシンのタスク状態で直接定義されます。タスク状態で直接呼び出すリソースに関する情報を含めると、Step Functions はそのリソースのみの IAM ロールを作成します。

動的リソースは状態入力で渡され、パスを使用してアクセスできるリソースです (パス を参照してください)。動的リソースをタスクに渡す場合、Step Functions は `"Resource": "*"` を指定する権限の高いポリシーを作成します。

[ジョブを実行] パターンを使用したタスクの追加許可

[ジョブを実行] パターン (`.sync` で終了するもの) を使用するタスクについては、接続されたサービスの API アクションからのレスポンスのモニタリングおよび受信に、追加許可が必要です。関連するポリシーには、[リクエストレスポンス] または [コールバックを待つ] パターンを使用するタスクよりも多くの許可が含まれます。同期タスクに関する情報は、サービス統合パターン を参照してください。

> **Note**
>
> Run a Job (.sync) パターンをサポートするサービス統合には、追加の権限を提供する必要があります。

Step Functions は、接続サービスでジョブが実行されたときのジョブのステータスをモニタリングするため、ポーリングとイベントという 2 つの方法を使用します。

ポーリングには、`Describe` または `Get` API アクション に対する許可 (`ecs:DescribeTasks` または `glue:GetJobRun` など) が必要です。これらの許可がロールに欠落している場合、Step Functions はジョブのステータスを判断できない場合があります。これは、Run a Job (.sync) EventBridge サービスの統合の中にはイベントをサポートしていないものや、ベストエフォート方式でのみイベントを送信するものがあるためです。

AWS サービスから Amazon EventBridge に送信されるイベントは、マネージドルールを使用して Step Functions に送られ`events:PutTargets`、、`events:PutRule`、の権限が必要です`events:DescribeRule`。これらの許可がロールに欠落している場合、Step Functions がジョブの完了を認識するまで遅延が発生する可能性があります。 EventBridge イベントについて詳しくは、「AWS サービスからのイベント」を参照してください。

> **Note**
>
> ポーリングとイベントの両方をサポートする[ジョブを実行] (.sync) タスクの場合、イベントを使用してタスクが正常に完了する場合があります。これは、ロールにポーリングに必要な許可がない場合でも発生します。この場合、ポーリング許可が正しくない、または欠落していることにすぐに気付かないことがあります。まれにイベントが Step Functions に配信されない場合や、Step Functions によって処理されない場合、実行がスタックする可能性があります。ポーリング権限が正しく設定されていることを確認するには、 EventBridge 以下の方法でイベントのない環境で実行を実行できます。
>
> - Step Functions にイベントを転送するマネージドルールをから削除します。 EventBridge このマネージドルールはアカウント内のすべてのステートマシンで共有されるため、他のステートマシンに思いがけない影響が出ないようにするため、このアクションを実行するのはテストアカウントまたは開発アカウントだけにしてください。削除する特定のマネージドルールは、ターゲットサービスのポリシーテンプレートで `events:PutRule` に使用する `Resource` フィールドを検査して特定できます。マネージドルールは、そのサービス統合を使用するステートマシンを次回作成または更新するときに再度作成されます。EventBridge ルールの削除について詳しくは、「ルールを無効化または削除する」を参照してください。
> - Step Functions Local でテストします。これは、[ジョブを実行] (.sync) タスクを完了するために行われるイベントの使用をサポートしていません。Step Functions Local を使用するには、ステートマシンで使用する IAM ロールを引き受けます。[信頼関係] を編集する必要がある場合もあります。`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`、およ

び AWS_SESSION_TOKEN 環境変数を、引き受けたロールの値に設定してから、java -jar StepFunctionsLocal.jar を使用する Step Functions Local を起動します。最後に、 AWS CLI --endpoint-urlとパラメーターを使用してステートマシンを作成し、実行を開始し、実行履歴を取得します。詳細については、「ステートマシンのローカルテスト」を参照してください。

[ジョブを実行] (.sync) パターンを使用するタスクが停止すると、Step Functions はベストエフォートでタスクをキャンセルしようとします。これには、Cancel、Stop、Terminate、または Delete API アクション (batch:TerminateJob または eks:DeleteCluster など) への許可が必要です。ロールにこれらの許可がない場合、Step Functions はタスクをキャンセルできず、タスクの実行中に追加料金が発生する可能性があります。タスク停止の詳細については、ジョブ実行を参照してください。

IAM ロールの作成に使用されるポリシーテンプレート

次のトピックでは、Step Functions で自分たち用に新しいロールの作成を選択する場合に使用するポリシーテンプレートなどについて説明します。

> **Note**
>
> これらのテンプレートを確認して、Step Functions が IAM ポリシーを作成する方法を理解してください。また、 AWS 他のサービスと連携する場合に Step Functions の IAM ポリシーを手動で作成する方法の例としても役立ちます。Step Functions サービス統合の詳細については、AWS Step Functions 他のサービスとの併用 を参照してください。

トピック

- Amazon API Gateway IAM ポリシー
- Amazon Athena の IAM ポリシー
- の IAM ポリシー AWS Batch
- IAM policies for Amazon Bedrock
- の IAM ポリシー AWS CodeBuild
- Amazon DynamoDB の IAM ポリシー
- Amazon ECS の IAM ポリシー/AWS Fargate
- Amazon EKS の IAM ポリシー

- Amazon EMR の IAM ポリシー
- EKS での Amazon EMR の IAM ポリシー
- Amazon EMR Serverless の IAM ポリシー
- Amazon の IAM ポリシー EventBridge
- の IAM ポリシー AWS Lambda
- の IAM ポリシー AWS Glue
- の IAM ポリシー AWS Glue DataBrew
- Amazon の IAM ポリシー SageMaker
- Amazon SNS の IAM ポリシー
- Amazon SQS の IAM ポリシー
- の IAM ポリシー AWS Step Functions
- の IAM ポリシー AWS X-Ray
- アクティビティまたはタスクなし

Amazon API Gateway IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

リソース:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:[[region]]:[[accountId]]:*"
            ]
        }
    ]
}
```

次のコード例では、API Gateway を呼び出すためのリソースポリシーを示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "states.amazonaws.com"
            },
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:<region>:<account-id>:<api-id>/<stage-name>/<HTTP-VERB>/<resource-path-specifier>",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": [
                        "<SourceStateMachineArn>"
                    ]
                }
            }
        }
    ]
}
```

Amazon Athena の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

`StartQueryExecution`

静的リソース

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
```

```
            "athena:stopQueryExecution",
            "athena:getQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/[[workGroup]]",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
```

```
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/[[workGroup]]",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
```

```
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
```

```
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

動的リソース

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:stopQueryExecution",
            "athena:getQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
```

```
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
```

```
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
```

```
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

StopQueryExecution

リソース

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

```
            "athena:stopQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    }
    ]
}
```

GetQueryExecution

リソース

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:getQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    }
    ]
}
```

GetQueryResults

リソース

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:getQueryResults"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
```

```
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    }
    ]
}
```

の IAM ポリシー AWS Batch

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

AWS Batch はリソースレベルのアクセス制御を部分的にサポートしているため、を使用する必要があります。`"Resource": "*"`

Run a Job (.sync)

```
{
    "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"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM policies for Amazon Bedrock

コンソールを使用してステートマシンを作成すると、必要な最小特権を持つステートマシンの実行ロールがStep Functions によって自動的に作成されます。IAMこれらの自動生成されたロールは、AWS リージョン ステートマシンを作成する場合に有効です。

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

IAM ポリシーを作成するときは、ポリシーにワイルドカードを含めないことをお勧めします。セキュリティのベストプラクティスとして、ポリシーの範囲をできるだけ絞り込む必要があります。動的ポリシーは、ランタイム中に特定の入力パラメータが不明な場合にのみ使用してください。

このトピックの内容

- Step Functions との Amazon Bedrock 統合の IAM ポリシー例

Step Functions との Amazon Bedrock 統合の IAM ポリシー例

次のセクションでは、特定の基盤モデルまたはプロビジョニングされたモデルに使用する Amazon Bedrock API に基づいて必要な IAM アクセス許可について説明します。このセクションには、フルアクセスを許可するポリシーの例も含まれています。

######のテキストを、リソース固有の情報に必ず置き換えてください。

- IAMを使用して特定の基盤モデルにアクセスするポリシーの例 InvokeModel
- IAMを使用して特定のプロビジョニング済みモデルにアクセスするポリシーの例 InvokeModel
- IAMフルアクセスポリシーの使用例 InvokeModel
- 特定の基盤モデルに基本モデルとしてアクセスする IAM ポリシーの例
- 特定のカスタムモデルに基本モデルとしてアクセスする IAM ポリシーの例
- CreateModelCustomizationJob.sync IAM を使用するフルアクセスポリシーの例
- IAM CreateModelCustomizationJob.sync を使用して特定の基盤モデルにアクセスするポリシー例
- IAM.sync CreateModelCustomizationJob を使用してカスタムモデルにアクセスするポリシーの例
- .sync IAM CreateModelCustomizationJob を使用するフルアクセスポリシーの例

IAMを使用して特定の基盤モデルにアクセスするポリシーの例 InvokeModel

以下は、InvokeModelAPI IAM `amazon.titan-text-express-v1` アクションを使用して指定された特定の基盤モデルにアクセスするステートマシンのポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "InvokeModel1",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-v1"
            ]
        }
    ]
}
```

IAMを使用して特定のプロビジョニング済みモデルにアクセスするポリシーの例 InvokeModel

以下は、API IAM c2oi931ulksx アクションを使用して指定された特定のプロビジョニング済みモデルにアクセスするステートマシンのポリシーの例です。InvokeModel

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Sid": "InvokeModel1",
      "Action": [
        "bedrock:InvokeModel"
      ],
      "Resource": [
        "arn:aws:bedrock:us-east-2:123456789012:provisioned-model/c2oi931ulksx"
      ]
    }
  ]
}
```

IAMフルアクセスポリシーの使用例 InvokeModel

以下は、InvokeModelAPI IAM アクションを使用するときにフルアクセスを提供するステートマシンのポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "InvokeModel1",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:provisioned-model/*"
            ]
        }
    ]
}
```

特定の基盤モデルに基本モデルとしてアクセスする IAM ポリシーの例

以下は、ステートマシンが CreateModelCustomizationJobAPI IAM `amazon.titan-text-express-v1` アクションを使用してベースモデルとして指定された特定の基盤モデルにアクセスする場合のポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-v1",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

特定のカスタムモデルに基本モデルとしてアクセスする IAM ポリシーの例

以下は、ステートマシンが CreateModelCustomizationJobAPI IAM アクションを使用してベースモデルとして特定のカスタムモデルにアクセスする場合のポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/[[roleName]]"
            ]
        }
    ]
}
```

CreateModelCustomizationJob.sync IAM を使用するフルアクセスポリシーの例

以下は、CreateModelCustomizationJobAPI IAM アクションを使用するときにフルアクセスを提供するステートマシンのポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

IAM CreateModelCustomizationJob.sync を使用して特定の基盤モデルにアクセスするポリシー例

以下は、CreateModelCustomizationJobステートマシンが .sync API IAM amazon.titan-text-express-v1 アクションを使用して指定された特定の基盤モデルにアクセスする場合のポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-
v1",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
            ],
```

```
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

IAM.sync CreateModelCustomizationJob を使用してカスタムモデルにアクセスするポリシーの例

以下は、CreateModelCustomizationJobステートマシンが .sync API IAM アクションを使用してカスタムモデルにアクセスする場合のポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
            ],
```

```
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

.sync IAM CreateModelCustomizationJob を使用するフルアクセスポリシーの例

以下は、CreateModelCustomizationJob.sync API IAM アクションを使用するときにフルアクセスを提供するステートマシンのポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
```

```
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

の IAM ポリシー AWS CodeBuild

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

リソース:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution1111-2222-3333-wJalrXUtnFEMI-SNSTopic-bPxRfiCYEXAMPLEKEY"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "codebuild:StartBuild",
                "codebuild:StopBuild",
```

```
                "codebuild:BatchGetBuilds",
                "codebuild:BatchGetReports"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

StartBuild

静的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:BatchGetBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

動的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild",
```

```
        "codebuild:StopBuild",
        "codebuild:BatchGetBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

StopBuild

静的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

動的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

BatchDeleteBuilds

静的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

動的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

BatchGetReports

静的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:report-group/[[reportName]]"
```

```
      ]
    }
  ]
}
```

動的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:report-group/*"
      ]
    }
  ]
}
```

StartBuildBatch

静的リソース

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildBatchRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

動的リソース

Run a Job (.sync)

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

```
            "Action": [
                "codebuild:StartBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildBatchRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

StopBuildBatch

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

RetryBuildBatch

静的リソース

Run a Job (.sync)

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

動的リソース

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

DeleteBuildBatch

静的リソース

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

```
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

Amazon DynamoDB の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
```

```
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:[[region]]:[[accountId]]:table/[[tableName]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

すべての DynamoDB API アクションの IAM ポリシーの詳細については、「Amazon DynamoDB デベロッパーガイド」の「DynamoDB での IAM ポリシー」を参照してください。さらに、DynamoDB 用 PartiQL の IAM ポリシーについては、「Amazon DynamoDB DynamoDB デベロッパーガイド」の「DynamoDB 用 PartiQL の IAM ポリシー」を参照してください。

Amazon ECS の IAM ポリシー/AWS Fargate

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

`TaskId` の値は、タスクが送信されるまで不明なため、Step Functions は、権限の高い `"Resource": "*"` ポリシーを作成します。

> **Note**
>
> "*" IAM ポリシーにもかかわらず、Step Functions によってスタートした Amazon Elastic Container Service (Amazon ECS) タスクを停止できるだけです。

Run a Job (.sync)

静的リソース

```
{
    "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"
            ]
        }
```

```
    ]
}
```

動的リソース

```
{
    "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"
            ]
        }
    ]
}
```

Request Response and Callback (.waitForTaskToken)

静的リソース

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

```
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:[[region]]:
[[accountId]]:task-definition/[[taskDefinition]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": "*"
        }
    ]
}
```

スケジュールされた Amazon ECS タスクでタスク実行ロール、タスクロール、またはタスクロールオーバーライドの使用が必要な場合は、各タスク実行ロール、タスクロール、`iam:PassRole`またはタスクロールオーバーライドのアクセス権限を、呼び出し元エンティティ (この場合は Step Functions) の CloudWatch Events IAM ロールに追加する必要があります。

Amazon EKS の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

`CreateCluster`

リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:CreateCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:DeleteCluster"
            ],
            "Resource": "arn:aws:eks:sa-east-1:444455556666:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::444455556666:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

```
```

CreateNodeGroup

リソース

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeSubnets",
                "eks:CreateNodegroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeNodegroup",
                "eks:DeleteNodegroup"
            ],
            "Resource": "arn:aws:eks:sa-east-1:444455556666:nodegroup/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "arn:aws:iam::444455556666:role/*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::444455556666:role/StepFunctionsSample-EKSClusterMan-
NodeInstanceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

DeleteCluster

リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DeleteCluster",
                "eks:DescribeCluster"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:cluster/ExampleCluster"
            ]
        }
    ]
}
```

DeleteNodegroup

リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DeleteNodegroup",
                "eks:DescribeNodegroup"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:nodegroup/ExampleCluster/
ExampleNodegroup/*"
            ]
        }
    ]
}
```

Step Functions での Amazon EKS の使用の詳細については、Step Functions で Amazon EKS を呼び出す を参照してください。

Amazon EMR の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

addStep

静的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

動的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
```

```
}
```

cancelStep

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

createCluster

静的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:RunJobFlow",
        "elasticmapreduce:DescribeCluster",
        "elasticmapreduce:TerminateJobFlows"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": [
        "arn:aws:iam::{{account}}:role/[[roleName]]"
      ]
    }
  ]
}
```

setClusterTerminationProtection

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

動的リソース

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

```
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

modifyInstanceFleetByName

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

modifyInstanceGroupByName

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

terminateCluster

静的リソース

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

動的リソース

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
}
```

EKS での Amazon EMR の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

`CreateVirtualCluster`

リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:CreateVirtualCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/emr-
containers.amazonaws.com/AnAWSServiceRoleForAmazonEMRContainers",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "emr-containers.amazonaws.com"
                }
            }
        }
    ]
}
```

DeleteVirtualCluster

静的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster",
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ]
    }
  ]
}
```

動的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster",
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

Request Response

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

StartJobRun

静的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
      ],
```

```
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]/jobruns/*"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    }
  ]
}
```

動的リソース

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
```

```
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    }
  ]
```

```
}
```

Amazon EMR Serverless の IAM ポリシー

コンソールを使用してステートマシンを作成すると、必要な最小特権を持つステートマシンの実行ロールがStep Functions によって自動的に作成されます。IAMこれらの自動生成されたロールは、AWS リージョン ステートマシンを作成する際に有効です。

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

IAM ポリシーを作成するときは、ポリシーにワイルドカードを含めないことをお勧めします。セキュリティのベストプラクティスとして、ポリシーの範囲をできるだけ絞り込む必要があります。動的ポリシーは、ランタイム中に特定の入力パラメータが不明な場合にのみ使用してください。

さらに、管理者ユーザーが非管理者ユーザーに、ステートマシンを実行するための実行ロールを付与する場合には注意が必要です。自分でポリシーを作成する場合は、実行ロールに PassRole ポリシーを含めることをお勧めします。また、実行ロールには `aws:SourceARN` および `aws:SourceAccount` のコンテキストキーを追加することをお勧めします。

このトピックの内容

- EMR Serverless と Step Functions の統合の場合の IAM ポリシーの例

EMR Serverless と Step Functions の統合の場合の IAM ポリシーの例

- の IAM ポリシーの例 CreateApplication
- の IAM ポリシーの例 StartApplication
- の IAM ポリシーの例 StopApplication
- の IAM ポリシーの例 DeleteApplication
- の IAM ポリシーの例 StartJobRun
- の IAM ポリシーの例 CancelJobRun

の IAM ポリシーの例 CreateApplication

ステートのあるステートマシンの IAM ポリシーの例を以下に示します。 CreateApplication タスク

> **Note**
>
> アカウントで初めてアプリケーションを作成するときに、IAM CreateServiceLinkedRole ポリシーで権限を指定する必要があります。それ以降、この許可を追加する必要はありません。の詳細については CreateServiceLinkedRole、https://docs.aws.amazon.com/IAM/latest/APIReference/ CreateServiceLinkedRoleのを参照してください。

以下のポリシーは、静的リソースと動的リソースで同じです。

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CreateApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:GetApplication",
                "emr-serverless:DeleteApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
```

```
                "arn:aws:events:{{region}}:{{accountId}}:rule/
StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWS ServiceRoleForAmazonEMRServerless*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

Request Response

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

   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CreateApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWS ServiceRoleForAmazonEMRServerless*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"
                }
            }
```

```
        }
    ]
}
```

の IAM ポリシーの例 StartApplication

静的リソース

ステートマシンをステートとともに使用する場合の静的リソースの IAM ポリシーの例を以下に示します。 StartApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication",
               "emr-serverless:GetApplication",
               "emr-serverless:StopApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        }
    ]
}
```

動的リソース

ステートマシンとステートマシンを使用する場合の動的リソースの IAM ポリシーの例を以下に示します。 StartApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication",
               "emr-serverless:GetApplication",
               "emr-serverless:StopApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

の IAM ポリシーの例 StopApplication

静的リソース

ステートマシンをステートとともに使用する場合の静的リソースの IAM ポリシーの例を以下に示します。 StopApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "emr-serverless:StopApplication",
                "emr-serverless:GetApplication"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
             ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
             ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:StopApplication"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
             ]
        }
    ]
}
```

動的リソース

ステートマシンとステートマシンを使用する場合の動的リソースの IAM ポリシーの例を以下に示します。 StopApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StopApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

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

```
                "emr-serverless:StopApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

の IAM ポリシーの例 DeleteApplication

静的リソース

ステートマシンをステートとともに使用する場合の静的リソースの IAM ポリシーの例を以下に示します。 DeleteApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
```

```
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        }
    ]
}
```

動的リソース

ステートマシンとステートマシンを使用する場合の動的リソースの IAM ポリシーの例を以下に示します。 DeleteApplication タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

の IAM ポリシーの例 StartJobRun

静的リソース

ステートマシンをステートとともに使用する場合の静的リソースの IAM ポリシーの例を以下に示します。 StartJobRun タスク

Run a Job (.sync)

```
{
```

```
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
       },
       {
           "Effect": "Allow",
           "Action": "iam:PassRole",
           "Resource": [
               "[[jobExecutionRoleArn]]"
           ],
           "Condition": {
               "StringEquals": {
                   "iam:PassedToService": "emr-serverless.amazonaws.com"
               }
           }
       },
       {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:GetJobRun",
               "emr-serverless:CancelJobRun"
           ],
           "Resource": [
               "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": [
               "events:PutTargets",
               "events:PutRule",
               "events:DescribeRule"
           ],
           "Resource": [
```

```
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

動的リソース

ステートマシンとステートマシンを使用する場合の動的リソースの IAM ポリシーの例を以下に示します。 StartJobRun タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun",
               "emr-serverless:GetJobRun",
               "emr-serverless:CancelJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

の IAM ポリシーの例 CancelJobRun

静的リソース

ステートマシンをステートとともに使用する場合の静的リソースの IAM ポリシーの例を以下に示します。 CancelJobRun タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CancelJobRun",
                "emr-serverless:GetJobRun"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/[[jobRunId]]"
             ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
             ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CancelJobRun"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/[[jobRunId]]"
             ]
        }
    ]
}
```

動的リソース

ステートマシンとステートマシンを使用する場合の動的リソースの IAM ポリシーの例を以下に示します。 CancelJobRun タスク

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:CancelJobRun",
               "emr-serverless:GetJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

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

```
                "emr-serverless:CancelJobRun"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

Amazon の IAM ポリシー EventBridge

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

`PutEvents`

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:123456789012:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "events:PutEvents"
            ],
            "Resource": "arn:aws:events:*:*:event-bus/*"
        }
    ]
}
```

Step EventBridge Functions での使用の詳細については、を参照してくださいStep EventBridge Functions による呼び出し。

の IAM ポリシー AWS Lambda

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

AWS Step Functions ステートマシン定義に基づいて IAM ポリシーを生成します。`function1`とを呼び出す 2 AWS Lambda `function2` つのタスクステートを持つステートマシンでは、2 `lambda:Invoke` つの関数に対する権限を持つポリシーを使用する必要があります。

以下の例ではこれを示しています。

```
{
    "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]]"
            ]
        }
    ]
}
```

の IAM ポリシー AWS Glue

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

AWS Glue にはリソースベースの制御はありません。

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartJobRun",
                "glue:GetJobRun",
                "glue:GetJobRuns",
                "glue:BatchStopJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

の IAM ポリシー AWS Glue DataBrew

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:startJobRun",
                "databrew:listJobRuns",
                "databrew:stopJobRun"
            ],
            "Resource": [
                "arn:aws:databrew:{{region}}:{{accountId}}:job/*"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:startJobRun"
            ],
            "Resource": [
                "arn:aws:databrew:{{region}}:{{accountId}}:job/*"
            ]
        }
    ]
}
```

Amazon の IAM ポリシー SageMaker

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

> **Note**
>
> これらの例では、ML コンピュートインスタンスへのデプロイやバッチ変換ジョブのために、モデルアーティファクトや docker SageMaker イメージにアクセスするために使用する IAM ロールの Amazon リソースネーム (ARN) *[[roleArn]]* を指します。詳細については、「Amazon SageMaker ロール」を参照してください。

`CreateTrainingJob`

静的リソース

Run a Job (.sync)

```
{
  "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"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "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"
        }
      }
    }
  ]
}
```

動的リソース

.sync or .waitForTaskToken

```
{
  "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"
      ]
    }
  ]
```

```
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "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 CreateTransformJobと統合するステートマシンを作成したときのポリシーは、自動的に作成されません。 SageMaker次のいずれかの IAM の例に基づいて、作成されたロールにインラインポリシーをアタッチする必要があります。

静的リソース

Run a Job (.sync)

```
{
  "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"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "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"
        }
      }
    }
  ]
}
```

動的リソース

Run a Job (.sync)

```
{
  "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"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "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": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

Amazon SNS の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

静的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:[[region]]:[[accountId]]:[[topicName]]"
            ]
        }
    ]
}
```

パスに基づくリソース、または *TargetArn* か *PhoneNumber* に発行する

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon SQS の IAM ポリシー

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

静的リソース

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": [
                "arn:aws:sqs:[[region]]:[[accountId]]:[[queueName]]"
            ]
        }
    ]
}
```

動的リソース

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
    ]
}
```

の IAM ポリシー AWS Step Functions

1 つのネストされたワークフロー実行のために StartExecution を呼び出すステートマシンの場合は、そのステートマシンへの許可を制限する IIAM ポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
```

```
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:stateMachine:[[stateMachineName]]"
            ]
        }
    ]
}
```

詳細については、次を参照してください。

- AWS Step Functions 他のサービスとの併用
- サービス API にパラメータを渡す
- AWS Step Functions 実行を統合サービスとして管理

Synchronous

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
            "Resource": [
                "arn:aws:states:[[region]]:[[accountId]]:stateMachine:
[[stateMachineName]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "states:DescribeExecution",
                "states:StopExecution"
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:execution:[[stateMachineName]]:*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
            ]
        }
    ]
}
```

Asynchronous

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:stateMachine:[[stateMachineName]]"
            ]
        }
    ]
}
```

ネストされたワークフロー実行の詳細については、タスク状態からワークフロー実行を開始する を参照してください。

の IAM ポリシー AWS X-Ray

以下のサンプルテンプレートは、ステートマシン定義内のリソースに基づいて IAM AWS Step Functions ポリシーを生成する方法を示しています。詳細については、統合サービスの IAM ポリシーおよびサービス統合パターンを参照してください。

X-Ray のトレースを有効にするには、トレースを許可する適切な許可を含んだ IAM ポリシーが必要です。ステートマシンが他の統合サービスを使用している場合は、追加の IAM ポリシーが必要になる場合があります。特定のサービス統合の IAM ポリシーを参照してください。

X-Ray のトレースを有効にしてステートマシンを作成すると、IAM ポリシーが自動的に作成されます。

> **Note**
>
> 既存のステートマシンのために X-Ray のトレースを有効にする場合は、X-Ray のトレースを有効にできる許可を含んだポリシーを追加する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Step Functions での X-Ray の使用の詳細については、AWS X-Ray および Step Functions を参照してください。

アクティビティまたはタスクなし

`Activity` タスクのみを持つステートマシン、またはタスクをまったく持たないステートマシンでは、すべてのアクションとリソースへのアクセスを拒否する IAM ポリシーを使用します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

`Activity` タスクの使用の詳細については、アクティビティ を参照してください。

分散マップ状態を使用するための IAM ポリシー

Step Functions コンソールでワークフローを作成すると、Step Functions はワークフロー定義内のリソースに基づいて IAM ポリシーを自動的に生成できます。これらのポリシーには、ステートマシンロールが分散マップ状態の `StartExecution` API アクションを呼び出すために必要な最小特権が含まれています。これらのポリシーには、Amazon S3 バケットやオブジェクト、Lambda AWS 関数などのリソースへのアクセスに必要な最低限の権限を持つステップ関数も含まれています。IAM ポリシーには必要なアクセス許可のみを含めることを強くお勧めします。例えばワークフローに分散モードの `Map` 状態が含まれている場合は、ポリシーの範囲をデータセットを含む特定の Amazon S3 バケットとフォルダに限定します。

> **Important**
>
> 分散マップ状態の入力で、既存のキーと値のペアへの参照パスとともに、Amazon S3 バケットやオブジェクト、またはプレフィックスを指定する場合は、ワークフローの IAM ポリシーを必ず更新してください。ポリシーの範囲は、ランタイムでパスから解釈されるバケット名とオブジェクト名に限定します。

このトピックの内容

- 分散マップ状態を実行するための IAM ポリシーの例
- 分散マップの IAM redriving ポリシーの例
- Amazon S3 データセットからデータを読み取る IAM ポリシーの例
- Amazon S3 バケットにデータを書き込むための IAM ポリシーの例

分散マップ状態を実行するための IAM ポリシーの例

ワークフローに分散マップ状態を含める場合、Step Functions には、ステートマシンのロールで分散マップ状態の StartExecution API アクションを呼び出すための適切な許可が必要です。

次の IAM ポリシー例では、ステートマシンのロールで分散マップ状態を実行するために必要な最小特権を付与しています。

> **Note**
>
> 必ず *stateMachineName* を、分散マップ状態を使用しているステートマシン名に置き換えてください。例えば arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine* です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": "arn:aws:states:region:accountID:execution:stateMachineName:*"
    }
  ]
}
```

分散マップの IAM redriving ポリシーの例

失敗した子ワークフローは、親ワークフローのマップ実行で redriving できます。redriven された親ワークフローは、分散マップを含むすべての失敗状態を redrives します。実行ロールには、親ワークフローで RedriveExecution API アクションを呼び出すために必要な最小特権があることを確認してください。

次の IAM ポリシー例では、ステートマシンのロールで分散マップ状態を redriving (再処理) するために必要な最小特権を付与しています。

> **Note**
>
> 必ず *stateMachineName* を、分散マップ状態を使用しているステートマシン名に置き換えてください。例えば arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine* です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
}
```

Amazon S3 データセットからデータを読み取る IAM ポリシーの例

次の IAM ポリシーの例では、ListObjectsV2 と GetObjectAPI アクションを使用して Amazon S3 データセットにアクセスするのに必要な最小限の権限を付与しています。

Example データセットとしての Amazon S3 オブジェクトの IAM ポリシー

以下は、Amazon S3 バケットの *processImages* に配置した *myBucket* という名前のオブジェクトにアクセスするための最小特権を付与する IAM ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example データセットとしての CSV ファイルの IAM ポリシー

以下は、*ratings.csv* という名前の CSV ファイルにアクセスするための最小特権を付与する IAM ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example データセットとしての Amazon S3 インベントリの IAM ポリシー

以下は、Amazon S3 インベントリレポートにアクセスするための最小特権を付与する IAM ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.json",
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/data/*"
            ]
        }
    ]
}
```

Amazon S3 バケットにデータを書き込むための IAM ポリシーの例

次の IAM ポリシーの例では PutObject API アクションを使用して、Amazon S3 バケット内にある *csvJobs* という名前のフォルダに、子ワークフローの実行結果を書き込むために必要な最小特権を付与しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::resultBucket/csvJobs/*"
            ]
```

```
        }
    ]
}
```

AWS KMS key 暗号化された Amazon S3 バケットの IAM アクセス許可

分散マップ状態では、マルチパートアップロードを使用して Amazon S3 バケットに子ワークフローの実行結果を書き込みます。バケットが AWS Key Management Service (AWS KMS) キーを使用して暗号化されている場合は、キーに対して `kms:Decrypt`、`kms:Encrypt`、`kms:GenerateDataKey` アクションを実行する許可も、IAM ポリシーに含める必要があります。マルチパートアップロードを完了する前に、暗号化されたファイル部分からデータを復号して読み取る必要があるため、Amazon S3 にはこれらの許可が必要です。

次の IAM ポリシーの例では、Amazon S3 バケットの暗号化に使用されるキーに対する `kms:Decrypt`、`kms:Encrypt`、および `kms:GenerateDataKey` アクションに許可を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:Encrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": [
      "arn:aws:kms:us-east-1:123456789012:key/111aa2bb-333c-4d44-5555-a111bb2c33dd"
    ]
  }
}
```

詳細については、AWS ナレッジセンターの「AWS KMS keyで暗号化した大容量ファイルを Amazon S3 にアップロードする」を参照してください。

IAM AWS アカウント ユーザーまたはロールがと同じ場合はKMS key、キーポリシーにこれらのアクセス権限が必要です。IAM ユーザーまたはロールが KMS key とは異なるアカウントに属している場合、キーポリシーへのアクセス許可と、 IAM ユーザーまたはロールへのアクセス許可が両方とも必要です。

タグベースのポリシー

Step Functions は、タグベースのポリシーをサポートしています。例えば、キー `environment` および値 `production` のタグを含むすべての Step Functions リソースへのアクセスを制限できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "states:TagResource",
                "states:UntagResource",
                "states:DeleteActivity",
                "states:DeleteStateMachine",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

このポリシーでは、ステートマシンまたはアクティビティを削除する機能、実行を停止する機能、および `environment/production` としてタグ付けされているすべてのリソースに対してタグを追加または削除する機能を Deny (拒否) します。

次の例のように、タグベースの認証の場合、ステートマシンの実行リソースはステートマシンに関連付けられたタグを継承します。

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

実行リソース ARN DescribeExecutionを指定するまたはその他の API を呼び出すと、Step Functions はステートマシンに関連付けられたタグを使用して、タグベースの認証を実行しながらリクエストを承認または拒否します。これにより、ステートマシンレベルの実行へのアクセスを許可または拒否できます。

タグ付けの詳細については、以下を参照してください。

- Step Functions でのタグ付け
- IAM タグを使用したアクセスの制御

ID とアクセスのトラブルシューティング AWS Step Functions

次の情報は、Step Functions と IAM の使用時に発生する可能性のある、一般的な問題の診断や修復に役立ちます。

トピック

- Step Functions でアクションを実行する権限がない
- 私にはiam を実行する権限がありません:PassRole
- 自分以外の人が自分の Step Functions AWS アカウント リソースにアクセスできるようにしたい

Step Functions でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

以下のエラー例は、`mateojackson` ユーザーがコンソールを使用して架空の *`my-example-widget`* リソースに関する詳細情報を表示しようとしているが、架空の `states:`*`GetWidget`* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 states:GetWidget on resource: my-example-widget
```

この場合、Mateo のポリシーでは、*`my-example-widget`* アクションを使用して `states:`*`GetWidget`* リソースへのアクセスを許可するように更新する必要があります。

サポートが必要な場合は、 AWS 管理者に問い合わせてください。サインイン資格情報を提供した担当者が管理者です。

私にはiam を実行する権限がありません:PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Step Functions にロールを渡せるようにする必要があります。

新しいサービスロールやサービスにリンクされたロールを作成する代わりに、 AWS のサービス 既存のロールをそのサービスに渡すことができるものもあります。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下のエラー例は、marymajor という名前の IAM ユーザーがコンソールを使用して、Step Functions でアクションを実行した場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された許可が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、メアリーのポリシーを更新してメアリーに iam:PassRoleアクションの実行を許可する必要があります。

サポートが必要な場合は、管理者に問い合わせてください。 AWS サインイン資格情報を提供した担当者が管理者です。

自分以外の人が自分の Step Functions AWS アカウント リソースにアクセスできるようにしたい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Step Functions がこれらの機能をサポートしているかどうかを確認するには、「IAM AWS Step Functions との連携方法」を参照してください。
- AWS アカウント 所有しているリソース全体のリソースへのアクセスを提供する方法については、『IAM ユーザーガイド』の「AWS アカウント 所有する別の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスを第三者に提供する方法については AWS アカウント、IAM ユーザーガイドの「AWS アカウント 第三者が所有するリソースへのアクセスの提供」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。

- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

ログ記録とモニタリング

ログインとモニタリングについては AWS Step Functions、記録とモニタリングセクションを参照してください。

のコンプライアンス検証 AWS Step Functions

第三者監査人は、 AWS Step Functions AWS 複数のコンプライアンスプログラムの一環としてセキュリティとコンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

AWS 特定のコンプライアンスプログラムの対象となるサービスの一覧については、「AWS コンプライアンスプログラムの対象サービス」、「」を参照してください。一般的な情報については、「AWS」を参照してください。

サードパーティの監査レポートはを使用してダウンロードできます AWS Artifact。詳細については、の「レポートのダウンロード」の「 AWS Artifact」を参照してください AWS Artifact。

Step Functions を使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や貴社のコンプライアンス目的、適用される法律および規制によって決まります。 AWS では、コンプライアンスに役立つ以下のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイドセキュリティとコンプライアンスのクイックスタートガイド 、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境をに展開する手順について説明します。 AWS
- Amazon Web Services での HIPAA セキュリティとコンプライアンスのためのアーキテクチャ — このホワイトペーパーでは、企業が HIPAA 準拠のアプリケーションを作成する方法について説明しています。 AWS
- AWS — この一連のワークブックとガイドは、お客様の業界や地域に当てはまる場合があります。
- AWS Config 開発者ガイドのルールによるリソースの評価 — AWS Config このサービスでは、リソース構成が社内慣行、業界のガイドライン、規制にどの程度準拠しているかを評価します。

- AWS Security Hub— AWS このサービスでは、内部のセキュリティ状態を包括的に把握できるため、 AWS セキュリティ業界の標準やベストプラクティスに準拠しているかどうかを確認できます。

レジリエンスとは AWS Step Functions

AWS AWS グローバルインフラストラクチャはリージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンには、物理的に分離され隔離された複数のアベイラビリティーゾーンがあり、低レイテンシー、高スループット、冗長性の高いネットワークで接続されています。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、「グローバルインフラストラクチャ」を参照してください。AWS

AWS グローバルなインフラストラクチャに加えて、Step Functions はデータの回復力とバックアップのニーズをサポートするのに役立ついくつかの機能を提供します。

のインフラストラクチャーセキュリティ AWS Step Functions

マネージドサービスとして、 AWS Step Functions AWS グローバルなネットワークセキュリティによって保護されています。 AWS AWS セキュリティサービスとインフラストラクチャの保護方法については、「AWS クラウドセキュリティ」を参照してください。 AWS インフラストラクチャセキュリティのベストプラクティスを使用して環境を設計するには、「Security Pillar AWS Well-Architected Framework におけるインフラストラクチャ保護」を参照してください。

AWS 公開されている API 呼び出しを使用して、ネットワーク経由で Step Functions にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2、できれば TLS 1.3 が必要です。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWS API オペレーションはネットワークのどの場所からでも呼び出すことができますが、ソース IP Step Functions アドレスに基づく制限を含むリソースベースのアクセスポリシーはサポートされていません。さらに、Step Functions ポリシーを使用して、特定の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントや特定の VPC からアクセスを制御することもできます。これにより、Step Functions特定のリソースへのネットワークアクセスが、ネットワーク内の特定の VPC からのみ効果的に分離されます。 AWS

の設定と脆弱性の分析 AWS Step Functions

構成と IT 管理は、 AWS お客様とお客様との間で共有される責任です。詳細については、「 AWS 責任分担モデル」を参照してください。

から Step Functions AWS Data Pipeline へのワークロードの移行

AWS は 2012 年に AWS Data Pipeline サービスを開始しました。当時、顧客は、さまざまなコンピューティングオプションを使用してさまざまなデータソース間でデータを移動できるサービスを求めていました。データ転送のニーズは時間とともに変化するため、そのニーズに対応するソリューションも変化します。現在は、ビジネス要件に最も近いソリューションを選択できるようになりました。例えば、次のいずれかを実行できます。

- Step Functions を使用して、複数の AWS のサービス 間でワークフローをオーケストレーションします。
- Amazon Managed Workflows for Apache Airflow (Amazon MWAA) を使用して、Apache Airflow のワークフローオーケストレーションを管理します。
- AWS Glue を使用して、Apache Spark アプリケーションを実行およびオーケストレーションします。

の一般的なユースケース AWS Data Pipeline は AWS Glue、、Step Functions、または Amazon MWAA のいずれかに移行できます。選択するオプションは、 AWS Data Pipeline の現在のワークロードによって異なります。このトピックでは、 から Step Functions AWS Data Pipeline に移行する方法について説明します。

トピック

- AWS Data Pipeline からのワークロードの移行
- Step Functions と AWS Data Pipeline の間のコンセプトマッピング
- Step Functions サンプルプロジェクト
- 料金比較

AWS Data Pipeline からのワークロードの移行

Step Functions は、ビジネスクリティカルなアプリケーションのワークフローを構築できる、サーバーレスのオーケストレーションサービスです。Step Functions のワークフロースタジオを使用すると、ワークフローを構築して、250 以上の AWS のサービス からの、11,000 を超える API アクションと統合できます。これには、 AWS Lambda、Amazon EMR、Amazon DynamoDB AWS のサービス などが含まれます。 DynamoDB Step Functions を使用すると、データ処理パイプラインをオーケ

ストレーションしたり、エラーを処理したり、基盤となるパイプラインのスロットリング制限を処理したりすることもできます。 AWS のサービスワークフローを使用して、機械学習モデルの処理と公開、マイクロサービスのオーケストレーション、抽出、変換、ロード (ETL) のワークフローを AWS Glue で処理するワークフローを作成できます。手動による介入が必要なアプリケーション用に、実行時間が長い自動化されたワークフローを作成することもできます。

Step Functions は、 AWS が提供するフルマネージドサービスです。つまり、ユーザーに代わって、インフラストラクチャのメンテナンス、ワーカーへのパッチ適用、OS バージョン更新の管理などのAWS タスクを管理します。

ユースケースが次の条件に一致する場合は、 から AWS Data Pipeline Step Functions に移行することをお勧めします。

- サーバーレスで可用性の高いワークフローオーケストレーションサービスを希望する。
- 1 つのタスク実行の詳細度に応じて課金するソリューションが必要である。
- ワークロードには、Amazon EMR AWS のサービス、Lambda、DynamoDB など AWS Glue、他の複数の のタスクのオーケストレーションが含まれます。
- ワークフロー作成には、 drag-and-drop ビジュアルデザイナーを備えたローコードソリューションが必要です。このソリューションでは、なじみのない複雑なプログラミング概念を学ぶ必要はありません。
- 11,000 を超える API アクション AWS のサービス をカバーする 250 を超える と統合するサービスが必要です。このサービスは、 以外のカスタムサービスやアクティビティとも統合する必要がありますAWS。

Step Functions と AWS Data Pipeline の間のコンセプトマッピング

AWS Data Pipeline と Step Functions は、いくつかの一般的な概念を共有します。例えば、ワークフローを定義するには、 AWS Data Pipeline と Step Functions の両方で JSON 形式を使用します。Step Functions では Amazon ステートメント言語 を使用します。これは JSON ベースの構造化言語です。Amazon States Language (ASL) を使用してワークフローを定義し、ワークフローのテキスト表現と視覚表現を切り替えます。この JSON ベースの形式は、ワークフローをソースコントロールツールに簡単に保存するのに役立ちます。また、ワークフローの複数のバージョンを管理したり、アクセスを制御したり、CI/CD メソッドによるオーケストレーションを自動化したりするのに役立ちます。

次の表は、両方のサービスで使用されている主要な概念のマッピングを示しています。左側のデータパイプラインの概念列には の概念がリストされ AWS Data Pipeline、右側の Step Functions の概念列には Step Functions の同等の概念がリストされます。

データパイプラインの概念	Step Functions の概要
パイプライン	ワークフロー
パイプライン定義	Amazon ステートメント言語 (ASL)
アクティビティ	状態 および タスク
インスタンス	実行
Attempts	Catchers と retriers
パイプラインスケジュール	• Amazon EventBridge Scheduler での実行 • EventBridge Pipes を介してトリガーされるイベント
パイプラインの式と関数	• 組み込み関数 • サービス統合を使用する Lambda 関数

Step Functions サンプルプロジェクト

Step Functions の概要については、次のビデオを参照してください。

サービスオーケストレーション AWS Step Functions のための の開始方法

次のリストには、Step Functions を使って最も一般的な AWS Data Pipeline ユースケースを実装するいくつかのサンプルプロジェクトの概要を示しています。これらのサンプルプロジェクトは、 から Step Functions AWS Data Pipeline への移行のリファレンスとして使用できます。また、これらを定型文として使用して独自のワークフローを構築し、ユースケースに基づいてサポートされている AWS のサービス ワークフローと統合することができます。

- Amazon EMR ジョブを管理する
- Amazon EMR Serverlessでのデータ処理ジョブの実行
- Hive/Pig/Hadoop ジョブの実行

- 大規模なデータセット (Amazon Athena、Amazon S3 AWS Glue、Amazon SNS) へのクエリ
- Amazon Redshift を使用した ETL/ELT ワークフローの実行
- AWS Glue クローラのオーケストレーション
- Step Functions を使用したシェルスクリプトの実行

Step Functions に関する詳細については、以下のトピックとリソースを参照してください。

- Step Functions チュートリアル
- Step Functions サンプルプロジェクト
- AWS Step Functions ワークショップ

料金比較

AWS Data Pipeline は、パイプラインの数とその使用レベルによって料金が設定されます。1 日に 2 回以上 (高頻度) 実行されるアクティビティの料金は、1 アクティビティにつき 1 か月あたり 1 USD です。1 日 1 回以下 (低頻度) に実行されるアクティビティの料金は、1 アクティビティにつき 1 か月あたり 0.60 USD です。アクティブではないパイプラインの料金は、パイプライン 1 つにつき 1 USD です。料金の詳細については、「AWS Data Pipeline 料金表」ページを参照してください。

Step Functions には標準と Express という 2 つのワークフロータイプがあります。ワークフローのタイプごとに料金モデルが異なります。この比較は、 の一般的なユースケースに最も適しているため、標準ワークフローに基づいています AWS Data Pipeline。標準ワークフローの料金は、状態遷移 1000 回あたり 0.025 USD です。アクティブではないステートマシンについては料金はかかりません。使用した分だけお支払いいただきます。料金の詳細については、「AWS Step Functions 料金表」ページを参照してください。

トラブルシューティング

Step Functions の操作中に問題が発生した場合は、次のトラブルシューティングのリソースを使用してください。

トピック

- 一般的なトラブルシューティング
- サービス統合のトラブルシューティング
- アクティビティのトラブルシューティング。
- Express ワークフローのトラブルシューティング

一般的なトラブルシューティング

ステートマシンを作成できません。

ステートマシンに関連付けられた IAM ロールには、十分な許可があるわけではないかもしれません。AWS サービス統合タスク、X-Ray、CloudWatch ログ記録を含む IAM ロールのアクセス許可をチェックします。`.sync` タスク状態には、追加許可が必要です。

JsonPath を使用して前のタスクの出力を参照できません。

JsonPath の場合、JSON キーは `.$` で終わる必要があります。つまり、JsonPath はキーバリューペアでのみ使用できます。JsonPath を配列などの他の場所で使用する場合は、組み込み関数を使用できます。例えば、以下のようなものを実行できます。

タスク A の出力:

```
{
    "sample": "test"
}
```

タスク B:

```

{
    "JsonPathSample.$": "$.sample"
}
```

> **Tip**
>
> Step Functions コンソールのデータフローシミュレーターを使用して、JSON パス構文をテストし、ステート内でのデータの操作方法に関する理解を深め、ステート間でデータを渡す方法を確認します。

状態遷移に遅延がありました。

Standard ワークフローでは、状態遷移の数に制限があります。状態遷移の制限を超えると、Step Functions はクォータのバケットがいっぱいになるまで、状態遷移を遅らせます。状態遷移制限のスロットリングは、CloudWatch メトリクスページの 実行メトリクス セクションにおける`ExecutionThrottled` メトリクス をレビューしてモニタリングできます。

新しいStandard ワークフローの実行をスタートすると、**`ExecutionLimitExceeded`** エラーがあれば、失敗します。

各 AWS リージョン の各 AWS アカウント の Step Functions オープン実行数は 1,000,000 回に制限されています。この制限を超えた場合、Step Functions は、`ExecutionLimitExceeded`というエラーをスローします。この制限は Express Workflow には適用されません。「Amazon CloudWatch ユーザーガイド」にある以下の「CloudWatch メトリクス数式」を使用して、オープン実行数はおおよそ `ExecutionsStarted - (ExecutionsSucceeded + ExecutionsTimedOut + ExecutionsFailed + ExecutionsAborted)` にできます。

並列状態の 1 つのブランチで障害が発生すると、実行全体が失敗となります。

これは想定される動作です。parallel 状態を使用するときに障害が発生しないようにするには、Step Functions が各ブランチからスローされるエラーを補足できるように設定します。

サービス統合のトラブルシューティング

ジョブはダウンストリームサービスで完了していますが、Step Functions ではタスクの状態が「進行中」のままになるか、完了が遅れます。

`.sync` サービス統合パターンについては、Step Functions は、EventBridge ルール、ダウンストリーム API、またはその両方の組み合わせを使用して、ダウンストリームジョブのステータスを検

出します。一部のサービスでは、Step Functions はモニタリングのため EventBridge ルールを作成しません。例えば、AWS Glue サービス統合のため、EventBridge ルールを使用する代わりに、Step Functions が `glue:GetJobRun` 呼び出しを行います。API 呼び出し頻度のため、ダウンストリームタスク完了と Step Functions タスクの完了時間は異なります。Step Functions には、EventBridge ルールを管理し、ダウンストリームサービスを呼び出すために IAM 許可が必要です。実行ロールへの許可が不十分であるために、タスクの完了にどれほどの影響がでているのかついての詳細は、[ジョブを実行] パターンを使用したタスクの追加許可 を参照してください。

ネストされたステートマシンの実行から JSON 出力を返したいと思っています。

Step Functions には、`startExecution.sync` と `startExecution.sync:2` の 2 つの Step Functions 同期サービス統合があります。どちらもネストされたステートマシンが完了するのを待機しますが、異なる `Output` 形式を返します。`startExecution.sync:2` を使用して、`Output` で JSON 出力を返すことができます。

Lambda 関数を別のアカウントから呼び出すことはできません。

クロスアカウントサポートによる Lambda 関数へのアクセス

リージョンで AWS リソースのクロスアカウントアクセスを利用できる場合は、次の方法を使用して別のアカウントから Lambda 関数を呼び出します。

ワークフローで、クロスアカウントリソースを呼び出すには次の操作を行います。

1. リソースを含むターゲットアカウントに IAM ロールを作成します。このロールは、ステートマシンを含むソースアカウントに、ターゲットアカウントのリソースにアクセスするアクセス許可を付与します。
2. `Task` ステートの定義で、クロスアカウントリソースを呼び出す前にステートマシンが引き受けるターゲット IAM ロールを指定します。
3. ターゲット IAM ロールの信頼ポリシーを変更して、ソースアカウントがこのロールを一時的に引き受けることができるようにします。信頼ポリシーには、ソースアカウントで定義したステートマシンの Amazon リソースネーム (ARN) を含める必要があります。また、AWS リソースを呼び出すための適切なアクセス許可をターゲット IAM ロールに定義してください。
4. ソースアカウントの実行ロールを更新して、ターゲット IAM ロールを引き受けるのに必要なアクセス許可を含めます。

例については「チュートリアル: AWS クロスアカウントリソースへのアクセス」を参照してください。

> **Note**
>
> 複数の AWS アカウント からリソースにアクセスするための IAM ロールを引き受けるようにステートマシンを設定できます。ただし、ステートマシンは指定された時間に 1 つの IAM ロールしか引き受けられません。

クロスアカウントリソースを指定する Task ステート定義の例については、「タスク状態の認証情報フィールドの例」を参照してください。

クロスアカウントサポートなしで Lambda 関数にアクセスする

リージョンで AWS リソースのクロスアカウントアクセスを利用できない場合は、次の方法を使用して別のアカウントから Lambda 関数を呼び出します。

Task 状態の `Resource` フィールドでは、パラメータ内で `arn:aws:states:::lambda:invoke` を使用し、`FunctionArn` を渡します。ステートマシンに関連付けられている IAM ロールには、クロスアカウント Lambda 関数 `lambda:invokeFunction` を呼び出す適切な許可が必要です。

```
{
   "StartAt":"CallLambda",
   "States":{
      "CallLambda":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-west-2:123456789012:function:my-function"
         },
         "End":true
      }
   }
}
```

`.waitForTaskToken` 状態から渡されたタスクークンが表示されません。

Task 状態の `Parameters` フィールドでは、タスクトークンを渡す必要があります。例えば、以下のようなコードを実行できます。

```
{
   "StartAt":"taskToken",
   "States":{
      "taskToken":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
         "Parameters":{
            "FunctionName":"get-model-review-decision",
            "Payload":{
               "token.$":"$$.Task.Token"
            },
         },
         "End":true
      }
   }
}
```

> **Note**
>
> 任意の API アクションを使用して、.waitForTaskToken の使用を試みることができます。ただし、一部の API には適切なパラメーターがありません。

アクティビティのトラブルシューティング。

ステートマシンの実行がアクティビティ状態でスタックされます。

アクティビティタスクの状態は、[GetActivityTask] API アクションを使用してタスクトークンをポーリングするまでスタートされません。ベストプラクティスとして、スタックの実行を回避するために、タスクレベルのタイムアウトを追加します。詳細については「タイムアウトを使用して実行のスタックを回避する」を参照してください。

ステートマシンが ActivityScheduled イベントで停止している場合は、アクティビティのワーカーフリートに問題があるか、スケールが不足していることを示しています。ActivityScheduleTime CloudWatch メトリクスをモニタリングし、その時間が長くなるとアラームを設定する必要があります。ただし、Activity 状態が ActivityStarted 状態に遷移しないようなステートマシンの実行をタイムアウトさせるには、ステートマシンレベルでタイムアウトを定義してください。これを行うには、States フィールドの外側のステートマシン定義の先頭に TimeoutSeconds フィールドを指定します。

タスクトークンを待っている間に、アクティビティワーカーがタイムアウトします。

ワーカーは、GetActivityTask API アクションを使用して、実行中のステートマシンにより、実行スケジュールがある指定されたアクティビティ ARN を持つタスクを取得する API アクション。GetActivityTask はロングポールをスタートするため、サービスは HTTP 接続を開いたままにして、タスクが使用可能になるとすぐに応答します。応答前にサービスがリクエストする最大時間は 60 秒です。60 秒以内に利用可能なタスクがない場合、ポーリングは Null 文字列を持つ taskToken を返します。このタイムアウトを回避するには、AWS SDK または API コールを活用するために用いるクライアントで 65 秒以上のタイムアウトがあるクライアント側のソケットを設定します。

Express ワークフローのトラブルシューティング

StartSyncExecution API コールからの応答を受信する前にアプリケーションをタイムアウトします。

API コール を行うために使用している AWS SDKまたはクライアントでクライアント側のソケットタイムアウトを設定します。応答を受信するには、タイムアウトには Express ワークフローの実行時間よりも長い値を指定する必要があります。

Express Workflow 障害をトラブルシューティングするために、実行履歴を表示できません。

Express ワークフローは、AWS Step Functions に実行履歴を記録しません。代わりに、CloudWatch ログ記録を有効にする必要があります。ログ記録を有効にすると、CloudWatch Logs Insights クエリを使用して、Express Workflow 実行への確認が可能になります。[実行] タブの [有効化] ボタンを選択すると、Express Workflow 実行の実行履歴を Step Functions コンソールに表示することもできます。詳細については「Step Functions コンソールでの実行の表示とデバッグ」を参照してください。

所要時間に基づいて実行を一覧表示するには:

```
fields ispresent(execution_arn) as exec_arn
| filter exec_arn
| filter type in ["ExecutionStarted", "ExecutionSucceeded", "ExecutionFailed",
 "ExecutionAborted", "ExecutionTimedOut"]
| stats latest(type) as status,
```

```
  tomillis(earliest(event_timestamp)) as UTC_starttime,
  tomillis(latest(event_timestamp)) as UTC_endtime,
  latest(event_timestamp) - earliest(event_timestamp) as duration_in_ms  by
 execution_arn
| sort duration desc
```

失敗した実行とキャンセルされた実行を一覧表示するには:

```
fields ispresent(execution_arn) as isRes | filter type in ["ExecutionFailed",
 "ExecutionAborted", "ExecutionTimedOut"]
```

関連情報

次の表は、本サービスを利用する際に役立つと思われる関連リソースの一覧です。

リソース	説明
AWS Step Functions API リファレンス	API アクション、パラメータ、データ型に関する説明と、サービスから返されるエラーのリスト。
AWS Step Functions コマンドラインリファレンス	AWS Step Functions で使用できる AWS CLI コマンドの説明。
Step Functions 製品情報	Step Functions に関する情報のメインのウェブページ。
ディスカッションフォーラム	Step Functions と他の AWS のサービスに関する技術的な質疑応答の場である開発者向けのコミュニティフォーラム。
AWS Support 情報	AWS のインフラストラクチャサービスでのアプリケーションの構築と実行を支援する AWS Support、 1 対 1 で対応が迅速なサポートチャネルに関する情報のメインウェブページです。

最近リリースされた機能

次の表には、Step Functions の新機能が利用できるリージョンの一覧が記載されています。

開始日	機能名	利用できるリージョン
2023 年 11 月 26 日	パブリック HTTPS エンドポイントを呼び出し、個々の状態をテストする	• 米国東部 (バージニア北部) – us-east-1 • 米国西部 (オレゴン) – us-west-2 • 米国東部 (オハイオ) – us-east-2 • 欧州 (アイルランド) – eu-west-1 • 欧州 (フランクフルト) – eu-central-1 • 欧州 (ストックホルム) - eu-north-1 • アジアパシフィック (シドニー) – ap-southeast-2 • アジアパシフィック (東京) – ap-northeast-1 • アジアパシフィック (シンガポール) – ap-southeast-1
2023 年 11 月 15 日	Redrive 実行	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。
2023 年 10 月 12 日	Amazon EMR Serverless 用に最適化された統合	この機能を利用できるすべてのリストについては、[リー

開始日	機能名	利用できるリージョン
		ジョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。
2023 年 9 月 7 日	強化されたエラー処理	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。
2023 年 8 月 31 日	効率化されたワークフローのオーサリング作業のための、Workflow Studio の強化	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。
2023 年 6 月 22 日	バージョニングとエイリアス	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。

開始日	機能名	利用できるリージョン
2023 年 6 月 16 日	新しい AWS SDK 統合	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。
2022 年 12 月 1 日	分散マップ状態によるデータ処理のための大規模な並列ワークフローのオーケストレーション	この機能を利用できるすべてのリストについては、[リージョン別の AWS のサービス] というタイトルのページにある [リージョン] ドロップダウンリストのオプションを参照してください。

ドキュメント履歴

このセクションでは、AWS Step Functions デベロッパーガイドの主な変更点を一覧表示します。

変更	説明	変更日
更新	Step Functions はオープンワークフローメトリックのサポートを追加します オープンワークフローメトリクスにより、進行中の標準ワークフローの数とオープンワークフローの上限をアカウントレベルで可視化できるようになりました。開始方法に関係なく、すべてのワークフローのワークロードを管理できるため、ワークフローをスムーズに運用できます。 CloudWatch アラームを設定してワークフローを監視し、制限に近づくと事前にアラートを受信できます。アラートを受け取ったら、特定のワークフローを停止したり、制限の引き上げをリクエストしたりするなどのアクションを実行することで、ワークフローを効果的に管理できます。 オープンワークフローメトリクスは、 CloudWatch 追加の設定なしで標準ワークフローで使用できます。詳細については、「実行メトリクス」を参照してください。	2024 年 2 月 29 日
更新	サービス統合の追加と更新。新規および更新された AWS SDK インテグレーションのリストについては、を参照してください。サポートされている AWS SDK インテグレーションの変更ログサービスの全リストについては、を参照してくださいサポートされている AWS SDK サービス統合。	2024 年 1 月 18 日
新機能	Application Composer の Workflow Studio で、AWS CloudFormation テンプレートを使用してサーバーレスワークフローを構築できます。詳細については、「Application Composer の Workflow Studio を使用する」を参照してください。	2023 年 11 月 27 日
新機能	Step Functions では、新しい Test State API を使用して、パブリック HTTPS エンドポイントを直接呼び出して、個々のステートをテストできるようになりました。詳細については、以下を参照してください。	2023 年 11 月 26 日

変更	説明	変更日
	• サードパーティーの API を呼び出す • TestState API を使用して状態をテストする	
新機能	Step Functions は Amazon Bedrock と統合するようになりました。詳細については、次のトピックを参照してください。 • Step Functions を使用して Amazon Bedrock を呼び出す • Amazon Bedrock の IAM アクセス許可 • Amazon Bedrock で AI プロンプトチェーンを実行する • AWS Step Functions 他のサービスとの併用	2023 年 11 月 26 日
新機能	Step Functions では、タイプ Standard のワークフロー実行を障害発生時点から再処理できるようになりました。詳細については、「Redriving の実行」および「マップ実行の再処理」を参照してください。	2023 年 11 月 15 日
ドキュメントのみの更新	Amazon EventBridge Scheduler を使用してステートマシンをスケジュールどおりに実行する方法を説明する新しいトピックを公開しました。詳細については、「AWS Step Functions での Amazon EventBridge スケジューラの使用」を参照してください。	2023 年 10 月 16 日
新機能	Step Functions は Amazon EMR Serverless と統合するようになりました。詳細については、次のトピックを参照してください。 • Step Functions を使用して Amazon EMR Serverless を呼び出す • EMR Serverless ジョブを実行する • Step Functions 用統合最適化 • AWS Step Functions 他のサービスとの併用	2023 年 10 月 12 日
ドキュメントのみの更新	Amazon EventBridge Scheduler を使用してステートマシンをスケジュールどおりに実行する方法についての情報を追加しました。詳細については、「EventBridge スケジューラの使用」を参照してください。	2023 年 10 月 5 日
更新	明確さ、簡潔さ、および新規ユーザー向けの明確なジャーニー マップの確立のために、分散マップ状態トピックを再構成および更新しました。詳細については、「マップステートを分散モードで使用して大規模な並行ワークロードをオーケストレーションする」を参照してください。	2023 年 10 月 6 日

変更	説明	変更日
修正内容	AWS CDK v2 を使用するようにチュートリアルのコードサンプルを修正しました。詳細については、「AWS CDK を使用してStep Functions 用 Lambda ステートマシンを作成する」を参照してください。	2023 年 9 月 19 日
更新	エラーを明確に識別し、より詳細に制御して再試行を実装するために Step Functions が導入した強化されたエラー処理機能についての情報を追加しました。詳細については、失敗およびエラー後の再試行を参照してください。	2023 年 9 月 7 日
更新	Step Functions は、ワークフローオーサリングエクスペリエンスを効率化するために Workflow Studio に機能強化を追加しました。詳細については、「AWS Step Functions ワークフロースタジオ」を参照してください。	2023 年 8 月 31 日
ドキュメントのみの更新	その `ExecutionsStarted` メトリクスについて報告された実際のメトリクス数の 2 倍に関する情報を追加しました。詳細については、「カウントを報告するメトリクス」を参照してください。	2023 年 7 月 25 日
ドキュメントのみの更新	Step Functions は、分散マップ状態の次の一般的な使用例を示す 2 つの新しいサンプルプロジェクトを追加しました。 • CSV ファイルの処理 • Amazon S3 バケット内のデータの処理	2023 年 7 月 17 日
ドキュメントのみの更新	Terraform を使用したステートマシンのデプロイに関する新しいトピックを公開しました。詳細については、「Terraform を使用する、ステートマシンのデプロイ」を参照してください。	2023 年 7 月 5 日
ドキュメントのみの更新。	Amazon EventBridge インターフェースの変更に合わせて、以下の手順を更新しました。 • Step Functions イベントを にルーティングする EventBridge • Amazon S3 イベント発生時にステートマシンの実行をスタートする	2023 年 6 月 26 日

変更	説明	変更日
新機能	Step Functions では、複数のステートマシンのバージョニングとエイリアス作成できるようになり、サーバーレスワークフローをデプロイする際の耐障害性が向上しました。詳細については、「バージョニングとエイリアスによる継続的デプロイの管理」を参照してください。	2023 年 6 月 22 日
ドキュメントのみの更新	`TimeoutSeconds` フィールドと `HeartbeatSeconds` フィールドの説明が改善され、それぞれの違いが説明されるようになりました。詳細については、「タスク状態フィールド」を参照してください。	2023 年 6 月 22 日
ドキュメントのみの更新	Parallel ステートと Map ステートの結果として通常返される配列の配列をフラット化する方法を説明する新しいセクションを公開しました。詳細については、「配列の配列の平坦化」を参照してください。	2023 年 6 月 20 日
更新	Step Functions は、7 つの新しい API AWS のサービス アクションと 468 の新しい API アクションを追加することで AWS SDK 統合のサポートを拡張しました。詳細については、サポートされている AWS SDK サービス統合およびサポートされている AWS SDK インテグレーションの変更ログを参照してください。	2023 年 6 月 16 日
ドキュメントのみの更新。	最近リリースされた Step AWS リージョン Functions の機能の一覧を記載した新しいトピックを公開しました。詳細については、「最近リリースされた機能」を参照してください。	2023 年 6 月 16 日
ドキュメントのみの更新。	Step Functions AWS のサービス に AWS User Notifications、に関するセクションが追加されました。このセクションは、 AWS 内の通知の中心となる場所です AWS Management Console。詳細については、「Step Functions で AWS User Notifications を使用する」を参照してください。	2023 年 5 月 4 日
ドキュメントのみの更新	AWS Key Management Service (AWS KMS) キーで暗号化された Amazon S3 バケットに子ワークフローの実行結果を書き込むために必要な権限について説明する新しいセクションを追加しました。詳細については、「AWS KMS key 暗号化された Amazon S3 バケットの IAM アクセス許可」を参照してください。	2023 年 4 月 25 日

変更	説明	変更日
ドキュメントのみの更新	データフローシミュレーター機能について説明する新しいトピックを追加しました。詳細については、「データフローシミュレータ」を参照してください。	2023 年 4 月 14 日
クォータの更新	各アカウントのオープンマップ実行のデフォルトクォータ 1000 件についての情報を追加しました。詳細については、「アカウントに関連するクォータ」を参照してください。	2023 年 4 月 5 日
ドキュメントのみの更新。	AWS Data Pipeline ワークロードを Step Functions に移行するタイミングを説明するトピックを追加しました。このトピックでは、移行の実行方法を説明する例のリストも提供しています。詳細については、「から Step Functions AWS Data Pipeline へのワークロードの移行」を参照してください。	2023 年 3 月 30 日
ドキュメントのみの更新	分散マップ状態で X-Ray トレースを使用できないことに関する注記を追加しました。詳細については、「AWS X-Ray および Step Functions」を参照してください。	2023 年 3 月 21 日
ドキュメントのみの更新	Step Functions がタグベースの認証を処理する方法についての情報を追加しました。詳細については、Step Functions でのタグ付けおよびタグベースのポリシーを参照してください。	2023 年 3 月 15 日
ドキュメントのみの更新	Step Functions が分散マップ状態で入力として使用される CSV ファイルを解析する方法についての情報を追加しました。詳細については、「Amazon S3 バケット内の CSV ファイル」を参照してください。	2023 年 3 月 14 日
ドキュメントのみの更新	Step Functions がジョブを実行 (.sync) パターンのクロスアカウント呼び出しを処理する方法に関する情報を追加しました。詳細については、「ジョブを実行 (.sync)」を参照してください。	2023 年 3 月 1 日
ドキュメントのみの更新	完了したワークフロー実行の履歴保持期間を 90 日から 30 日に短縮します。ログ保持期間の変更の詳細については、「実行の保証」と「ステートマシンの実行に関連するクォータ」を参照してください。	2023 年 2 月 21 日

変更	説明	変更日
更新	Step Functions は、35 AWS のサービスと 1100 の新しい API アクションを追加することで、 AWS SDK インテグレーションのサポートを拡張しました。詳細については、サポートされている AWS SDK サービス統合およびサポートされている AWS SDK インテグレーションの変更ログを参照してください。	2023 年 2 月 17 日
ドキュメントのみの更新	Step Functions を使用してクレジットカード申請のワークフローを作成するプロセスを説明する入門チュートリアルシリーズを公開しました。詳細については、「AWS Step Functions の開始方法」を参照してください。	2022 年 12 月 30 日
新機能	Step Functions では、Map ステート用の新しい分散モードを使用して、データ処理の大規模な並列ワークフローをオーケストレーションするサポートが追加されました。詳細については、「マップステートを分散モードで使用して大規模な並行ワークロードをオーケストレーションする」を参照してください。	2022 年 12 月 1 日
新機能	Step Functions は、 AWS 他のアカウントで設定されているクロスアカウントリソースへのアクセスをサポートするようになりました。詳細については、以下を参照してください。 • AWS アカウント ワークフロー内の他のリソースへのアクセス • チュートリアル: AWS クロスアカウントリソースへのアクセス • Task state	2022 年 11 月 18 日
更新	Step Functions は、Express ワークフローの実行を表示およびデバッグするための新しいコンソールエクスペリエンスを提供するようになりました。詳細については、次を参照してください。 • コンソールでの Standard ワークフローと Express ワークフローの実行 • Step Functions コンソールでの実行の表示とデバッグ	2022 年 10 月 18 日

変更	説明	変更日
更新	Amazon EMR に最適化されたサービス統合のために、addStep および addStep.sync API を使用する際にオプションで ExecutionRoleArn パラメータを指定するサポートが追加されました。詳細については、「Step Functions を使用して Amazon EMR を呼び出す」を参照してください。	2022 年 9 月 20 日
ドキュメントのみの更新	Step Functions を使用してサーバーレスワークフローを構築する際のコストの最適化に関する推奨事項を提供する新しいトピックを追加しました。詳細については、「Express ワークフローによるコスト最適化」を参照してください。	2022 年 9 月 15 日
更新	Step Functions では、配列操作、データのエンコードとデコード、ハッシュ計算、JSON データ操作、数学関数演算、固有識別子の生成など、データ処理タスクを実行するための 14 件の新しい組み込み関数のサポートが追加されました。 ドキュメントのみの更新。 既存の組み込み関数と新しく導入された組み込み関数を、実行に役立つデータ処理タスクの種類に基づいて次のカテゴリに分類しました。 • 配列の組み込み関数 • データのエンコードとデコードのための組み込み関数 • ハッシュ計算用組み込み関数 • JSON データ操作用の組み込み関数 • 数学演算用組み込み関数 • String 演算の組み込み関数 • 一意の ID を生成するための組み込み関数 • 汎用操作用の組み込み関数 詳細については、「組み込み関数」を参照してください。	2022 年 8 月 31 日

変更	説明	変更日
更新	Step Functions は、AWS Billing Conductor、Amazon GameSparksおよび Amazon Pinpoint SMS and Voice V2 3 AWS つのサービスを追加することで AWS SDK インテグレーションのサポートを拡張しました。詳細については、「サポートされている AWS SDK インテグレーションの変更ログ」を参照してください。	2022 年 7 月 26 日
ドキュメントのみの更新	Step Functions がサポートする AWS SDK インテグレーションに加えられたすべての更新の概要を含む新しいトピックを追加しました。詳細については、「サポートされている AWS SDK インテグレーションの変更ログ」を参照してください。	2022 年 7 月 26 日
ドキュメントのみの更新	AWS Step Functions 開発者ガイドには、Express Workflow 専用に発行される実行メトリクスに関する詳細が含まれるようになりました。詳細については、「Express ワークフローの実行メトリクス」を参照してください。	2022 年 6 月 9 日

変更	説明	変更日
更新	Step Functions コンソールの強化 コンソールの [実行の詳細] ページが再設計され、以下の機能強化が行われました。 • 実行に失敗した理由を一目で特定できます。 • ステートマシン用の新しい可視化モードが 2 つあります。[テーブルビュー] と [イベントビュー]です。これらのビューでは、フィルターを適用して関心のある情報のみを表示することもできます。さらに、イベントのタイムスタンプに基づいて [イベントビュー] の内容をソートできます。 • [グラフビュー] モードではドロップダウンリストを使用するか、[テーブルビュー] モードの Map ステートのツリービューで、Map ステートのさまざまな反復を切り替えます。 • 完全な入出力データ転送パスや Task または Parallel ステートの再試行など、ワークフローの各ステートに関する詳細情報が表示されます。 • ステートマシンの実行 Amazon Resource Name をコピーするオプション、ステートマシンの合計遷移回数を表示するオプション、実行の詳細を JSON 形式でエクスポートするオプションなど、さまざまな機能強化が行われています。 ドキュメントのみの更新。 [実行の詳細] ページに表示されるさまざまなタイプの情報を説明する新しいトピックを追加しました。また、この情報を調べる方法を示すチュートリアルを追加しました。詳細については、以下を参照してください。 • Step Functions コンソールでの実行の表示とデバッグ • チュートリアル: Step Functions コンソールを使用したステートマシン実行の調査	2022 年 5 月 9 日

変更	説明	変更日
更新	Step Functions は、エンティティ (サービスまたはアカウント) が別のエンティティによってアクションを実行するように強制された場合に発生する、混乱を招く副セキュリティ問題の回避策を提供するようになりました。詳細については、以下を参照してください。 • サービス間の混乱した代理の防止	2022 年 5 月 2 日

更新	• Step Functions は、さらに 21 AWS のサービスを追加することで AWS SDK インテグレーションのサポートを拡張しました。詳細については、「サポートされている AWS SDK サービス統合」を参照してください。
	• ドキュメントのみの更新。
	  • Step Functions と AWS SDK サービスの統合を誤って実行したときに生成される例外に存在するすべての例外プレフィックスのリストを追加しました。詳細については、「サポートされている AWS SDK サービス統合」を参照してください。
	  • サポートされている SDK インテグレーションのサポートされていない API アクションのリストを追加しました。 AWS 詳細については、「サポートされているサービスでサポートされていない API アクション」を参照してください。
	  • サポート対象のうち、現在廃止されているすべての AWS SDK インテグレーションのリストを追加しました。詳細については、「非推奨の AWS SDK サービス統合」を参照してください。
		2022 年 4 月 19 日

新機能	Step Functions Local は AWS SDK インテグレーションとサービスインテグレーションのモックをサポートするようになりました。詳細については、以下を参照してください。 • モックサービス統合の使用	2022 年 1 月 28 日

変更	説明	変更日
新機能	AWS Step Functions を使用したバックエンド統合として、同期エクスプレスステートマシンによる Amazon API Gateway REST API の作成がサポートされるようになりました。 AWS Cloud Development Kit (AWS CDK)詳細については、以下を参照してください。 • を使用した同期 Express ステートマシンを使用した API Gateway REST API の作成 AWS CDK	2021 年 12 月 10 日
更新	Step Functions は、Step Functions と Amazon Athena のアップグレードされたコンソールを示す 3 件の新しいサンプルプロジェクトを追加しました。詳細については、以下を参照してください。 • 複数のクエリを実行します (Amazon Athena、Amazon SNS) • 大規模なデータセット (Amazon Athena、Amazon S3 AWS Glue、Amazon SNS) へのクエリ • データを最新の状態に保つ (Amazon Athena、Amazon S3、 AWS Glue)	2021 年 11 月 22 日
新機能	Step Functions は、同期 Express ワークフローの Amazon VPC エンドポイントサポートを追加しました。詳細については、以下を参照してください。 • Step Functions の Amazon VPC エンドポイント	2021 年 11 月 15 日
更新	AWS Step Functions には、Step Functions AWS Batch インテグレーションの使用方法を示す 3 つの新しいサンプルプロジェクトが追加されました。詳細については、以下を参照してください。 • AWS Batch ジョブのファンアウト • AWS Batch Lambda 付き • エラー処理で Step Functions と AWS Batch を使用する	2021 年 10 月 14 日

変更	説明	変更日
新機能	AWS Step Functions には AWS SDK インテグレーションが追加され、200 AWS を超えるすべてのサービスで API アクションを使用できるようになりました。詳細については、以下を参照してください。 • AWS SDK サービスインテグレーション • AWS SDK サービスインテグレーションを使用して Amazon S3 バケット情報を収集する	2021 年 9 月 30 日
新機能	AWS Step Functions には、ビジュアルワークフローデザイナー、Workflow Studio が追加されました。 AWS Step Functions 詳細については、以下を参照してください。 • AWS Step Functions ワークフロースタジオ • AWS Step Functions ワークフロースタジオの使い方を学びましょう。	2021 年 6 月 17 日
更新	AWS Step Functions 4 つの新しい API、`StartBuildBatch` 、`StopBuildBatch` `RetryBuildBatch` `DeleteBuildBatch` 、 CodeBuild がインテグレーションに追加されました。詳細については、以下を参照してください。 • Step AWS CodeBuild Functions による呼び出し	2021 年 6 月 4 日
新機能	AWS Step Functions Amazon EventBridge と統合されるようになりました。詳細については、以下を参照してください。 • Step EventBridge Functions による呼び出し • Step Functions と Amazon の IAM ポリシー EventBridge の IAM ポリシー • カスタムイベントの送信先 EventBridge の方法を示すサンプルプロジェクト	2021 年 5 月 14 日

変更	説明	変更日
更新	AWS Step Functions Step Functions と Amazon Redshift データ API を使用して ETL/ELT ワークフローを実行する方法を示す新しいサンプルプロジェクトが追加されました。詳細については、以下を参照してください。 • Amazon Redshift (Lambda、Amazon Redshift データ API) を使用して ETL/ELT ワークフローを実行する	2021 年 4 月 16 日
新機能	AWS Step Functions コンソールに新しいデータフローシミュレーターが追加されました。詳細については、以下を参照してください。 • Step Functions コンソール	2021 年 4 月 8 日
新機能	AWS Step Functions EKS 上の Amazon EMR と統合されるようになりました。詳細については、以下を参照してください。 • EKS のAmazon EMR に電話をかけるには AWS Step Functions	2021 年 3 月 29 日
更新	ステートマシン定義の YAML サポートが AWS Toolkit for Visual Studio Code そして AWS CloudFormationに追加されました。詳細については、以下を参照してください。 • 定義形式のサポート • AWS Toolkit for Visual Studio Code	2021 年 3 月 4 日
新機能	AWS Step Functions と統合されるようになりました。 AWS Glue DataBrew詳細については、以下を参照してください。 • Step Functions AWS Glue DataBrew によるジョブの管理 • とは何ですか AWS Glue DataBrew? DataBrew 開発者ガイドにあります。	2021 年 1 月 6 日

変更	説明	変更日
新機能	AWS Step Functions Synchronous Express ワークフローが利用可能になり、マイクロサービスを簡単にオーケストレーションできるようになりました。詳細については、以下を参照してください。 • 同期および非同期 Express ワークフロー • 同期 Express ワークフローを呼び出す の方法を示すサンプルプロジェクト • API StartSyncExecutionドキュメント。	2020 年 11 月 24 日
新機能	AWS Step Functions Amazon API Gateway と統合されるようになりました。詳細については、以下を参照してください。 • Step Functions を使用して API Gateway を呼び出し • Step Functions と Amazon API Gateway IAM ポリシー の IAM ポリシー • API Gateway を呼び出す の方法を示すサンプルプロジェクト	2020 年 11 月 17 日
新機能	AWS Step Functions Amazon Elastic Kubernetes Service 統合されるようになりました。詳細については、以下を参照してください。 • Step Functions で Amazon EKS を呼び出す • Step Functions と Amazon EKS の IAM ポリシー の IAM ポリシー • Amazon EKS クラスターの管理 の方法を示すサンプルプロジェクト	2020 年 11 月 16 日
新機能	AWS Step Functions Amazon Athena と統合されるようになりました。詳細については、以下を参照してください。 • Step Functions で Athena を呼び出す • Step Functions と Amazon Athena の IAM ポリシー の IAM ポリシー • Athena クエリをスタートする の方法を示すサンプルプロジェクト	2020 年 10 月 22 日

変更	説明	変更日
新機能	AWS Step Functions end-to-end とのトレーシングワークフローをサポートするようになり AWS X-Ray、ステートマシンの実行状況を完全に把握できるようになり、分散アプリケーションの分析とデバッグが容易になりました。詳細については、以下を参照してください。 • AWS X-Ray および Step Functions • Step Functions と の IAM ポリシー AWS X-Ray の IAM ポリシー • AWS Step Functions API リファレンス • TracingConfiguration	2020 年 9 月 14 日

変更	説明	変更日

更新	AWS Step Functions UTF-8 でエンコードされた文字列として最大 256 KB のデータのペイロードサイズをサポートするようになりました。こうして、標準ワークフローと Express ワークフローの両方で、より大きなペイロードを処理できます。
	
	より大きなペイロードを使用するために、既存のステートマシンを変更する必要はありません。ただし、更新された API を使用するには、Step Functions SDK および Local Runner の最新バージョンに更新する必要があります。詳細については、以下を参照してください。
	
	• クォータ
	• the section called “ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する”
	• `States.DataLimitExceeded`
	• the section called “CloudWatch Logs ペイロード”
	• the section called “EventBridge ペイロード”
	• AWS Step Functions API リファレンス
	
	  • CloudWatchEventsExecutionDataDetails
	  • HistoryEventExecutionDataDetails
	  • GetExecutionHistory
	  • ActivityScheduledEventDetails
	  • ActivitySucceededEventDetails
	  • CloudWatchEventsExecutionDataDetails
	  • ExecutionSucceededEventDetails
	  • LambdaFunctionScheduledEventDetails
	  • ExecutionSucceededEventDetails
	  • StateEnteredEventDetails
	  • StateExitedEventDetails
	  • TaskSubmittedEventDetails
	  • TaskSucceededEventDetails
		2020 年 9 月 3 日

変更	説明	変更日

更新	Amazon ステートメント言語 は次のように更新されました。
	• 選択ルール が追加
	  • Null 比較演算子、IsNull。IsNull は JSON Null 値に対してテストし、前の状態の出力が NULL かどうかを検出するために使用できます。
	  • 他にも 4 つの新しい演算子、、 IsBoolean、 IsNumericが追加されました。 IsString IsTimestamp
	  • IsPresent 演算子を使ったフィールドの存在または非存在のテスト。IsPresent を使って、存在しないキーへのアクセスを試みたときの States.Runtime エラーを防止できます。
	  • ワイルドカードパターンマッチングにより、1 つ以上のワイルドカードがあるパターンに対する文字列比較をサポートします。
	  • サポートされている比較演算子の 2 つの変数の比較。
	• Task 状態のタイムアウト値とハートビートの値は、TimeoutSecondsPath と HeartbeatSecondsPath フィールドを使う固定値ではなく 状態入力から動的に提供できるようになりました。詳細は、タスク を参照してください。
	• 新しい ResultSelector フィールドは、ResultPath を適用する前の状態の結果を操作する方法を提供します。ResultSelector フィールドは、マッピング、並行、および タスク 状態のオプションフィールドです。
	• 組み込み関数 は、Task 状態がなくてもベーシックな操作を許可するために追加されました。組み込み関数は、Parameters フィールドと ResultSelector フィールド内で使用できます。
		2020 年 8 月 13 日

更新	AWS Step Functions Amazon SageMaker CreateProcessingJob API コールをサポートするようになりました。詳細については、以下を参照してください。 • SageMaker Step Functions による管理 • データを前処理し、機械学習モデルをトレーニングする、CreateProcessingJob を実証するサンプルプロジェクトです。	2020 年 8 月 4 日

変更	説明	変更日
新機能	AWS Step Functions がサポートされるようになったため AWS Serverless Application Model、ワークフローオーケストレーションをサーバーレスアプリケーションに簡単に統合できるようになりました。詳細については、以下を参照してください。 • AWS Step Functions および AWS SAM • AWS::Serverless::StateMachine • AWS SAM ポリシーテンプレート	2020 年 5 月 27 日
新機能	AWS Step Functions Step Functions の実行をネストするための新しい同期呼び出しが導入されました。新しい呼び出し `arn:aws:states:::states:startExecution.sync:2` は、JSON オブジェクトを返します。元の呼び出し `arn:aws:states:::states:startExecution.sync` は引き続きサポートされ、JSON のエスケープされた文字列を返します。詳細については、以下を参照してください。 • AWS Step Functions 実行を統合サービスとして管理	2020 年 5 月 19 日
新機能	AWS Step Functions AWS CodeBuildと統合されるようになりました。詳細については、以下を参照してください。 • AWS Step Functions 他のサービスとの併用 • Step AWS CodeBuild Functions による呼び出し • Step Functions 用統合最適化	2020 年 5 月 5 日
新機能	Step Functions は、AWS Toolkit for Visual Studio Code でサポートされるようになり、コードエディタを離れなくても、ステートマシンベースのワークフローを簡単に作成および視覚化できます。	2020 年 3 月 31 日
更新	標準ワークフローの Amazon CloudWatch Logs への口ギングを設定できるようになりました。詳細については、以下を参照してください。 • CloudWatch Logs を使用した口グ記録	2020 年 2 月 25 日

変更	説明	変更日
新機能	AWS Step Functions パブリック IP アドレスを必要とせずに、Amazon Virtual Private Cloud (VPC) から直接アクセスできるようになりました。詳細については、以下を参照してください。 • Step Functions の Amazon VPC エンドポイント	2019 年 12 月 23 日

変更	説明	変更日

新機能	Express ワークフローは新しいワークフロータイプで、IoT データの取り込み、ストリーミングデータ処理と変換、モバイルアプリケーションのバックエンドなど、大容量のイベント処理ワークロードに適しています。
	
	詳細については、次の新規および更新されたトピックを参照してください。
	
	• 標準ワークフロー対 Express ワークフロー
	  • 実行の保証
	• AWS Step Functions 他のサービスとの併用
	  • Step Functions 用統合最適化
	• Amazon SQS からの大容量メッセージの処理 (Express ワークフロー)
	• 選択的チェックポイントの例 (Express ワークフロー)
	• クォータ
	  • クォータ
	• CloudWatch Logs を使用したログ記録
	• AWS Step Functions API リファレンス
	  • CreateStateMachine
	  • UpdateStateMachine
	  • DescribeStateMachine
	  • DescribeStateMachineForExecution
	  • StopExecution
	  • DescribeExecution
	  • GetExecutionHistory
	  • ListExecutions
	  • ListStateMachines
	  • StartExecution
	  • CloudWatchLogsLogGroup
	  • LogDestination
		2019 年 12 月 3 日

変更	説明	変更日
	• LoggingConfiguration	
新機能	AWS Step Functions Amazon EMR と統合されるようになりました。詳細については、以下を参照してください。 • AWS Step Functions 他のサービスとの併用 • Step Functions を使用して Amazon EMR を呼び出す • Step Functions 用統合最適化	2019 年 11 月 19 日

更新	AWS Step Functions AWS Step Functions データサイエンス SDK をリリースしました。詳細については、以下を参照してください。
	• GitHub のプロジェクト
	• SDK ドキュメント
	• 以下のサンプルノートブックは、SageMakerGitHub コンソールと関連プロジェクトで利用できます。
	  • hello_world_workflow.ipynb
	  • machine_learning_workflow_abalone.ipynb
	  • training_pipeline_pytorch_mnist.ipynb
		2019 年 11 月 7 日

更新	Step Functions は Amazon でより多くの API アクションをサポートするようになり SageMaker、機能を実証するための 2 つの新しいサンプルプロジェクトが追加されました。詳細については、以下を参照してください。 • SageMaker Step Functions による管理 • AWS Step Functions 他のサービスとの併用 • 機械学習モデルのトレーニング • 機械学習モデルのチューニング	2019 年 10 月 3 日

変更	説明	変更日
新機能	Step Functions は、統合サービス API として `StartExecution` を呼び出すことで、新しいワークフロー実行の開始をサポートします。以下を参照してください。 • タスク状態からワークフロー実行を開始する • AWS Step Functions 実行を統合サービスとして管理 • AWS Step Functions 他のサービスとの併用 • Step Functions ワークフロー実行スタート用 IAM ポリシー	2019 年 8 月 12 日
新機能	Step Functions には、統合サービスにタスクトークンを渡し、`SendTaskSuccess` または `SendTaskFailure` でそのタスクトークンが返されるまで実行を一時停止する能力が含まれています。以下を参照してください。 • サービス統合パターン • タスクトークンのコールバックまで待機する • コールバックパターンの例 (Amazon SQS、Amazon SNS、Lambda) • Step Functions 用統合最適化 • 人間による承諾プロジェクト例をデプロイする • サービス統合メトリクス Step Functions は、状態の定義の `"Parameters"` フィールド内で直接に現在の実行に関する動的な情報にアクセスする方法を提供するようになりました。以下を参照してください。 • コンテキストオブジェクト • コンテキストオブジェクトノードをパラメータとして渡す	2019 年 5 月 23 日
新機能	Step Functions は、 CloudWatch 実行ステータスを変更するためのイベントをサポートしています。以下を参照してください。 • EventBridge Step Functions 実行ステータス変更の (CloudWatch イベント) • Amazon CloudWatch イベントユーザーガイド	2019 年 5 月 8 日

変更	説明	変更日
新機能	Step Functions がタグを使用して IAM の許可をサポートするようになりました。詳細については、以下を参照してください。 • Step Functions でのタグ付け • タグベースのポリシー	2019 年 3 月 5 日
新機能	Step Functions Local が利用可能になりました。テストと開発のためにローカルマシンで Step Functions を実行できます。Step Functions Local は Java アプリケーションまたは Docker イメージとしてダウンロードできます。ステートマシンのローカルテスト を参照してください。	2019 年 2 月 4 日
新機能	AWS Step Functions 北京と寧夏地域でご利用いただけるようになりました。サポートされるリージョン を参照してください。	2018 年 1 月 15 日
新機能	Step Functions は、コスト配分を追跡するのに役立つリソースタグ付けをサポートします。ステートマシンへのタグ付けは、[Details] (詳細) ページで、または API アクションを使用して行うことができます。Step Functions でのタグ付け を参照してください。	2019 年 1 月 7 日
新機能	AWS Step Functions ヨーロッパ (パリ) と南米 (サンパウロ) リージョンでご利用いただけるようになりました。サポートされるリージョン を参照してください。	2018 年 12 月 13 日
新機能	AWS Step Functions ヨーロッパ (ストックホルム) リージョンでご利用いただけるようになりました。サポートされているリージョンのリストについては、「サポートされるリージョン」を参照してください。	2018 年 12 月 12 日

変更	説明	変更日
新機能	Step Functions AWS が一部のサービスと統合されるようになりました。Amazon ステートメント言語のタスク状態から、これらの統合サービスの API にパラメータを直接呼び出して渡すことができるようになりました。詳細については、以下を参照してください。 • AWS Step Functions 他のサービスとの併用 • サービス API にパラメータを渡す • Step Functions 用統合最適化	2018 年 11 月 29 日
更新	タスク状態のドキュメントの `TimeoutSeconds` および `HeartbeatSeconds` の説明を見直しました。タスク を参照してください。	2018 年 10 月 24 日
更新	実行履歴の最大サイズの制限についての説明を改善し、関連するベストプラクティスのトピックへのリンクを追加しました。 • ステートマシンの実行に関連するクォータ • 履歴のクォータに到達しないようにする	2018 年 10 月 17 日
更新	AWS Step Functions ドキュメントに新しいチュートリアルを追加しました。「」を参照してくださいAmazon S3 イベント発生時にステートマシンの実行をスタートする。	2018 年 9 月 25 日
更新	上限ドキュメントから、Step Functions コンソールに表示される最大実行エントリを削除しました。クォータ を参照してください。	2018 年 9 月 13 日
更新	AWS Step Functions アクティビティタスクのポーリング時のレイテンシーの改善に関するベストプラクティスのトピックをドキュメントに追加しました。アクティビティタスクのポーリング時のレイテンシーを回避する を参照してください。	2018 年 8 月 30 日
更新	AWS Step Functions アクティビティとアクティビティワーカーに関するトピックを改善しました。アクティビティ を参照してください。	2018 年 8 月 29 日

変更	説明	変更日
更新	AWS Step Functions CloudTrail 統合に関するトピックを改善しました。AWS CloudTrail を使用して Step Functions をログ記録 を参照してください。	2018 年 8 月 7 日
更新	JSON AWS CloudFormation の例をチュートリアルに追加しました。AWS CloudFormationを使用して Step Functions 用の Lambda ステートマシンを作成する を参照してください。	2018 年 23 月 6 日
更新	Lambda サービスエラー処理に関する新しいトピックが追加されました。Lambda サービスの例外を処理する を参照してください。	2018 年 6 月 20 日
新機能	AWS Step Functions アジアパシフィック (ムンバイ) リージョンで利用できるようになりました。サポートされているリージョンのリストについては、「サポートされるリージョン」を参照してください。	2018 年 6 月 28 日
新機能	AWS Step Functions AWS GovCloud (米国西部) リージョンで利用できるようになりました。サポートされているリージョンのリストについては、「サポートされるリージョン」を参照してください。 AWS GovCloud (米国西部) リージョンでの Step Functions の使用については、を参照してくださいAWS GovCloud (US)。	2018 年 6 月 28 日
更新	`Parallel` 状態のエラー処理に関するドキュメントを改良しました。エラー処理 を参照してください。	2018 年 6 月 20 日
更新	Step Functions での入出力処理に関するドキュメントが改良されました。`InputPath` 、`ResultPath` 、`OutputPath` を使用して、ワークフロー、状態、タスクにおける JSON のフローを制御する方法について説明します。以下を参照してください。 • Step Functions の入出力処理 • ResultPath	2018 年 6 月 7 日
更新	並行状態のサンプルコードを改良しました。並行 を参照してください。	2018 年 6 月 4 日

変更	説明	変更日
新機能	API CloudWatch とサービスのメトリクスをでモニタリングできるようになりました。を使用した Step Functions のモニタリング CloudWatch を参照してください。	2018 年 5 月 25 日
更新	`StartExecution` 、`StopExecution` 、および `StateTransition` のスロットリング制限が、次のリージョンで引き上げられました。 • 米国東部 (バージニア北部) • 米国西部 (オレゴン) • 欧州 (アイルランド) 詳細については、クォータ を参照してください。	2018 年 5 月 16 日
新機能	AWS Step Functions 米国西部 (北カリフォルニア) リージョンとアジアパシフィック (ソウル) リージョンで利用できるようになりました。サポートされているリージョンのリストについては、サポートされるリージョン を参照してください。	2018 年 5 月 5 日
更新	インターフェイスの変更に合わせて手順と画像を更新しました。	2018 年 4 月 25 日
更新	作業を継続するために、新しい実行を開始する方法を示す新しいチュートリアルが追加されました。長時間実行されているワークフロー実行を新しい実行として継続する を参照してください。このチュートリアルでは、サービスの制限を回避するための設計パターンについて説明します。履歴のクォータに到達しないようにする を参照してください。	2018 年 4 月 19 日
更新	ステートマシンに関する概念情報を追加し、状態に関するドキュメントの概要を改善しました。状態 を参照してください。	2018 年 3 月 9 日

変更	説明	変更日
更新	HTML、PDF、および Kindle に加えて、 AWS Step Functions 開発者ガイドもで入手できます。 GitHubフィードバックを残すには、 GitHub右上隅にあるアイコンを選択します。 	2018 年 3 月 2 日
更新	Step Functions に関するその他のリソースを説明するトピックを追加しました。 関連情報 を参照してください。	2018 年 2 月 20 日

新機能	• 新しいステートマシンを作成する場合、Lambda 関数へのアクセスを許可する IAM ロールを AWS Step Functions で作成することを承認する必要があります。
	• 以下のチュートリアルを更新し、ステートマシンの作成ワークフローにおける小さな変更を反映しました。
	  • Lambda を使用する Step Functions ステートマシン状態の作成
	  • Step Functions を使用してアクティビティステートマシンを作成する
	  • Step Functions ステートマシンを使用してエラー条件を処理する
	  • Lambda を使用してループを反復処理する
		2018 年 2 月 19 日

更新	Ruby で記述されたサンプルアクティビティワーカーについて説明するトピックを追加しました。この実装は、Ruby アクティビティワーカーを直接作成するために使用するか、別の言語でアクティビティワーカーを作成するための設計パターンとして使用できます。 Ruby のサンプルアクティビティワーカー を参照してください。	2018 年 2 月 6 日
更新	Lambda 関数を使用してカウントを反復する設計パターンについて説明する新しいチュートリアルを追加しました。 Lambda を使用する Step Functions ステートマシン状態の作成 を参照してください。	2018 年 1 月 31 日

変更	説明	変更日
更新	IAM 許可に関するコンテンツを更新し、DescribeStateMachineForExecution および UpdateStateMachine API を追加しました。 管理者以外のユーザー用の詳細な IAM 許可の作成 を参照してください。	2018 年 1 月 26 日
更新	新たに利用可能となったリージョンを追加しました:カナダ (中部)、アジアパシフィック (シンガポール)。 サポートされるリージョン を参照してください。	2018 年 1 月 25 日
更新	チュートリアルと手順を更新し、Step Functions をロールとして選択する許可が IAM から下りたことを反映しました。	2018 年 1 月 24 日
更新	状態間で大きいペイロードを渡さないことを推奨する新しいベストプラクティストピックを追加しました。 ラージペイロードを渡す代わりに Amazon S3 ARNs を使用する を参照してください。	2018 年 1 月 23 日
更新	ステートマシンを作成するための更新されたインターフェイスに合わせて手順を修正しました。 • Lambda を使用する Step Functions ステートマシン状態の作成 • Step Functions を使用してアクティビティステートマシンを作成する • Step Functions ステートマシンを使用してエラー条件を処理する	2018 年 1 月 17 日

変更	説明	変更日
新機能	サンプルプロジェクトを使用して、ステートマシンとすべての関連 AWS リソースをすばやくプロビジョンできます。Step Functions サンプルプロジェクト を参照してください。 使用可能なサンプルプロジェクトには、以下が含まれます。 • Job ステータスの投票 (Lambda、) AWS Batch • タスクタイマー (Lambda、Amazon SNS) **Note** これらのサンプルプロジェクトおよび関連ドキュメントで、同じ機能の実装について説明したチュートリアルを置き換えました。	2018 年 1 月 11 日
更新	実行のスタックの回避に関する情報を含むベストプラクティスセクションを追加しました。Step Functions のベストプラクティス を参照してください。	2018 年 1 月 5 日
更新	再試行が料金にどのように影響を与えるかに関する注意を追加しました。 **Note** 再試行は状態遷移として扱われます。状態遷移が課金に及ぼす影響については、Step Functions コストを参照してください。	2017 年 12 月 8 日

変更	説明	変更日
更新	リソース名に関する情報を追加しました。 **Note** Step Functions では、ステートマシン、実行、アクティビティの名前と、非 ASCII 文字を含むラベルを作成できます。これらの非ASCII名はAmazonでは機能しません。 CloudWatch CloudWatch メトリクスを追跡できるようにするには、ASCII 文字のみを使用する名前を選択してください。	2017 年 12 月 6 日
更新	セキュリティの概要情報を改訂し、詳細な IAM アクセス権限に関するトピックを追加しました。「のセキュリティ AWS Step Functions」および「管理者以外のユーザー用の詳細な IAM 許可の作成」を参照してください。	2017 年 11 月 27 日
新機能	既存のステートマシンを更新できます。「ステートマシンを更新する」を参照してください。	2017 年 11 月 15 日

変更	説明	変更日
更新	`Lambda.Unknown` エラーを明確にするための注意を追加し、以下のセクションで Lambda のドキュメントへのリンクを追加しました。 • エラー名 • ステップ 3: Catch フィールドを使用するステートマシンを作成する **Note** Lambda での未処理のエラーは、エラー出力で `Lambda.Unknown` として報告されます。これには、 out-of-memory エラーや関数のタイムアウトが含まれます。`Lambda.Unknown` 、`States.ALL` 、または `States.TaskFailed` を一致させて、こういったエラーに処理できます。Lambda が最大呼び出し数に達すると、エラーは `Lambda.TooManyRequestsException` となります。Lambda 関数のエラーの詳細については、「AWS Lambda デベロッパーガイド」の「エラー処理と自動再試行」を参照してください。	2017 年 10 月 17 日
更新	IAM の手順を訂正して明確にし、すべてのチュートリアルのスクリーンショットを更新しました。	2017 年 10 月 11 日

変更	説明	変更日

更新	• Step Functions コンソールの変更を反映するために、ステートマシンの実行結果の新しいスクリーンショットを追加しました。Lambda コンソールの変更点を反映するために、以下のチュートリアルの Lambda の手順を書き換えました。
	  • Lambda を使用する Step Functions ステートマシン状態の作成
	  • ジョブステータスポーリングを作成する
	  • タスクタイマーの作成
	  • Step Functions ステートマシンを使用してエラー条件を処理する
	• 以下のセクションで、ステートマシンの作成について情報を訂正して明確にしました。
	  • Step Functions を使用してアクティビティステートマシンを作成する
		2017 年 10 月 6 日

更新	IAM コンソールの変更点を反映するため、以下のセクションの IAM の手順を書き換えました。 • ステートマシンの IAM ロールの作成 • Lambda を使用する Step Functions ステートマシン状態の作成 • ジョブステータスポーリングを作成する • タスクタイマーの作成 • Step Functions ステートマシンを使用してエラー条件を処理する • API Gateway を使用した Step Functions API の作成	2017 年 10 月 5 日
更新	ステートマシンデータ セクションを書き換えました。	2017 年 9 月 28 日
新機能	API アクションのスロットリングに関する制限が、Step Functions が利用可能なすべてのリージョンで引き上げられました。	2017 年 9 月 18 日

<table>
<tr><th>変更</th><th>説明</th><th>変更日</th></tr>
<tr><td>更新</td><td>• すべてのチュートリアルで新しい実行の開始に関する情報を訂正し明確にしました。
• アカウントに関連するクォータ セクションの情報を訂正し明確にしました。</td><td>2017 年 9 月 14 日</td></tr>
<tr><td>更新</td><td>Lambda コンソールの変更を反映して、以下のチュートリアルを書き換えました。
• Lambda を使用する Step Functions ステートマシン状態の作成
• Step Functions ステートマシンを使用してエラー条件を処理する
• ジョブステータスポーリングを作成する</td><td>2017 年 8 月 28 日</td></tr>
<tr><td>新機能</td><td>Step Functions は欧州 (ロンドン) で利用可能になりました。</td><td>2017 年 8 月 23 日</td></tr>
<tr><td>新機能</td><td>ステートマシンのビジュアルワークフローでは、グラフのズームイン、ズームアウト、および中央揃えができます。</td><td>2017 年 8 月 21 日</td></tr>
<tr><td>新機能</td><td>⚠ Important
実行では、その名前を 90 日間、別の実行で使用できなくなりました。

複数の <code>StartExecution</code> の呼び出しを同じ名前で行うと、新しい実行は開始されず、以下のルールが適用されます。
<table><tr><th>入力の型</th><th colspan="2">実行状態</th></tr><tr><td></td><td>を開きます。</td><td>[Closed] (クローズ)</td></tr></table></td><td>2017 年 8 月 18 日</td></tr>
</table>

変更	説明			変更日
	Identical	Success	ExecutionAlreadyExists	
	異なる	ExecutionAlreadyExists	ExecutionAlreadyExists	
	詳細については、AWS Step Functions API リファレンス の StartExecution API アクションの name リクエストパラメータを参照してください。			
更新	ステートマシンの ARN を API Gateway を使用した Step Functions API の作成 チュートリアルに渡す代替手段に関する情報を追加しました。			2017 年 8 月 17 日
更新	新しいジョブステータスポーリングの作成チュートリアルを追加しました。			2017 年 8 月 10 日
新機能	• Step Functions ExecutionThrottled CloudWatch はメトリクスを出力します。詳細については、「を使用した Step Functions のモニタリング CloudWatch」を参照してください。 • 状態のスロットリングに関連するクォータ セクションを追加しました。			2017 年 8 月 3 日
更新	ステップ 1: API Gateway 用にIAM ロールを作成する セクションの手順が更新されました。			2017 年 7 月 18 日
更新	選択 セクションの情報を訂正し明確にしました。			2017 年 6 月 23 日

変更	説明	変更日
更新	AWS 他のアカウントでのリソースの使用に関する情報が以下のチュートリアルに追加されました。 • Lambda を使用する Step Functions ステートマシン状態の作成 • AWS CloudFormationを使用して Step Functions 用の Lambda ステートマシンを作成する • Step Functions を使用してアクティビティステートマシンを作成する • Step Functions ステートマシンを使用してエラー条件を処理する	2017 年 6 月 22 日
更新	以下のセクションの情報を訂正し明確にしました。 • Step Functions ステートマシンを使用してエラー条件を処理する • 状態 • Step Functions のエラー処理	2017 年 6 月 21 日
更新	Step Functions コンソールの更新に合わせて、すべてのチュートリアルが書き直されました。	2017 年 6 月 12 日
新機能	Step Functions は、アジアパシフィック (シドニー) で利用可能になりました。	2017 年 6 月 8 日
更新	Amazon ステートメント言語 セクションを再構成しました。	2017 年 6 月 7 日
更新	Step Functions を使用してアクティビティステートマシンを作成する セクションの情報を訂正し明確にしました。	2017 年 6 月 6 日
更新	Retry と Catch を使用するステートマシンの例 のコード例を訂正しました。	2017 年 6 月 5 日
更新	AWS ドキュメント標準を使用してこのガイドを再構成しました。	2017 年 5 月 31 日

変更	説明	変更日
更新	並行 セクションの情報を訂正し明確にしました。	2017 年 5 月 25 日
更新	パスセクションとフィルタセクションを Step Functions の入出力処理 セクションに統合しました。	2017 年 5 月 24 日
更新	を使用した Step Functions のモニタリング CloudWatch セクションの情報を訂正し明確にしました。	2017 年 5 月 15 日
更新	`GreeterActivities.java` チュートリアルの Step Functions を使用してアクティビティステートマシンを作成する ワーカーコードを更新しました。	2017 年 5 月 9 日
更新	とは何ですか AWS Step Functions? セクションに入門動画を追加しました。	2017 年 4 月 19 日
更新	以下のチュートリアルの情報を訂正し明確にしました。 • Lambda を使用する Step Functions ステートマシン状態の作成 • Step Functions を使用してアクティビティステートマシンを作成する • Step Functions ステートマシンを使用してエラー条件を処理する	2017 年 4 月 19 日
更新	Lambda を使用する Step Functions ステートマシン状態の作成 および Step Functions ステートマシンを使用してエラー条件を処理する チュートリアルに Lambda テンプレートに関する情報を追加しました。	2017 年 4 月 6 日
更新	「Maximum input or result data size」の制限を「Maximum input or result data size for a task, state, or execution」に変更しました (32,768 文字)。詳細については、「タスクの実行に関連するクォータ」を参照してください。	2017 年 3 月 31 日

変更	説明	変更日
新機能	• Step Functions は、Step Functions を Amazon CloudWatch イベントターゲットとして設定することで、ステートマシンの実行をサポートします。	2017 年 3 月 21 日
新機能	• Step Functions で、優先エラー処理メソッドとして Lambda 関数エラー処理を使用できます。 • Step Functions ステートマシンを使用してエラー条件を処理する チュートリアルおよび Step Functions のエラー処理 セクションを更新しました。	2017 年 3 月 16 日
新機能	Step Functions は欧州 (フランクフルト) で利用可能になりました。	2017 年 3 月 7 日
更新	目次のトピックを再編成し、次のチュートリアルを更新しました。 • Lambda を使用する Step Functions ステートマシン状態の作成 • Step Functions を使用してアクティビティステートマシンを作成する • Step Functions ステートマシンを使用してエラー条件を処理する	2017 年 2 月 23 日
新機能	• Step Functions コンソールの [State Machines] ページには、[Copy to New] (新しいものにコピー) ボタンおよび [Delete] (削除) ボタンが含まれます。 • コンソールでの変更点に合わせてスクリーンショットを更新しました。	2017 年 2 月 23 日
新機能	• Step Functions は API Gateway を使用した API の作成をサポートしています。 • API Gateway を使用した Step Functions API の作成 チュートリアルを追加しました。	2017 年 2 月 14 日
新機能	• Step Functions はとの統合をサポートしています AWS CloudFormation。 • AWS CloudFormationを使用して Step Functions 用の Lambda ステートマシンを作成する チュートリアルを追加しました。	2017 年 2 月 10 日

変更	説明	変更日
更新	`Parallel` 状態に関連して `ResultPath` フィールドおよび `OutputPath` フィールドの現在の動作を明確にしました。	2017 年 2 月 6 日
更新	• チュートリアルにおけるステートマシンの命名の制約を明確にしました。 • コード例をいくつか修正しました。	2017 年 1 月 5 日
更新	最新のプログラミングモデルを使用するように Lambda 関数の例を更新しました。	2016 年 12 月 9 日
新機能	Step Functions の初回リリース。	2016 年 12 月 1 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。