aws

开发人员指南

AWS Step Functions

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

AWS Step Functions: 开发人员指南

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

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS Step Functions？ 1
AWS SDK 和优化的集成 1
标准和快速工作流 2
标准工作流规范 2
快速工作流规范 2
使用案例 3
用例 #1：函数编排 3
用例 #2：分支 3
用例 #3：错误处理 4
用例 #4：人机回路 5
用例 #5：并行处理 5
用例 #6：动态并行 6
服务集成 6
支持的 区域 9
第一次使用 Step Functions？ 10
开始使用 11
重要概念 11
本系列教程 13
先决条件 16
注册获取 AWS 账户 16
创建具有管理访问权限的用户 16
教程 1：为状态机创建原型 18
后续步骤 19
教程 2：使用 Lambda 函数定义第一个服务集成 19
第 1 步：创建并测试 Lambda 函数 19
第 2 步：更新工作流 – 配置“获取信用额度”状态 20
后续步骤 21
教程 3：在工作流中实现 if-else 条件 21
第 1 步：创建接收回调令牌的 Amazon SNS 主题 22
第 2 步：创建 Lambda 函数来处理回调 22
第 3 步：更新工作流 – 在“选择”状态下添加 if-else 条件逻辑 24
后续步骤 26
教程 4：定义并行执行的多项任务 26
第 1 步：创建 Lambda 函数执行所需的检查 26

第 2 步：更新工作流 – 添加要执行的并行任务 28
教程 5：同时迭代一组项目 30
第 1 步：创建 DynamoDB 表来存储所有征信机构的名称 30
第 2 步：更新状态机 – 从 DynamoDB 表中获取结果 31
第 3 步：创建可返回所有征信机构信用评分的 Lambda 函数 31
第 4 步：更新状态机 – 添加 Map 状态以迭代方式获取信用评分 32
教程 6：保存工作流并执行状态机 32
第 1 步：保存状态机 33
第 2 步：添加剩余的 IAM 策略 34
第 3 步：运行状态机 34
教程 7：配置输入和输出 35
使用 InputPath 过滤器选择原始输入的特定部分 37
使用 Parameters 筛选条件处理选定的输入 40
使用 ResultSelector、ResultPath 和 OutputPath 筛选条件配置输出 41
教程 8：在控制台中调试错误 43
调试无效路径 Choice 状态错误 44
在应用输入和输出筛选条件时调试 JSON 路径表达式错误 46
使用案例 48
数据处理 48
机器学习 49
微服务编排 50
IT 和安全自动化 51
Step Functions 的工作原理 53
标准和快速工作流 53
同步和异步快速工作流 56
执行保障 57
使用快速工作流实现成本优化 58
状态 60
Amazon States Language 61
Pass 80
任务 82
Choice 101
Wait 108
Succeed 110
Fail 110
Parallel 112

Map 116
Map 状态处理模式 117
内联模式和分布式模式的区别 117
在内联模式下使用 Map 状态 119
使用分布式模式下的 Map 状态 127
分布式 Map 状态容许的故障阈值 137
转换 139
分布式 Map 状态下的转换 140
状态机数据 140
数据格式 141
状态机输入/输出 141
状态输入/输出 142
输入和输出处理 143
路径 145
InputPath、参数和 ResultSelector 147
ResultPath 152
OutputPath 161
InputPath、ResultPath 和 OutputPath 示例 162
Map 状态输入和输出字段 166
Context 对象 196
数据流模拟器 202
使用数据流模拟器 203
数据流模拟器注意事项 204
版本与别名功能 205
版本 206
别名 209
版本与别名功能的授权 212
将执行与版本或别名关联 214
部署示例 217
版本的逐步部署 220
执行 228
从任务开始执行 229
使用 EventBridge 调度器 231
标准和快速工作流执行 236
查看和调试执行 240
Redriving执行 259

检查 Map Run 268
错误处理 279
错误名称 279
出错后重试 282
回退状态 286
使用 Retry 和使用 Catch 的状态机示例 288
调用 Step Functions 292
读取一致性 293
Step Functions 中的标记 293
成本分配的标记 294
标记以提高安全性 294
查看和管理 295
标记 API 296
Workflow Studio 297
界面概述 298
设计模式 298
代码模式 304
配置模式 307
键盘快捷键 310
使用 Workflow Studio 311
创建工作流 311
设计工作流 313
运行工作流 319
编辑工作流 320
导出工作流 322
创建工作流原型 323
配置输入和输出 324
配置状态的输入 325
配置状态的输出 327
Workflow Studio 中的执行角色 333
关于自动生成的角色 334
自动生成角色 334
解决角色生成问题 336
在 Workflow Studio 中测试 HTTP 任务的角色 336
在 Workflow Studio 中测试优化的服务集成的角色 337
在工作流工作室中测试 AWS SDK 服务集成的角色 337

在 Workflow Studio 中测试流状态的角色 338
错误处理 338
出错时重试 339
捕获错误 340
超时 340
HeartbeatSeconds 340
教程：学习使用 AWS Step Functions Workflow Studio 340
第 1 步：导航到 Workflow Studio 341
第 2 步：创建状态机 341
第 3 步：查看自动生成的 Amazon States Language 定义 343
第 4 步：在“代码”模式下编辑工作流定义 345
第 5 步：保存状态机 347
第 6 步：运行状态机 347
第 7 步：更新状态机 349
第 8 步：清除 350
教程 351
创建使用 Lambda 的 Step Functions 状态机 351
第 1 步：创建 Lambda 函数 352
第 2 步：测试 Lambda 函数 353
第 3 步：创建状态机 353
第 4 步：运行状态机 355
使用状态机处理错误情形 357
第 1 步：创建一个失败的 Lambda 函数 357
第 2 步：测试 Lambda 函数 358
第 3 步：创建使用 Catch 字段的状态机 359
第 4 步：运行状态机 361
使用内联 Map 状态重复操作 362
第 1 步：创建工作流原型 363
第 2 步：配置输入和输出 363
第 3 步：查看自动生成的 Amazon States Language 定义并保存工作流 364
第 4 步：运行状态机 366
开始使用分布式 Map 状态 367
先决条件 368
第 1 步：创建工作流原型 368
第 2 步：配置 Map 状态的必填字段 368
第 3 步：配置其他选项 370

第 4 步：配置 Lambda 函数 370
第 5 步：更新工作流原型 371
第 6 步：查看自动生成的 Amazon States Language 定义并保存工作流 371
第 7 步：运行状态机 374
使用 Lambda 函数处理整批数据 374
第 1 步：创建状态机 375
第 2 步：创建 Lambda 函数 377
第 3 步：运行状态机 378
使用 Lambda 函数处理单个数据项 379
第 1 步：创建状态机 380
第 2 步：创建 Lambda 函数 382
第 3 步：运行状态机 378
启动状态机执行以响应 Amazon S3 事件 386
先决条件：创建状态机 387
第 1 步：在 Amazon S3 中创建一个存储桶 387
第 2 步：使用 EventBridge 启用 Amazon S3 事件通知 387
第 3 步：创建 Amazon EventBridge 规则 388
第 4 步：测试规则 389
执行输入的示例 390
使用 API Gateway 创建 Step Functions API 390
第 1 步：为 API Gateway 创建 IAM 角色 391
第 2 步：创建 API Gateway API 392
第 3 步：测试和部署 API Gateway API 395
使用 AWS SAM 创建 Step Functions 状态机 397
先决条件 398
第 1 步：下载示例 AWS SAM 应用程序 399
第 2 步：构建应用程序 400
第 3 步：将应用程序部署到 AWS 云 401
故障排除 402
清除 402
创建活动状态机 403
第 1 步：创建活动 403
第 2 步：创建状态机 404
第 3 步：实现工作线程 406
第 4 步：运行状态机 408
第 5 步：运行和停止工作线程 409

使用 Lambda 迭代循环 409
第 1 步：创建迭代计数的 Lambda 函数 410
第 2 步：测试 Lambda 函数 411
第 3 步：创建状态机 412
第 4 步：启动新的执行 414
将进行中的工作作为新执行继续执行 415
使用 Step Functions API 操作（推荐） 416
使用 Lambda 函数 419
部署示例人员批准项目 430
第 1 步：创建一个模板 431
第 2 步：创建一个堆栈 431
第 3 步：批准 SNS 订阅 432
第 4 步：运行状态机 432
模板源代码 435
在 Step Functions 中查看 X-Ray 跟踪 445
第 1 步：为 Lambda 创建 IAM 角色。 445
第 2 步：创建 Lambda 函数 446
第 3 步：再创建两个 Lambda 函数 447
第 4 步：创建状态机 448
第 5 步：运行状态机 450
使用 AWS 软件开发工具包服务集成收集 Amazon S3 存储桶信息 453
第 1 步：创建状态机 453
第 2 步：添加必要的 IAM 角色权限 455
第 3 步：启动一个标准状态机执行 456
第 4 步：运行一个快速状态机执行 457
开发人员工具 458
开发选项 458
Step Functions 控制台 459
AWS 软件开发工具包 459
标准和快速工作流 460
HTTPS 服务 API 460
开发环境 460
端点 460
AWS CLI 461
Step Functions Local 461
AWS Toolkit for Visual Studio Code 461

AWS Serverless Application Model 和 Step Functions 461
Terraform 和 Step Functions 462
定义格式支持 462
Step Functions 和 AWS SAM 469
为什么要将 Step Functions 配合使用 AWS SAM？ 469
Step Functions 与 AWS SAM 规范集成 470
Step Functions 与 SAM CLI 集成 470
DefinitionSubstitutions 在AWS SAM模板中 471
后续步骤 475
使用 Application Composer 中的 Workflow Studio 475
使用 Application Composer 中的 Workflow Studio 476
使用 CloudFormation 定义替换项动态引用资源 476
将服务集成任务连接到增强型组件卡 476
导入现有项目并在本地同步它们 477
AWS 应用程序编辑器 中不可用的 Workflow Studio 功能 477
使用创建 Lambda 状态机 AWS CloudFormation 478
第 1 步：设置您的 AWS CloudFormation 模板 478
步骤 2：使用 AWS CloudFormation 模板创建 Lambda 状态机 484
第 3 步：启动状态机执行 489
使用 AWS CDK 创建一个 Lambda 状态机 490
第 1 步：设置您的 AWS CDK 项目 490
第 2 步：使用 AWS CDK 创建状态机 492
第 3 步：启动状态机执行 501
第 4 步：清除 501
后续步骤 501
使用同步 Express 状态机创建 API Gateway REST API 使用 AWS CDK 502
第 1 步：设置您的 AWS CDK 项目 503
第 2 步：使用创建具有同步 Express 状态机后端集成的 API Gateway REST API AWS CDK 506
第 3 步：测试 API Gateway Gateway 516
第 4 步：清除 519
数据科学开发工具包 519
使用 Terraform 部署状态机 520
先决条件 520
使用 Terraform 的开发生命周期 521
状态机的 IAM 角色和策略 523

测试和调试 525
使用 TestState API 525
使用 TestState API 的注意事项 526
在 TestState API 中使用检查级别 526
IAM使用 TestState API 的权限 533
测试状态（控制台） 533
使用 AWS CLI 测试状态 534
测试和调试输入和输出数据流 539
在本地测试状态机 543
设置 Step Functions Local（可下载版本）和 Docker 544
设置 Step Functions Local（可下载版本）- Java 版本 544
为 Step Functions Local 设置配置选项 546
在计算机上运行 Step Functions Local 548
测试 Step Function 和 AWS SAM CLI Local 550
使用模拟服务集成 554
最佳实操 571
使用超时避免执行卡顿 571
使用 Amazon S3 ARN 而不是传递大量有效负载 572
避免达到历史记录的配额 574
处理 Lambda 服务异常 575
避免轮询活动任务时发生延迟 576
选择标准或快速工作流 576
Amazon CloudWatch Logs 资源策略大小限制 577
使用其他服务 578
致电其他 AWS 服务 578
优化集成 579
AWS 软件开发工具包集成 579
集成模式支持 579
跨账户存取 582
AWS SDK 服务集成 582
使用 AWS SDK 服务集成 583
支持的 服务 584
支持服务的不支持 API 操作 624
已弃用的 AWS SDK 服务集成 626
优化集成 627
Amazon API Gateway 630

Amazon Athena 637
AWS Batch 640
Amazon Bedrock 642
AWS CodeBuild 645
Amazon DynamoDB 650
Amazon ECS/Fargate 654
Amazon EKS 657
Amazon EMR 670
Amazon EMR on EKS 681
Amazon EMR Serverless 684
亚马逊 EventBridge 691
AWS Glue 694
AWS Glue DataBrew 695
AWS Lambda 696
AWS Elemental MediaConvert 699
Amazon SageMaker 702
Amazon SNS 713
Amazon SQS 715
AWS Step Functions 718
调用第三方 API 721
HTTP 任务定义 722
HTTP 任务字段 722
HTTP 任务的身份验证 728
合并 EventBridge 连接和 HTTP 任务定义数据 729
在请求正文上应用 URL 编码 732
运行 HTTP 任务的 IAM 权限 733
HTTP 任务示例 734
测试 HTTP 任务 737
不支持的 HTTP 任务响应 738
服务集成模式 739
请求响应 739
运行作业 (.sync) 740
等待具有任务令牌的回调 741
将参数传递给服务 API 746
将静态 JSON 作为参数传递 747
使用路径将状态输入作为参数传递 747

将上下文对象节点作为参数传递 748
更改集成日志 748
Step Functions 的示例项目 772
管理批处理作业（AWS Batch、Amazon SNS） 773
第 1 步：创建状态机并预置资源 773
第 2 步：运行状态机 775
示例状态机代码 776
IAM 示例 777
管理容器任务（Amazon ECS、Amazon SNS） 778
第 1 步：创建状态机并预置资源 779
第 2 步：运行状态机 780
示例状态机代码 781
IAM 示例 783
传输数据记录（Lambda、DynamoDB、Amazon SQS） 784
第 1 步：创建状态机并预置资源 784
第 2 步：运行状态机 786
示例状态机代码 787
IAM 示例 789
任务状态投票 (Lambda,) AWS Batch 790
第 1 步：创建状态机并预置资源 790
第 2 步：运行状态机 793
示例状态机代码 795
任务计时器（Lambda、Amazon SNS） 797
第 1 步：创建状态机并预置资源 797
第 2 步：运行状态机 799
回调模式示例（Amazon SQS、Amazon SNS、Lambda） 801
第 1 步：创建状态机并预置资源 801
第 2 步：运行状态机 803
Lambda 回调示例 804
管理 Amazon EMR 任务 805
第 1 步：创建状态机并预置资源 806
第 2 步：运行状态机 780
示例状态机代码 781
IAM 示例 783
运行 EMR Serverless 作业 814
AWS CloudFormation 模板和其他资源 814

第 1 步：创建状态机并预置资源 814
第 2 步：运行状态机 816
在工作流内启动工作流（Step Functions、Lambda） 817
第 1 步：创建状态机并预置资源 817
第 2 步：运行状态机 819
示例状态机代码 820
使用 Map 状态动态处理数据 822
第 1 步：创建状态机并预置资源 822
第 2 步：订阅 Amazon SNS 主题 825
第 3 步：向 Amazon SQS 队列添加消息 826
第 4 步：运行状态机 826
示例状态机代码 827
IAM 示例 830
使用分布式 Map 处理 CSV 文件 831
AWS CloudFormation 模板和其他资源 831
第 1 步：创建状态机并预置资源 832
第 2 步：运行状态机 834
使用分布式 Map 处理 Amazon S3 存储桶中的数据 835
AWS CloudFormation 模板和其他资源 836
第 1 步：创建状态机并预置资源 837
第 2 步：运行状态机 839
训练机器学习模型 840
第 1 步：创建状态机并预置资源 841
第 2 步：运行状态机 843
示例状态机代码 844
IAM 示例 846
调整机器学习模型 847
第 1 步：创建状态机并预置资源 848
第 2 步：运行状态机 850
示例状态机代码 851
IAM 示例 855
处理来自 Amazon SQS（快速工作流）的大批量消息 858
第 1 步：创建状态机并预置资源 859
第 2 步：触发状态机执行 861
示例 Lambda 函数代码 862
示例状态机代码 862

IAM 示例 864
选择性检查点示例（快速工作流） 865
第 1 步：创建状态机并预置资源 865
第 2 步：运行状态机 868
父级（标准工作流）的示例状态机代码 868
父状态机的示例 IAM 角色 871
嵌套状态机（快速工作流）的示例状态机代码 868
子状态的示例 IAM 角色 875
构建 AWS CodeBuild 项目（CodeBuild，亚马逊 SNS） 876
第 1 步：创建状态机并预置资源 876
第 2 步：运行状态机 878
示例状态机代码 879
预处理数据并训练机器学习模型 881
第 1 步：创建状态机并预置资源 881
第 2 步：运行状态机 883
示例状态机代码 884
IAM 示例 887
Lambda 编排示例 888
第 1 步：创建状态机并预置资源 889
第 2 步：运行状态机 891
关于状态机及其执行 892
IAM 示例 895
启动 Athena 查询 898
第 1 步：创建状态机并预置资源 898
第 2 步：运行状态机 900
示例状态机代码 901
IAM 示例 903
执行多个查询（Amazon Athena、Amazon SNS） 905
第 1 步：创建状态机并预置资源 905
第 2 步：运行状态机 907
示例状态机代码 908
IAM 示例 911
查询大型数据集（亚马逊 Athena、亚马逊 S3、亚马逊 SN AWS Glue S） 914
第 1 步：创建状态机并预置资源 915
第 2 步：运行状态机 917
示例状态机代码 918

IAM 示例 920
保持数据最新（亚马逊 Athena、Amazon S3、）AWS Glue 923
第 1 步：创建状态机并预置资源 923
第 2 步：运行状态机 925
示例状态机代码 926
IAM 示例 927
管理 Amazon EKS 集群 929
第 1 步：创建状态机并预置资源 930
第 2 步：运行状态机 932
示例状态机代码 933
IAM 示例 938
调用 API Gateway 939
第 1 步：创建状态机并预置资源 939
第 2 步：运行状态机 941
示例状态机代码 942
IAM 示例 944
使用 API Gateway 调用微服务 944
第 1 步：创建状态机并预置资源 945
第 2 步：运行状态机 947
示例状态机代码 948
IAM 示例 949
将自定义事件发送到 EventBridge 951
第 1 步：创建状态机并预置资源 951
第 2 步：运行状态机 953
示例状态机代码 954
IAM 示例 955
调用同步快速工作流 956
第 1 步：创建状态机并预置资源 956
第 2 步：运行状态机 958
示例状态机代码 959
IAM 示例 961
使用 Amazon Redshift 运行 ETL/ELT 工作流程 962
第 1 步：创建状态机并预置资源 963
第 2 步：运行状态机 965
示例状态机代码 966
IAM 示例 987

使用 Step Functions 和 AWS Batch 进行错误处理 987
第 1 步：创建状态机并预置资源 988
第 2 步：运行状态机 989
示例状态机代码 990
IAM 示例 992
分散一份 AWS Batch 工作 993
第 1 步：创建状态机并预置资源 993
第 2 步：运行状态机 995
示例状态机代码 996
IAM 示例 997
AWS Batch 用 Lambda 999
第 1 步：创建状态机并预置资源 999
第 2 步：运行状态机 1001
示例状态机代码 1002
IAM 示例 1003
使用 Amazon Bedrock 执行 AI 提示链接 1004
AWS CloudFormation 模板和其他资源 1005
先决条件 1005
第 1 步：创建状态机并预置资源 1005
第 2 步：运行状态机 1007
配额 1009
常规配额 1010
与账户相关的配额 1010
与 HTTP 任务相关的配额 1011
与状态限制相关的配额 1012
与 API 操作限制相关的配额 1013
与 TestState API 相关的配额 1013
其他配额 1013
与状态机执行相关的配额 1016
与任务执行相关的配额 1017
与版本与别名功能相关的配额 1018
与标记相关的限制 1018
日记账记录和监控 1020
亚马逊 CloudWatch 指标 1020
报告时间间隔的指标 1021
报告计数的指标 1021

执行指标 1022
版本与别名功能的资源计数指标 1025
活动指标 1025
Lambda 函数指标 1026
服务集成指标 1027
服务指标 1028
API 指标 1029
尽力交付 CloudWatch 指标 1030
查看 Step Functions 的指标 1030
为 Step Functions 设置警报 1032
亚马逊 EventBridge 活动 1034
EventBridge 有效载荷 1035
Step Functions 事件示例 1036
将 Step Functions 事件路由到 EventBridge 1040
用录音 CloudTrail 1041
中的数据事件 CloudTrail 1042
中的管理活动 CloudTrail 1043
事件示例 1045
使用 CloudWatch Logs 进行日志记录 1047
配置日志记录 1047
CloudWatch Logs 有效负载 1048
用于记录到 CloudWatch Logs 的 IAM 策略 1048
日志级别 1049
X-Ray 1053
设置和配置 1055
概念 1058
服务集成 1058
查看 X-Ray 控制台 1059
查看 Step Functions 的 X-Ray 跟踪信息 1060
跟踪 1060
服务图 1061
分段和子分段 1062
分析 1064
配置 1065
如果追踪视图或服务图中没有数据怎么办？ 1066
配合使用 AWS 用户通知服务和 Step Functions 1067

安全性 1068
数据保护 1068
加密 1069
Identity and Access Management 1069
受众 1069
使用身份进行身份验证 1070
使用策略管理访问 1072
访问控制 1074
策略操作 1074
策略资源 1075
策略条件键 1076
ACL 1076
ABAC 1077
临时凭证 1077
主体权限 1078
服务角色 1078
服务相关角色 1078
如何 AWS Step Functions 与 IAM 配合使用 1079
基于身份的策略示例 1079
基于身份的策略 1082
基于资源的策略 1082
AWS 托管策略 1083
创建状态机 IAM 角色 1084
为非管理员用户创建精细的 IAM 权限 1087
访问跨账户资源 AWS 1090
VPC 端点 1099
集成服务的 IAM 策略 1102
使用分布式 Map 状态的 IAM 策略 1191
基于标签的策略 1196
故障排除 1197
日志记录和监控 1199
合规性验证 1199
弹性 1200
基础架构安全性 1200
配置和脆弱性分析 1200
从中迁移工作负载 AWS Data Pipeline 1201

迁移工作负载 1201
概念映射 1202
Step Functions 示例项目 1203
定价比较 1203
故障排除 1205
一般故障排除 1205
我无法创建状态机。 1205
我无法使用 JsonPath 来引用上一个任务的输出。 1205
状态转换出现延迟。 1206
启动新的标准工作流执行时，会执行失败并出现 ExecutionLimitExceeded 错误。 1206
一个处于并行状态的分支出现故障，导致整个执行失败。 1206
服务集成故障排除 1206
在下游服务中，我的作业已经完成，但在 Functions 中，任务状态仍为“进行中”或延迟完成。 1206
我想从嵌套状态机执行中返回 JSON 输出。 1207
我无法从另一个账户调用 Lambda 函数。 1207
我无法看到从 .waitForTaskToken 状态传递的任务令牌。 1208
活动故障排除 1209
我的状态机执行卡在活动状态。 1209
我的活动工作线程在等待任务令牌时超时。 1209
快速工作流故障排除 1209
我的应用程序在收到 StartSyncExecution API 调用的响应之前超时。 1209
我无法查看执行历史记录，无法解决快速工作流故障。 1209
相关信息 1211
最近发布的特征 1212
文档历史记录 1214
..... mccxlvi

什么是 AWS Step Functions？

AWS Step Functions 是一项可视化工作流服务，可帮助您构建分布式应用程序、实现流程自动化、协调微服务以及创建数据和机器学习 (ML) 管道。

在 Step Functions 的图形控制台中，您可以将应用程序的工作流程视为一系列事件驱动的步骤。

Step Functions 基于状态机和任务。在 Step Functions 中，状态机被称为工作流，这是一系列事件驱动的步骤。工作流程中的每个步骤都称为状态。例如，任务状态表示另一个 AWS 服务执行的工作单元，例如调用另一个服务 AWS 服务 或 API。

借助 Step Functions 的内置控件，您可以检查工作流程中每个步骤的状态，以确保应用程序按预期顺序运行。根据您的用例，您可以让 Step Functions 调用 AWS 服务（例如 Lambda）来执行任务。您可以创建处理和发布机器学习模型的工作流。您可以使用 Step Functions 控制 AWS 服务 AWS Glue，例如用于创建提取、转换和加载 (ETL) 工作流程。您还可以为需要人为交互的应用程序创建长时间运行的自动化工作流。

> **Tip**
>
> 要学习如何使用 Step Functions，请按照AWS Step Functions 研讨会中的交互式模块进行操作，或者阅读本指南中的 “入门” 部分，创建信用卡申请工作流程。

主题

- AWS SDK 和优化的集成
- 标准和快速工作流
- 使用案例
- 服务集成
- 支持的 区域
- 第一次使用 Step Functions？

AWS SDK 和优化的集成

要调用其他 AWS 服务，你可以使用 Step Functions 的 AWS SDK 集成，也可以使用 Step Functions 的优化集成之一。

- AWS SDK 集成允许您直接从状态机调用 200 多个 AWS 服务中的任何一个，从而可以访问九千多个 API 操作。
- Step Functions 的优化集成经过自定义，可简化在状态机中的使用。

标准和快速工作流

Step Functions 包含两种工作流类型。标准工作流包含仅一次工作流执行，并且最多可以运行一年。这表示标准工作流中的每个步骤将只执行一次。但是，Express at-least-once 工作流程具有工作流程执行功能，最多可以运行五分钟。这表示快速工作流中的一个或多个步骤可能会运行多次，而工作流中的每个步骤至少会执行一次。

执行是指运行工作流以执行任务的实例。标准工作流非常适合长时间运行、可审计的工作流程，因为这类工作流会显示执行历史记录和可视化调试。Express 工作流程非常适合 high-event-rate 工作负载，例如流数据处理和物联网数据摄取。

标准工作流规范

- 每秒 2000 的执行速率
- 每秒 4000 的状态转换速率
- 按状态转换定价
- 显示执行历史记录和可视化调试
- 支持所有服务集成和模式

快速工作流规范

- 每秒 100000 的执行速率
- 状态转换速率几乎不受限制
- 按执行次数和执行持续时间定价
- 向 Amazon 发送执行历史记录 CloudWatch
- 根据启用的日志级别显示执行历史记录和可视化调试
- 支持所有服务集成和大多数模式

有关标准和快速工作流的更多信息，包括 Step Functions 定价，请参阅以下内容：

- 标准和快速工作流
- AWS Step Functions 定价

使用案例

Step Functions 用于管理应用程序的组件和逻辑，因此可以减少编写的代码，从而专注于快速构建和更新应用程序。本节将介绍使用 Step Functions 的典型用例。

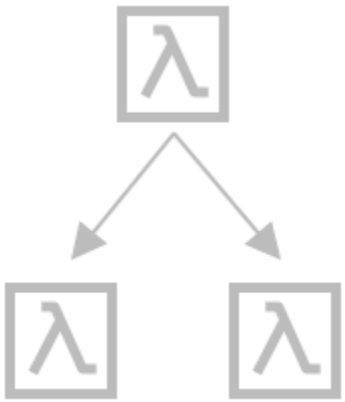
用例 #1：函数编排

您创建一个工作流，按照特定顺序运行一组 Lambda 函数（步骤）。一个 Lambda 函数的输出会传递作为下一个 Lambda 函数的输入。工作流的最后一个步骤将给出结果。使用 Step Functions，您可以看到工作流中的每个步骤是如何相互交互的，从而可以确保每个步骤都能按预期运行其函数。

有关说明如何使用一组函数创建状态机的教程，请参阅以下内容：

- 入门 AWS Step Functions

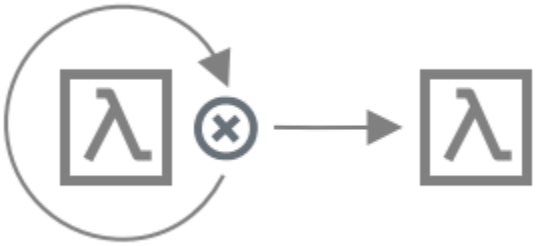
用例 #2：分支



一位客户申请提高信用额度。您可以使用 `Choice` 状态，让 Step Functions 根据 `Choice` 状态的输入做出决策。如果该申请超过了客户预先批准的信用额度，则可以让 Step Functions 将客户的申请发送

给经理进行签署。如果该申请低于客户预先批准的信用额度，则可以让 Step Functions 自动批准该申请。

用例 #3：错误处理



`Retry`

在此用例中，一名客户申请了一个用户名。客户的第一次申请失败了。您可以使用 `Retry` 语句，让 Step Functions 再次尝试客户的申请。客户的第二次申请成功了。

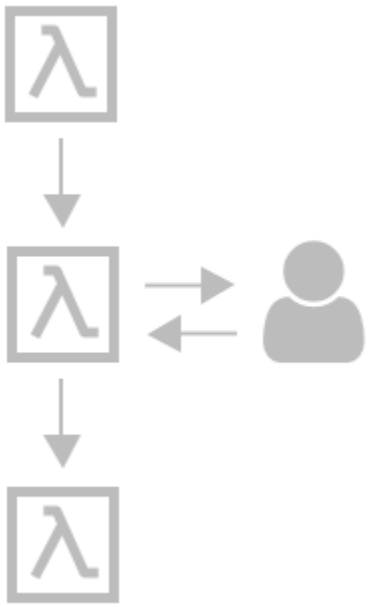
`Catch`

在类似的用例中，一名客户申请了一个不可用的用户名。您可以使用 `Catch` 语句，让 Step Functions 建议一个可用的用户名。如果客户使用了该用户名，则可以让 Step Functions 进入工作流的下一个步骤，即发送确认电子邮件。如果客户没有使用该用户名，则可以让 Step Functions 进入工作流的另一个步骤，即重新开始注册流程。

有关 `Retry` 和 `Catch` 语句的更多详细示例，请参阅以下内容：

- Step Functions 中的错误处理

用例 #4：人机回路

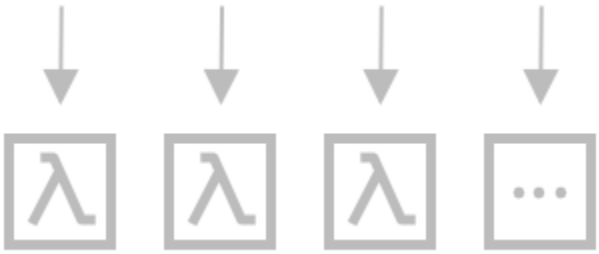


您的一位客户使用银行应用程序向朋友汇款。客户正在等待确认电子邮件。通过回调和任务令牌，Step Functions 可以让 Lambda 汇出客户的款项，并在客户的朋友收到款项时向客户报告。在 Lambda 报告客户的朋友收到款项后，您可以让 Step Functions 进入工作流的下一个步骤，即向客户发送一封确认电子邮件。

要查看显示使用任务令牌发出回调的示例项目，请参阅以下内容：

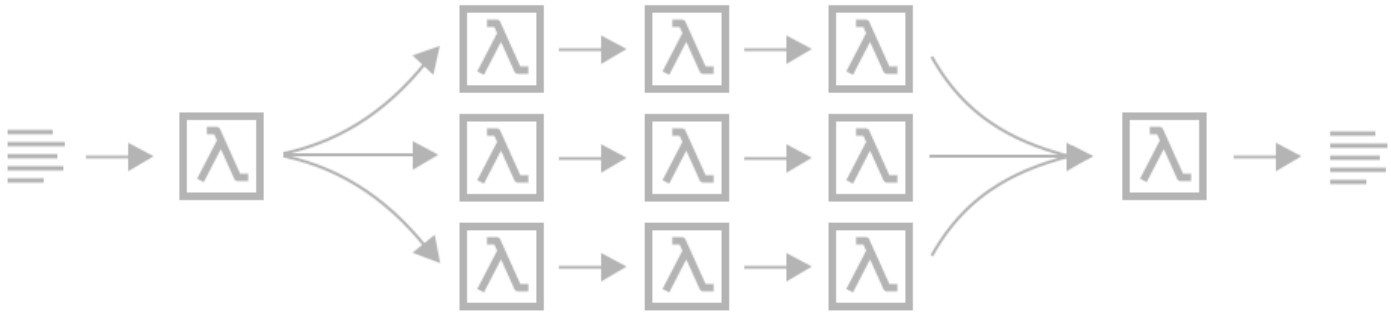
- 回调模式示例（Amazon SQS、Amazon SNS、Lambda）

用例 #5：并行处理



客户需要将视频文件转换为五种不同的显示分辨率，以便观看者可以在多台设备上观看视频。Step Functions 使用 `Parallel` 状态输入视频文件，以便 Lambda 可以同时将其处理为五种显示分辨率。

用例 #6：动态并行



客户订购了三件物品，您需要为每件物品做好配送准备。您需要检查每件物品的库存情况，收集物品，然后将每件物品打包以便配送。使用 Map 状态，Step Functions 让 Lambda 并行处理客户的每件物品。将客户的所有物品打包好送货后，Step Functions 将进入工作流的下一个步骤，即向客户发送一封包含追踪信息的确认电子邮件。

要查看使用 Map 状态展示动态并行的示例项目，请参阅以下内容：

- 使用 Map 状态动态处理数据

服务集成

Step Functions 与多种 AWS 服务集成。要将 Step Functions 与这些服务结合使用，请使用以下服务集成模式：

请求响应（默认）

- 调用服务，并让 Step Functions 在获得 HTTP 响应后进入下一个状态。

运行作业 (.sync)

- 调用服务，并让 Step Functions 等待作业完成。

等待带有任务令牌的回调 (. waitForTask代币)

- 使用任务令牌调用服务，并让 Step Functions 等到任务令牌带着回调返回。

下表显示了 Step Functions 的可用服务集成和服务集成模式。

标准工作流程和快速工作流程支持相同的集成，但不支持相同的集成模式。

- 每种集成的优化集成模式支持都不同。
- Express Workflows 不支持 Run a Job (.sync) 或等待回调 (. waitForTask代币 ）。
- 有关更多信息，请参阅 标准和快速工作流。

Standard Workflows

支持的服务集成

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
优化集成	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	✓	✓	

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS 软件开发工具包集成	两百多种	✓		✓

Express Workflows

支持的服务集成

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
优化集成	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS 软件开发工具包集成	两百多种	✓		

支持的 区域

大多数 AWS 区域都支持 Step Functions。有关提供 Step Fun AWS ctions 的区域的完整列表，请参阅AWS 区域表。

第一次使用 Step Functions？

如果这是您第一次使用 Step Functions，以下主题可帮助您了解使用 Step Functions 的不同部分，包括 Step Functions 如何与其他 AWS 服务相结合：

- Step Functions 教程
- Step Functions 的示例项目
- AWS Step Functions 适用于 Python 的数据科学软件开发工具包

入门 AWS Step Functions

Step Functions 是一项无服务器编排服务，您可以用其将应用程序工作流定义为一系列事件驱动的步骤。工作流中的每个步骤都称为状态。状态（例如任务状态、Choice、Parallel 和 Map）最常用于定义工作流。在Task各州内，你可以使用 Step Functions 支持的 AWS SDK 集成，并在工作流程 AWS 服务 中协调多个集成。

主题

- 重要概念
- 本系列教程
- 入门的先决条件 AWS Step Functions
- 教程 1：为状态机创建原型
- 教程 2：使用 Lambda 函数定义第一个服务集成
- 教程 3：在工作流中实现 if-else 条件
- 教程 4：定义并行执行的多项任务
- 教程 5：同时迭代一组项目
- 教程 6：保存工作流并执行状态机
- 教程 7：配置输入和输出
- 教程 8：在控制台中调试错误

重要概念

在开始教程之前，请查看以下主要的 Step Functions 术语以了解上下文。

租期	描述
工作流	通常反映业务流程的一系列步骤。
状态	状态机中的各个步骤可以根据其输入做出决策，根据这些输入执行操作，并将输出传递给其他状态。 有关更多信息，请参阅 状态。

租期	描述
Workflow Studio	一款可视化工作流设计器，可帮助您更快地进行工作流的原型设计和构建。 有关更多信息，请参阅 AWS Step Functions 工作流程工作室。
状态机	使用 JSON 文本定义的工作流，该文本表示工作流中的各个状态或步骤以及字段，例如 `StartAt`、`TimeoutSeconds` 和 `Version`。 有关更多信息，请参阅 状态机结构。
Amazon States Language	一种基于 JSON 的结构化语言，用于定义状态机。使用 ASL，您可以定义一组可以起作用的`Task`状态（状态），确定要过渡到下一个`Choice`状态（状态），并通过错误停止执行（`Fail`状态）。 有关更多信息，请参阅 Amazon States Language。
输入与输出配置	工作流程中的状态接收 JSON 数据作为输入，通常会将 JSON 数据作为输出传递到下一个状态。Step Functions 提供过滤器来控制各州之间的数据流。 有关更多信息，请参阅 Step Functions 中的输入和输出处理。
服务集成	您可以从工作流程中调用 AWS 服务 API 操作。 有关更多信息，请参阅 与其他服务 AWS Step Functions 一起使用。
服务集成类型	• AWS SDK 集成 — 直接从状态机调用二百 AWS 服务 九千多个 API 操作中的任何一个的标准方法。 • 优化的集成 — 自定义集成，可简化与某些服务的呼叫和数据交换。例如，Lambda Invoke 会自动将响应`Payload`字段从转义的 JSON 字符串转换为 JSON 对象。
服务集成模式	调用时 AWS 服务，您可以使用以下服务集成模式之一： • 请求响应（默认）-在收到 HTTP 响应后立即调用服务并进入下一个状态。 • 运行作业 (.sync) – 调用服务，并让 Step Functions 等待作业完成。 • 等待带有任务令牌（.wait Toke ForTask n）的回调 — 使用任务令牌调用服务，让 Step Functions 等到任务令牌返回并带有回调。

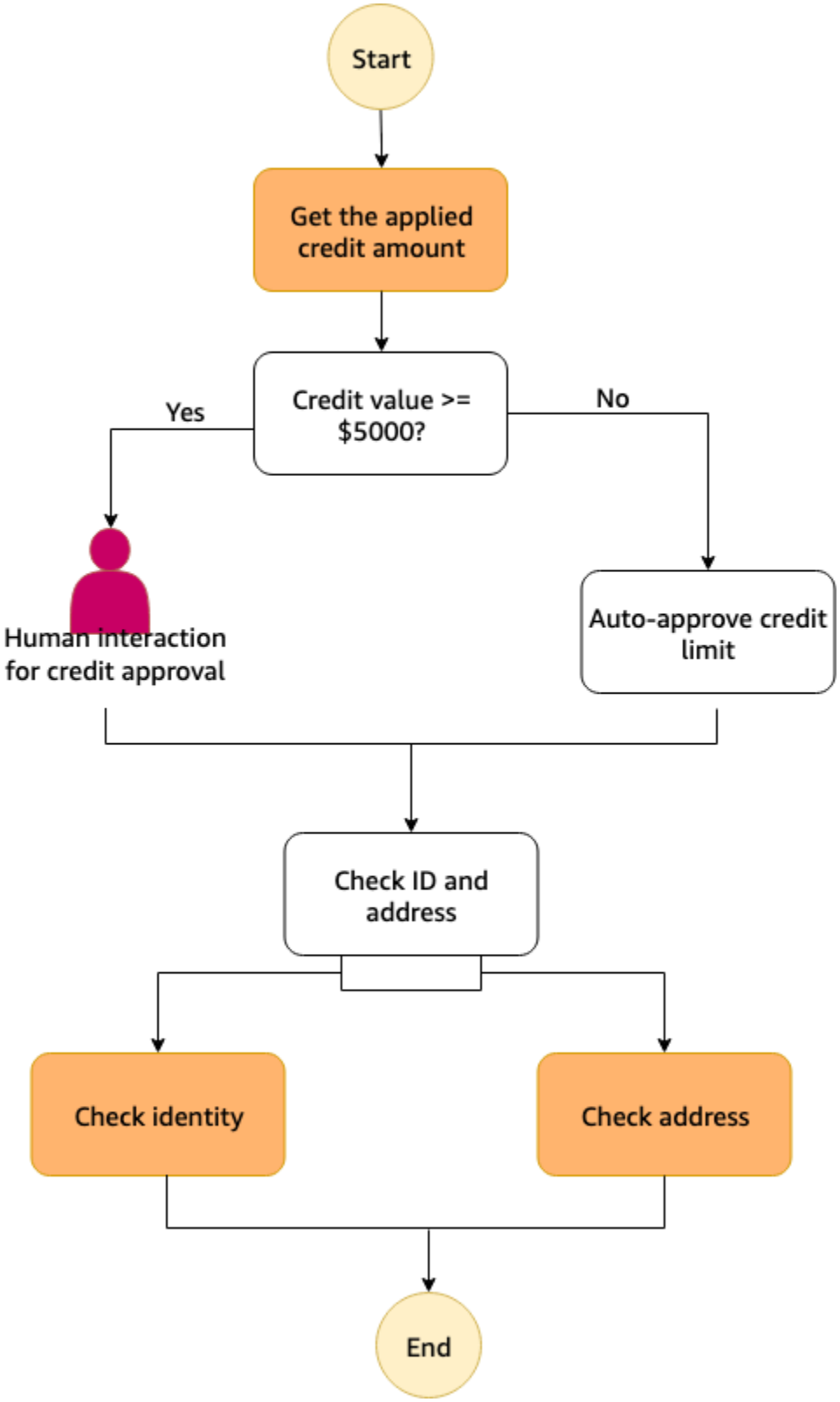
租期	描述
Execution	状态机执行是指运行工作流执行任务的实例。 有关更多信息，请参阅 Step Functions 中的执行。

本系列教程

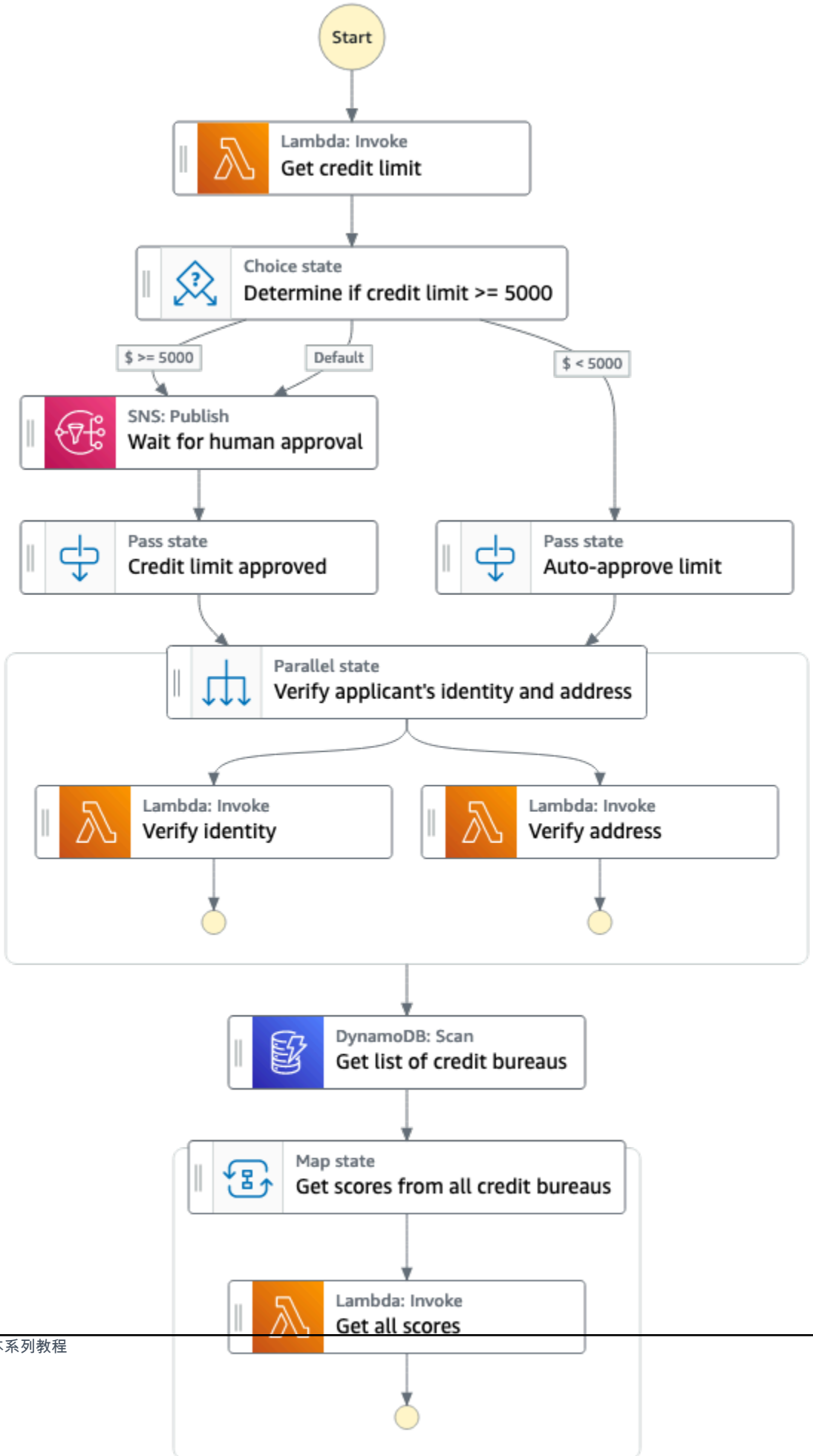
完成这些教程后，您将获得一个模拟处理信用卡申请的工作流程。您将学习如何使用常见状态以及如何将您的工作流程与其他状态集成 AWS 服务

Step Functions 可用于创建多种类型的工作流程，例如数据处理、IT 自动化、机器学习和媒体编码。

以下流程图描述了企业处理信用卡申请的步骤。如果申请的信用额度低于 5000 美元，则信用额度将自动获得批准。如果请求超出限制，工作流程将添加一名人员参与其中，以验证请求者的身份并查看信用评分。

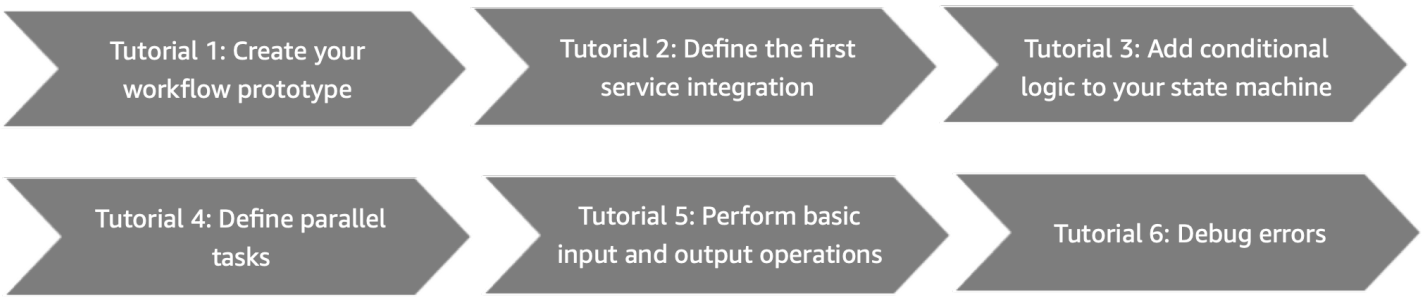


下图显示了 Step Functions 工作流中如何用状态表示信用申请业务流程步骤。


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

在以下系列教程中，您将构建信用卡处理工作流程。

我们建议您完成这些教程，以便学习 Step Functions 的主要功能。



在开始教程之前，请确保已完成先决条件。

入门的先决条件 AWS Step Functions

在 AWS Step Functions 首次使用之前，请完成以下任务。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅 《AWS 登录 用户指南》中的以根用户身份登录。

2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

教程 1：为状态机创建原型

在本教程中，您将使用 Step Functions 的 Workflow Studio 为信用卡处理工作流创建原型。您将分别从操作和流选项卡中选择所需的 API 操作和状态，然后使用 Workflow Studio 的拖放特征来创建工作流原型。在随后的教程中，您将学习如何配置将在此工作流中使用的 AWS 服务和 Step Functions 的状态。

创建状态机原型

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 在 Workflow Studio 中，从操作选项卡中，将AWS Lambda 调用 API 操作拖放到标有将第一个状态拖至此处的空白状态。按如下所示进行配置：
   - 在配置选项卡下，在状态名称中输入 **`Get credit limit`**。
5. 从流选项卡中，将 Choice 状态拖放到 Get credit limit 状态下方。将 Choice 状态重命名为 **`Credit applied >= 5000?`**。
6. 拖放一下状态，将其作为 Credit applied >= 5000? 状态的分支。
   a. Amazon SNS 发布 – 从操作选项卡中拖放 Amazon SNS 发布 API 操作。将此状态重命名为 **`Wait for human approval`**。
   b. Pass状态 - 从流选项卡中，拖放 Pass 状态。将此分支重命名为 **`Auto-approve limit`**。
7. 将 Pass 状态拖放到 Wait for human approval 状态下方。将此 Pass 状态重命名为 **`Credit limit approved`**。
8. 按如下所示将 Parallel 状态拖放到 Choice 状态之后：
   a. 在 Credit limit approved 状态之后放置 Parallel 状态。
   b. 将 Parallel 状态重命名为 **`Verify applicant's identity and address`**。
   c. 在 Parallel 状态的两个分支下，拖放两个 AWS Lambda 调用 API 操作。
   d. 将这两个状态分别重命名为 **`Verify identity`** 和 **`Verify address`**。
   e. 选择 Auto-approve limit 状态，然后在 Next 状态中选择Verify applicant's identity and address。
9. 将 DynamoDB Scan 状态拖放到 Verify applicant's identity and address 状态下方。将 DynamoDB Scan 状态重命名为 **`Get list of credit bureaus`**。

10. 将 Map 状态拖放到 Get list of credit bureaus 状态之后。按如下方式配置 Map 状态：
    a. 将其重命名为 **`Get scores from all credit bureaus`**。
    b. 对于处理模式，保留默认的内联选项。
    c. 将 AWS Lambda 调用 API 操作拖放到标有将状态拖至此处的空白状态处。
    d. 将 AWS Lambda 调用状态重命名为 **`Get all scores`**。
11. 保持此窗口处于打开状态，然后继续阅读下一个教程以进行后续操作。

后续步骤

在下一个教程中，您将学习如何集成获取信用额度状态使用的 Lambda 函数。

教程 2：使用 Lambda 函数定义第一个服务集成

在本教程中，您将了解如何为您的工作流定义第一个服务集成。您可以使用名为 Get credit limit 的 `Task` 状态来调用 Lambda 函数。在`Task`各州内，你可以使用 Step Functions 支持的 AWS SDK 集成。

要为您的工作流定义第一个服务集成，请先创建一个 Lambda 函数。然后，更新您的工作流，以便指定与 Lambda 函数的服务集成。本教程中使用的 Lambda 函数返回一个随机生成的整数，表示申请人申请的信用额度。

主题

- 第 1 步：创建并测试 Lambda 函数
- 第 2 步：更新工作流 – 配置“获取信用额度”状态
- 后续步骤

第 1 步：创建并测试 Lambda 函数

你可以在 AWS Management Console 或你最喜欢的编辑器中为该函数编写代码。在以下步骤中，您将创建名为 `RandomNumberforCredit` 的 Node.js Lambda 函数。

> **Important**
>
> 确保您在教程 1 中创建的工作流程原型与您将在本教程中创建的 Lambda 函数 AWS 区域 相同。

1. 在新选项卡或窗口中，打开 Lambda 控制台并创建一个名为 **RandomNumberforCredit** 的 Node.js 16.x Lambda 函数。有关使用控制台创建 Lambda 函数的信息，请参阅《AWS Lambda 开发人员指南》中的使用控制台创建 Lambda 函数。
2. 在该RandomNumberforCredit页面上，选择 index.mjs，然后将 “代码源” 区域中的现有代码替换为以下代码。

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

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

};
```

3. 在函数概述部分中，复制 Lambda 函数的 Amazon 资源名称并将其保存在文本文件中。在为 Get credit limit 状态指定服务集成时，您将需要函数 ARN。以下是示例 ARN：

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

4. 选择部署，然后选择测试以便部署更改并查看 Lambda 函数的输出。

第 2 步：更新工作流 – 配置“获取信用额度”状态

在 Step Functions 控制台中，您将更新工作流，以便指定与 `RandomNumberforCredit` Lambda 函数（第 1 步中创建的函数）的服务集成。

1. 打开 Step Functions 控制台窗口，其中应包含您在教程 1 中创建的工作流原型。
2. 选择 Get credit limit 状态，然后在配置选项卡中执行以下操作：
   a. 对于集成类型，保留默认的优化选项。

      使用 Step Functions，您可以与其他人集成 AWS 服务 并在工作流程中对其进行编排。有关服务集成及其类型的更多信息，请参阅 与其他服务 AWS Step Functions 一起使用。
   b. 在函数名称中，从下拉列表中选择 RandomNumberforCreditLambda 函数。
   c. 对于其余项目，保留默认选项。
3. 保持此窗口处于打开状态，然后继续阅读下一个教程以进行后续操作。

> **Note**
>
> 在本教程中，您学习了如何在工作流程的 `Task` 状态下与 Lambda 函数集成。您还可以通过指定服务名称和 API 调用，在`Task`状态下使用其他受支持的 AWS SDK 集成，如以下语法所示：
>
> ```
> arn:aws:states:::aws-sdk:serviceName:apiAction
> ```
>
> 有关更多信息，请参阅 与其他服务 AWS Step Functions 一起使用。

后续步骤

在下一个教程中，您将在工作流中实现条件逻辑。Step Functions 状态机中的条件逻辑的行为类似于大多数常见编程语言中的 if-else 语句。您将在工作流中使用条件逻辑，根据条件信息确定执行路径。

教程 3：在工作流中实现 if-else 条件

您可以使用 `Choice` 状态在工作流中实现 if-else 条件。它根据指定条件的计算结果是 true 还是 false 来确定工作流的执行路径。

在本教程中，您将添加条件逻辑来确定在教程 2 中使用的 `RandomNumberforCredit` Lambda 函数返回的申请的信用额度，是否超过特定的阈值限制。如果额度超过阈值限制，则申请需要人际互动才能获得批准。如果未超过限制，申请将自动批准，并进入下一个步骤。

您将需要暂停工作流执行，直至任务令牌返回，以此来模仿人际互动步骤。为此，您需要将一个任务令牌传递给将在本教程中使用的 AWS 开发工具包集成，即 Amazon Simple Notification Service。工作流执行将暂停，直到通过 `SendTaskSuccess` API 调用接收到任务令牌后再继续执行。有关使用任务令牌的更多信息，请参阅等待具有任务令牌的回调。

由于您已经在工作流原型中定义了人工批准和自动批准的步骤，因此在本教程中，您将首先创建一个接收回调令牌的 Amazon SNS 主题。然后，创建一个 Lambda 函数来实现回调功能。最后，您将通过添加这些 AWS 服务集成的详细信息来更新工作流原型。

主题

- 第 1 步：创建接收回调令牌的 Amazon SNS 主题
- 第 2 步：创建 Lambda 函数来处理回调

- 第 3 步：更新工作流 – 在“选择”状态下添加 if-else 条件逻辑
- 后续步骤

第 1 步：创建接收回调令牌的 Amazon SNS 主题

要实现人际互动步骤，您需要发布到 Amazon Simple Notification Service 主题并将回调任务令牌传递给该主题。回调任务将暂停工作流的执行，直至任务令牌与有效负载一起返回。

1. 打开 Amazon SNS 控制台，并创建一个标准主题类型。有关创建主题的信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的创建 Amazon SNS 主题。
2. 将主题名称指定为 **`TaskTokenTopic`**。
3. 确保复制主题 ARN 并将其保存在文本文件中。在为 Wait for human approval 状态指定服务集成时，您需要使用该主题 ARN。以下是一个示例主题 ARN：

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

4. 为该主题创建基于电子邮件的订阅，然后确认订阅。有关主题订阅的信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的创建主题订阅。

第 2 步：创建 Lambda 函数来处理回调

要处理回调功能，您需要定义一个 Lambda 函数，并将在第 1 步中创建的 Amazon SNS 主题添加为该函数的触发器。使用任务令牌发布到 Amazon SNS 主题时，将使用已发布消息的有效负载调用 Lambda 函数。

- 第 2.1 步：创建 Lambda 函数来处理回调
- 第 2.2 步：添加 Amazon SNS 主题作为 Lambda 函数的触发器
- 第 2.3 步：向 Lambda 函数 IAM 角色提供必要的权限

第 2.1 步：创建 Lambda 函数来处理回调

在此函数中，您将处理信用额度批准申请，并通过 `SendTaskSuccess` API 调用返回申请的成功结果。此 Lambda 函数还将返回它从 Amazon SNS 主题中接收的任务令牌。

为简单起见，用于人际互动步骤的 Lambda 函数会自动批准所有任务，并通过 `SendTaskSuccess` API 调用返回任务令牌。您可以将 Lambda 函数命名为 **`callback-human-approval`**。

1. 在新选项卡或窗口中，打开 Lambda 控制台并创建一个名为 **callback-human-approval** 的 Node.js 16.x Lambda 函数。有关使用控制台创建 Lambda 函数的信息，请参阅《AWS Lambda 开发人员指南》中的使用控制台创建 Lambda 函数。
2. 在 callback-human-approval 页面上，将代码源区域中已有的代码替换为以下代码。

```
// 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. 保持此窗口处于打开状态，并执行下一节中的步骤，进行后续操作。

第 2.2 步：添加 Amazon SNS 主题作为 Lambda 函数的触发器

当您添加在本教程第 1 步中创建的 Amazon SNS 主题作为在本教程第 2.1 步中创建的 Lambda 函数的触发器时，则每次发布到 Amazon SNS 主题时都会触发 Lambda 函数。使用任务令牌发布到 Amazon

SNS 主题时，将使用已发布消息的有效负载调用 Lambda 函数。有关为 Lambda 函数配置触发器的更多信息，请参阅《AWS Lambda 开发人员指南》中的配置触发器。

1. 在 `callback-human-approval` Lambda 函数的函数概述部分中，选择添加触发器。
2. 从触发器下拉列表中，选择 SNS 作为触发器。
3. 对于 SNS 主题，键入您在本教程第 1 步中创建的 Amazon SNS 主题的名称，然后从出现的下拉列表中进行选择。
4. 选择添加。
5. 保持此窗口处于打开状态，并执行下一节中的步骤，进行后续操作。

第 2.3 步：向 Lambda 函数 IAM 角色提供必要的权限

您必须向 `callback-human-approval` Lambda 函数提供访问 Step Functions 的权限，以便随 `SendTaskSucess` API 调用一起返回任务令牌。

1. 在 callback-human-approval 页面上，选择配置选项卡，然后选择权限。
2. 在执行角色下，选择角色名称以导航到 AWS Identity and Access Management 控制台的角色页面。
3. 如需添加所需的权限，选择添加权限，然后选择附加策略。
4. 在 搜索框中，输入 **`AWSStepFunctions`**，然后按 Enter 键。
5. 选择 AWSStepFunctionsFullAccess，然后向下滚动选择附加策略。这将添加包含 `callback-human-approval` Lambda 函数角色必要权限的策略。

第 3 步：更新工作流 – 在“选择”状态下添加 if-else 条件逻辑

在 Step Functions 控制台中，使用 `Choice` 状态为工作流定义条件逻辑。如果 `RandomNumberforCredit` Lambda 函数返回的输出小于 5000，则会自动批准所申请的额度。如果返回的输出大于或等于 5000，则工作流执行将进入信用额度审批的人际互动步骤。

在 `Choice` 状态下，您可以使用比较运算符将输入变量与特定值进行比较。您可以在启动状态机执行时将输入变量指定为执行输入，也可以使用前一步的输出作为当前步骤的输入。默认情况下，步骤的输出存储在名为 `Payload` 的变量中。要在 `Choice` 状态中使用 `Payload` 变量的值进行比较，请使用以下过程所示的 `$` 语法。

有关信息如何从一种状态流向另一种状态以及如何在工作流中指定输入和输出的信息，请参阅教程 7：配置输入和输出和 Step Functions 中的输入和输出处理。

> **Note**
>
> 如果 Choice 状态使用状态机执行输入中指定的输入变量进行比较，请使用 $.variable_name 语法执行比较。例如，要比较 myAge 变量，请使用语法 $.myAge。

由于在此步骤中，该 Choice 状态将接收来自Get credit limit状态的输入，您将使用 $ 语法来配置 Choice 状态。要了解在 Choice 状态配置中使用 $.variable_name 语法来引用前一步的输出时，状态机执行结果有何不同，请参阅教程 8 中的调试无效路径 Choice 状态错误部分。

使用 **Choice** 状态添加 if-else 条件逻辑

1. 打开 Step Functions 控制台窗口，其中应包含您在教程 1：为状态机创建原型中创建的工作流原型。
2. 选择已 Credit applied >= 5000? 状态，然后在配置选项卡中，按如下方式指定条件逻辑：
   a. 在选择规则下，选择规则 #1 磁贴中的编辑图标，定义第一条选择规则。
   b. 选择添加条件。
   c. 在规则 #1 的条件对话框中，对于变量，输入 **$**。
   d. 对于运算符，选择小于。
   e. 对于值，选择数字常量，然后在值下拉列表旁边的字段中输入 **5000**。
   f. 选择保存条件。
   g. 对于然后下一个状态是：下拉列表，选择自动批准额度。
   h. 选择添加新选择规则，然后重复 2.b 至 2.f 子步骤来定义第二条选择规则，该规则适用于信用额度大于或等于 5000 的情况。对于运算符，选择大于或等于。
   i. 对于然后下一个状态是：下拉列表，选择等待人工批准。
   j. 在默认规则框中，选择编辑图标以定义默认选择规则，然后从默认状态下拉列表中选择 Wait for human approval。您可以定义“默认规则”，以指定在 Choice 状态条件均未评估为 true 或 false 时要过渡到的下一个状态。
3. 按如下方式配置等待人工批准状态：
   a. 在配置选项卡中，对于主题，开始键入 Amazon SNS 主题的名称 TaskTokenTopic，然后选择下拉列表中显示的名称。
   b. 对于消息，从下拉列表中选择输入消息。在消息字段中，您可以指定要发布到 Amazon SNS 主题的消息。在本教程中，您将任务令牌作为消息发布。

使用任务令牌，您可以暂停标准类型的 Step Functions 工作流，直到外部流程完成并返回任务令牌。通过指定 `.waitForTaskToken` 服务集成模式将 Task 状态指定为回调任务时，会在任务启动时生成任务令牌并将其放置在上下文对象中。上下文对象是执行期间可用的内部 JSON 结构，包含有关状态机及其执行的信息。有关上下文对象的更多信息，请参阅Context 对象。

c. 在出现的框中，将以下内容作为消息输入：

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

d. 选中等待回调复选框。

e. 在随后显示的对话框中，选择完成。

4. 保持此窗口处于打开状态，然后继续阅读下一个教程以进行后续操作。

后续步骤

在下一个教程中，您将学习如何并行执行多个任务。

教程 4：定义并行执行的多项任务

到目前为止，您已经学会了如何按顺序运行工作流。然而，您还可以使用 `Parallel` 状态并行运行两个或多个步骤。`Parallel` 状态会使解释器同时执行每个分支。

处于同一个 `Parallel` 状态下的两个分支接收相同的输入，但是每个分支都会处理特定于该分支的输入部分。Step Functions 会等待每个分支斗执行完毕后再进行下一步。

在本教程中，您将使用 Parallel 状态同时检查申请人的身份和地址。

主题

- 第 1 步：创建 Lambda 函数执行所需的检查
- 第 2 步：更新工作流 – 添加要执行的并行任务

第 1 步：创建 Lambda 函数执行所需的检查

此信用卡申请工作流在 Parallel 状态下调用两个 Lambda 函数，用于检查申请人的身份和地址。两项检查使用 Parallel 状态同步进行。只有在两个并行分支都执行完毕后，状态机才会完成执行。

创建 check-identity 和 check-address Lambda 函数

1. 在新选项卡或窗口中，打开 Lambda 控制台，并创建两个名为 `check-identity` 和 `check-address` 的 Node.js 16.x Lambda 函数。有关使用控制台创建 Lambda 函数的信息，请参阅《AWS Lambda 开发人员指南》中的使用控制台创建 Lambda 函数。
2. 打开 check-identity 函数页面，将代码源区域中已有的代码替换为以下代码：

```
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. 打开 check-address 函数页面，将代码源区域中已有的代码替换为以下代码：

```
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. 对于这两个 Lambda 函数，请从函数概述部分复制各自的 Amazon 资源名称 (ARN)，并保存到一个文本文件中。在为 Verify applicant's identity and address 状态指定服务集成时，您需要这两个函数 ARN。以下是示例 ARN：

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

第 2 步：更新工作流 – 添加要执行的并行任务

在 Step Functions 控制台中，您将更新工作流，以便指定与您在第 1 步中创建的 check-identity 和 check-address Lambda 函数的服务集成。

在工作流中添加并行任务

1. 打开 Step Functions 控制台窗口，其中应包含您在教程 1：为状态机创建原型中创建的工作流原型。
2. 选择 Verify identity 状态，然后在配置选项卡中执行以下操作：
   a. 对于集成类型，保留默认的优化选项。

      > **Note**
      >
      > 使用 Step Functions，您可以与其他 AWS 服务集成，并在工作流中对其进行编排。有关服务集成及其类型的更多信息，请参阅 与其他服务 AWS Step Functions 一起使用

   b. 对于函数名称，从下拉列表中选择 check-identity Lambda 函数。
   c. 对于有效负载，选择输入有效负载，然后将示例有效负载替换为以下有效负载：

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

3. 选择 Verify address 状态，然后在配置选项卡中执行以下操作：
   a. 对于集成类型，保留默认的优化选项。
   b. 对于函数名称，从下拉列表中选择 check-address Lambda 函数。
   c. 对于有效负载，选择输入有效负载，然后将示例有效负载替换为以下有效负载：

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

4. 选择下一步。

教程 5：同时迭代一组项目

在上一个教程中，您学习了如何使用 Parallel 状态并行运行单独的步骤分支。使用 Map 状态，您可以为数据集中的每个项目运行一组工作流步骤。 Map 状态的迭代是并行运行的，可以实现快速处理数据集。

通过将 Map 状态包含在工作流中，您可以使用以下两种 Map 状态处理模式 之一来执行数据处理等任务：内联模式和分布式模式。要配置 Map 状态，您需要定义一个 ItemProcessor，其中包含用于指定 Map 状态处理模式及其定义的 JSON 对象。在本教程中，您将在默认的内联模式下运行 Map 状态，该模式最多支持 40 次并发迭代。当您运行分布式模式下的 Map 状态时，它最多支持 10000 个并行子工作流执行。

当您的工作流执行进入 Map 状态时，它将迭代状态输入中指定的 JSON 数组。对于每个数组项目，其相应的迭代在包含 Map 状态工作流的上下文中运行。当所有迭代完成后，Map 状态将返回一个数组，其中包含由 ItemProcessor 处理的每个项目的输出。

在本教程中，您将学习如何在内联模式下使用 Map 状态，通过迭代一组征信机构来获取申请人的信用评分。为此，您首先要获取存储在 Amazon DynamoDB 表中的所有征信机构的名称，然后使用 Map 状态循环访问征信机构列表，以获取每个征信机构报告的申请人的信用评分。

主题

- 第 1 步：创建 DynamoDB 表来存储所有征信机构的名称
- 第 2 步：更新状态机 – 从 DynamoDB 表中获取结果
- 第 3 步：创建可返回所有征信机构信用评分的 Lambda 函数
- 第 4 步：更新状态机 – 添加 Map 状态以迭代方式获取信用评分

第 1 步：创建 DynamoDB 表来存储所有征信机构的名称

在本步骤中，您将使用 DynamoDB 控制台创建一个名为 **GetCreditBureau** 的表。该表使用字符串属性 Name 作为分区键。在此表中，您存储了希望从中获取申请人信用评分的所有征信机构的名称。

1. 登录 AWS Management Console，并打开 DynamoDB 控制台：https://console.aws.amazon.com/dynamodb/。
2. 在控制台的导航窗格中，选择表，然后选择创建表。
3. 按以下所示输入表详细信息：

    a. 对于表名称，输入 **GetCreditBureau**。

b. 对于分区键，输入 **Name**。

c. 其他保留默认选择，然后选择创建表。

4. 创建表后，在表列表中，选择 GetCreditBureau 表。
5. 选择操作，然后选择创建项目。
6. 在值中，输入征信机构的名称。例如，**CredTrack**。
7. 选择创建项目。
8. 重复此过程，为其他征信机构的名称创建项目。例如，**KapFinn** 和 **CapTrust**。

第 2 步：更新状态机 – 从 DynamoDB 表中获取结果

在 Step Functions 控制台中，您将添加一个 Task 状态，然后使用 AWS 开发工具包集成从您在第 1 步中创建的 DynamoDB 表中获取征信机构的名称。您将使用此步骤的输出作为 Map 状态的输入，稍后您将在本教程的工作流中添加该状态。

1. 打开 CreditCardWorkFlow 状态机进行更新。
2. 选择 Get list of credit bureaus 状态。
3. 对于 API 参数，将表名称值指定为 **GetCreditBureau**。

第 3 步：创建可返回所有征信机构信用评分的 Lambda 函数

在此步骤中，您将创建一个 Lambda 函数，用于接收所有征信机构的名称作为输入，并返回申请人在每个征信机构中的信用评分。此 Lambda 函数将从您将在本教程第 4 步中添加到工作流中的 Map 状态调用。

1. 创建一个 Node.js 16.x Lambda 函数，并将其命名 **get-credit-score**。
2. 在标题为 get-credit-score 的页面上，将以下代码粘贴到代码源区域。

```
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. 部署 Lambda 函数。

第 4 步：更新状态机 – 添加 Map 状态以迭代方式获取信用评分

在 Step Functions 控制台中，您将添加一个 Map 状态，用于调用 get-credit-score Lambda 函数来检查 Get list of credit bureaus 状态返回的申请人在所有征信机构中的信用评分。

1. 打开 CreditCardWorkFlow 状态机进行更新。
2. 选择 Get scores from all credit bureaus 状态。
3. 在配置选项卡中，选择提供项目数组的路径，然后输入 **`$.Items`**。
4. 在 Map 状态中选择获取所有评分步骤。
5. 在配置选项卡中，确保在集成类型中选择了优化。
6. 在函数名称中，键入 get-credit-score Lambda 函数的名称，然后从出现的下拉列表中进行选择它。
7. 对于有效负载，选择无有效负载。

教程 6：保存工作流并执行状态机

现在，您已经配置了工作流程原型中使用的所有资源，可以将其另存为 Step Functions 状态机并开始执行。 AWS 服务

主题

- 第 1 步：查看自动生成的状态机定义并保存状态机
- 第 2 步：添加剩余的 IAM 策略
- 第 3 步：运行状态机

第 1 步：查看自动生成的状态机定义并保存状态机

当您将状态从流选项卡中拖放到 Workflow Studio 的画布上，构建工作流原型时，Step Functions 会自动实时撰写工作流的 Amazon States Language (ASL) 定义。您可以根据需要在代码编辑器中编辑此定义。

查看 ASL 定义并保存状态机

1. （可选）在 Inspector 上选择定义，查看状态机 Amazon States Language (ASL) 定义，该定义是根据您在操作和流选项卡以及 Inspector 面板中的选择自动生成的。

   > **Tip**
   >
   > 要编辑定义，您可以通过选择页面顶部的代码来打开代码编辑器。在本教程中，请使用自动生成的定义继续操作。

2. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **`CreditCardWorkflow`**。

3. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。

   > **Note**
   >
   > （可选）Step Functions 会自动为状态机创建一个执行角色，该角色具有调用 `RandomNumberforCredit` Lambda 函数并发布到 Amazon SNS 主题所需的最低权限。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

4. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 2 步：添加剩余的 IAM 策略

由于 Step Functions 不会自动生成在 `Parallel` 状态下调用 Lambda 函数的权限，因此您需要添加必要的策略。

添加剩余的策略

1. 在该CreditCardWorkflow页面上，为您的状态机选择 IAM 角色以导航到 IAM 控制台。您将在此页面上为其余 Lambda 函数添加必要的权限。
2. 选择添加权限，然后选择附加策略。
3. 在搜索框中，输入 **`AWSLambdaRole`**，然后按 Enter 键。
4. 选择 AWSLambdaRole，然后选择 “附加策略”。现在，此策略已添加到状态机的执行角色中。此策略允许您在状态机中调用任何 Lambda 函数。

第 3 步：运行状态机

状态机执行是指运行工作流执行任务的实例。

执行状态机

1. 在CreditCardWorkflow页面上，选择 “开始执行”。

   随即显示启动执行对话框。
2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

> **Note**
>
> 您无需提供任何输入，即可执行此状态机。但是，如果需要，您可以在其他状态机的启动执行对话框的输入区域中指定执行输入。有关如何向状态机提供执行输入的示例，请参阅第 4 步：开始重新执行学习使用 Workflo AWS Step Functions w Studio 教程。

b. 选择启动执行。

3. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

教程 7：配置输入和输出

Step Functions 执行接收 JSON 文本作为输入并将该输入传递到工作流中的第一个状态。工作流中的各个状态将接收输入形式的 JSON 数据并通常将 JSON 数据以输出形式传递到下一个状态。默认情况下，数据会从工作流中的一个状态传递到下一个状态，除非您为工作流中的一个或多个状态配置了输入和/或输出。了解信息如何在状态之间流动，并了解如何筛选和处理此数据，这对在 Step Functions 中高效设计和实施工作流至关重要。

Step Functions 提供了多个筛选条件来控制状态之间的输入和输出数据流。以下筛选条件可在您的工作流中使用：

> **Note**
>
> 根据您的使用案例，您可能不需要在工作流中应用所有这些筛选条件。

InputPath

选择将整个输入有效负载的哪个 部分用作任务的输入。如果您指定了此字段，Step Functions 将首先应用此字段。

##

指定在调用任务之前输入的外观。使用 Parameters 字段，您可以创建一个键值对的集合，这些键值对作为输入传递给 AWS 服务集成（例如 AWS Lambda 函数）。这些值可以是静态的，也可以从状态输入或工作流上下文对象中动态选择。

ResultSelector

确定从任务的输出中选择什么。使用 ResultSelector 字段，您可以创建一个键值对的集合，这些键值对用于替换状态的结果并将该集合传递给 ResultPath。

ResultPath

确定任务输出的放置位置。使用 ResultPath 来确定状态的输出是其输入的副本、其产生的结果还是两者的组合。

OutputPath

确定发送到下一个状态的内容。使用 OutputPath，您可以筛选出不需要的信息，并且仅传递所需的 JSON 数据。

> **Tip**
>
> Parameters 和 ResultSelector 筛选条件通过构造 JSON 来生效，而 InputPath 和 OutputPath 筛选条件通过筛选 JSON 数据对象中的特定节点来生效，ResultPath 筛选条件通过创建一个可以在其中添加输出的字段来生效。

在本教程中，您将学习如何执行以下任务：

- 使用 InputPath 过滤器选择原始输入的特定部分
- 使用 Parameters 筛选条件处理选定的输入
- 使用 ResultSelector、ResultPath 和 OutputPath 筛选条件配置输出

有关在工作流中配置输入和输出的更多信息，请参阅 Step Functions 中的输入和输出处理。

使用 InputPath 过滤器选择原始输入的特定部分

使用 `InputPath` 筛选条件，选择输入有效负载的特定部分。

如果您未指定 `InputPath`，则其值默认为 `$`，这会导致状态的任务将引用整个原始输入而不是特定部分。

要学习如何使用 InputPath 筛选条件，请执行以下步骤：

- 第 1 步：创建状态机
- 第 2 步：运行状态机
- 第 3 步：使用 InputPath 筛选条件选择执行输入的特定部分

第 1 步：创建状态机

> **Important**
>
> 确保您的状态机与您之前创建的 Lambda 函数位于相同的 AWS 账户和区域下。

1. 使用您在教程 4 中学到的 `Parallel` 状态示例来创建新的状态机。确保工作流原型类似于以下原型。
2. 为 `check-identity` 和 `check-address` Lambda 函数配置集成。有关创建 Lambda 函数并在状态机中使用函数的信息，请参阅第 1 步：创建 Lambda 函数执行所需的检查和第 2 步：更新工作流 – 添加要执行的并行任务。
3. 对于有效负载，请确保保留使用状态输入作为有效负载的默认选择。
4. 选择下一步，然后执行教程 5 第 1 步：保存状态机中的第 1 至 3 步，来创建新的zhuang't状态机。在本教程中，请将状态机命名为 **`WorkflowInputOutput`**。

第 2 步：运行状态机

1. 在WorkflowInputOutput页面上，选择开始执行。
2. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

3. 在输入区域中，添加以下 JSON 数据作为执行输入。

```
{
  "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. 选择启动执行。
5. 状态机执行出现错误，这是因为您尚未指定 `check-identity` 和 `check-address` Lambda 函数必须使用执行输入的哪一部分来执行所需的身份和地址验证。
6. 继续执行本教程的第 3 步，以便修复错误。

第 3 步：使用 **InputPath** 筛选条件选择执行输入的特定部分

1. 在执行详细信息页面上，选择编辑状态机。
2. 要验证第 2 步：运行状态机中提供的执行输入中提及的申请人身份，请按如下方式编辑 Verify identity 任务定义：

```
...
{
    "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
      }
    }
  }
  ...
```

因此，以下 JSON 数据可用作 `check-identity` 函数的输入。

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

3. 要验证执行输入中提及的申请人地址，请按如下方式编辑 `Verify address` 任务定义：

```
...
{
    "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
      }
    }
  }
```

```
  ...
```

因此，以下 JSON 数据可用作 check-address 函数的输入。

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

4. 选择启动执行。状态机执行现在成功完成。

使用 Parameters 筛选条件处理选定的输入

虽然 InputPath 筛选条件可以帮助您限制所提供的原始 JSON 输入，但使用 Parameters 筛选条件，您可以将一组键值对作为输入传递。这些键值对可以是您在状态机定义中定义的静态值，也可以是使用 InputPath 从原始输入中选择的值。

在您的工作流中，Parameters 将在 InputPath 之后应用。Parameters 帮助您指定底层任务接受其输入有效负载的方式。例如，如果 check-address Lambda 函数接受字符串参数而不是 JSON 数据作为输入，则您可以使用 Parameters 筛选条件来转换输入。

在以下示例中，Parameters 筛选条件接收您在第 3 步：使用 InputPath 筛选条件选择执行输入的特定部分中使用 InputPath 选择的输入，并将内置函数 States.Format 应用于输入项，用于创建名为 addressString 的字符串。内置函数可帮助您对给定输入执行基本的数据处理操作。有关更多信息，请参阅内置函数。

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

因此，将创建以下字符串并将其作为输入提供给 check-address Lambda 函数。

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

使用 ResultSelector、ResultPath 和 OutputPath 筛选条件配置输出

在 WorkflowInputOutput 状态机中调用 `check-address` Lambda 函数时，该函数将在执行地址验证后返回输出有效负载。在执行详细信息页面上，选择验证地址步骤，然后在步骤详细信息窗格的任务结果中查看输出有效负载。

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

使用 ResultSelector

现在，如果您需要向工作流中的以下状态提供身份和地址验证结果，则可以在输出 JSON 中选择 Payload.body 节点，然后使用 `ResultSelector` 筛选条件中的 `StringToJson` 内置函数根据需要格式化数据。

`ResultSelector` 从任务输出中选择所需的内容。在以下示例中，`ResultSelector` 获取 $.Payload.body 中的字符串，并应用 `States.StringToJson` 内置函数将该字符串转换为 JSON，并将生成的 JSON 放入身份节点中。

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

结果会创建以下 JSON 数据。

```
{
    "identity": {
```

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

在使用这些输入和输出筛选条件时，您还可能会遇到因指定了无效的 JSON 路径表达式而导致的运行时错误。有关更多信息，请参阅。

使用 ResultPath

您可以使用 ResultPath 字段在初始输入有效负载中指定一个位置，来保存状态的任务处理结果。如果您未指定 ResultPath，则采用默认值 $，这会导致初始输入有效负载被原始任务结果替换。如果指定 ResultPath 为 null，则原始结果将被丢弃，初始输入有效负载将成为有效输出。

如果您对使用 ResultSelector 字段创建的 JSON 数据应用 ResultPath 字段，任务结果将添加到输入有效负载的结果节点内，如以下示例所示：

```
{
  "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
    }
  }
}
```

使用 OutputPath

在应用 ResultPath 之后，您可以选择状态输出的一部分以传递到下一个状态。这可让您筛选出不需要的信息，并且仅传递所需的 JSON 部分。

在以下示例中，OutputPath 字段将状态输出保存在结果节点内："OutputPath": "$.results"。因此，该状态的最终输出可以传递到下一个状态，如下所示：

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

使用控制台特征可视化输入和输出数据流

您可以使用 Step Functions 控制台的数据流模拟器或“执行详细信息”页面中的“高级”视图选项，直观显示工作流中各状态之间的输入和输出数据流。

教程 8：在控制台中调试错误

在使用 Step Functions 时，可能会遇到由某些原因引起的运行时错误，原因示例如下：

- Choice 状态中变量字段的 JSON 路径无效。
- 状态机定义问题，例如没有为 Choice 状态匹配规则。
- 应用筛选条件处理输入和输出时，JSON 路径表达式无效。
- 由于 Lambda 函数异常导致任务失败。
- IAM 权限错误。

在本教程中，您将学习如何使用 Step Functions 控制台调试其中一些错误。有关更多信息，请参阅Step Functions 中的错误处理。

主题

- 调试无效路径 Choice 状态错误
- 在应用输入和输出筛选条件时调试 JSON 路径表达式错误

调试无效路径 Choice 状态错误

当您在 Choice 状态的变量字段中指定了不正确或无法解析的 JSON 路径，或者未在 Choice 状态中定义匹配规则时，您在运行工作流时会收到一个错误。

为了说明无效路径错误，本教程在您的工作流中引入了一个 Choice 状态错误。您将使用 CreditCardWorkflow 状态机并编辑其定义，引入错误。

1. 打开 Step Functions 控制台，然后选择 CreditCardWorkflow 状态机。
2. 选择编辑，编辑状态机定义。对状态机定义按照下方代码中突出显示的内容进行更改。

```
{
  "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. 选择保存，然后选择仍然保存。
4. 运行该状态机。

5. 在状态机执行的“执行详细信息”页面上，执行下列操作之一：

   a. 在错误消息上选择原因，查看执行失败的原因。

   b. 在错误消息上选择显示步骤详细信息，查看导致错误的步骤。

6. 在步骤详细信息部分的输入和输出选项卡中，选择高级视图切换按钮，查看所选状态的输入和输出数据传输路径。

7. 在图表视图中，确保已选中 Credit applied >= 5000?，然后执行以下操作：

   a. 在输入框中查看该状态机的输入值。

   b. 选择定义选项卡，注意为变量字段指定的 JSON 路径。

      Credit applied >= 5000? 状态的输入值是一个数值，而您已将该输入值的 JSON 路径指定为 `$.Payload`。在状态机执行期间，`Choice` 状态无法解析此 JSON 路径，因为该路径不存在。

8. 编辑状态机，将变量字段值指定为 **$**。

```
{
  "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"
    },
    ...
```

```
    ...
  }
}
```

在应用输入和输出筛选条件时调试 JSON 路径表达式错误

在使用输入和输出筛选条件时，您可能会遇到因指定了无效的 JSON 路径表达式而导致的运行时错误。

以下示例使用您在教程 5 中创建的 WorkflowInputOutput 状态机，并演示了使用 `ResultSelector` 筛选条件选择部分任务输出的场景。

1. 应用 `ResultSelector` 筛选条件为验证身份步骤选择部分任务输出。为此，请按如下方式编辑状态机定义：

```
{
  "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. 运行该状态机。
3. 在状态机执行的“执行详细信息”页面上，执行以下操作：

a. 在错误消息上选择原因，查看执行失败的原因。

b. 在错误消息上选择显示步骤详细信息，查看导致错误的步骤。

4. 在错误消息中，请注意 $.Payload.body 节点的内容是一个转义的 JSON 字符串。之所以出现此错误，是因为您无法使用 JSON 路径表示法引用字符串。

5. 要引用 $.Payload.body.message 节点，请执行以下操作：

a. 首先使用 States.StringToJSON 内置函数将字符串转换为 JSON 格式。

b. 在内置函数中指定 $.Payload.body.message 节点的 JSON 路径。

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

6. 再次运行状态机。

使用案例

使用 AWS Step Functions，您可以构建可视化工作流，帮助您将业务需求快速转化为应用程序。Step Functions 为您管理状态、检查点和重启，并提供自动处理错误和异常的内置功能。为了更好地了解 Step Functions 可以为您提供的功能，请通读以下使用案例：

主题

- 数据处理
- 机器学习
- 微服务编排
- IT 和安全自动化

数据处理

随着数据量不断增长，数据来源越来越多样化，各个组织发现他们需要快速处理这些数据，才能确保他们做出更快、更明智的业务决策。为了大规模处理数据，组织需要灵活地调配资源，来管理他们从移动设备、应用程序、卫星、市场营销和销售、运营数据存储、基础设施等接受的信息。

Step Functions 提供了成功管理数据处理工作流所需的可扩展性、可靠性和可用性。Step Functions 可以横向扩展并提供容错工作流，因此您可以使用 Step Functions 管理数百万个并发执行。使用并行执行（例如 Step Functions 的 `Parallel` 状态类型）或动态并行（使用 Step Functions 的 `Map` 状态类型）可更快地处理数据。作为工作流的一部分，您可以使用 `Map` 状态对静态数据存储（如 Amazon S3 存储桶）中的对象进行迭代。Step Functions 还可以帮助您轻松重试失败的执行，或者选择特定的方式来处理错误，而无需管理复杂的流程。

根据数据处理的需求，Step Functions 可直接与 AWS 提供的其他数据处理服务集成，例如用于批处理的 AWS Batch、用于大数据处理的 Amazon EMR、用于数据准备的 AWS Glue、用于数据分析的 Athena，以及用于计算的 AWS Lambda。

客户使用 Step Functions 完成的数据处理工作流类型的示例包括：

文件、视频和图像处理

- 将一系列视频文件转换为大小或分辨率不同的文件，从而适配显示这些文件的设备，例如手机、笔记本电脑或电视。
- 将用户上传的大量照片转换为缩略图或各种分辨率的图像，以便在用户的网站上显示。

- 将半结构化数据（例如 CSV 文件）与非结构化数据（例如发票）结合起来，生成每月发送给业务利益相关者的业务报告。
- 将卫星收集的地球观测数据转换为相互对应的格式，然后添加在地球上收集的其他数据来源以获得更多见解。
- 从各种运输方式中获取产品的运输日志，并使用蒙特卡罗模拟进行优化，然后将报告发送回使用您的运输运送货物的组织和人员。

协调提取、转换、加载 (ETL) 任务：

- 使用 AWS Glue 进行一系列数据准备步骤，将销售机会记录与营销指标数据集相结合，并生成可在整个组织中使用的商业智能报告。
- 创建、启动和终止用于大数据处理的 Amazon EMR 集群。

批处理和高性能计算 (HPC) 工作负载：

- 构建基因组二级分析管道，将原始的全基因组序列处理成变异调用。将原始文件与参考序列对齐，并使用动态并行调用指定染色体列表上的变异。
- 通过使用不同的电气和化合物模拟各种布局，提高下一代移动设备或其他电子产品的生产效率。通过各种模拟对工作负载进行大批量处理，以便获得最佳设计。

机器学习

机器学习能够帮助组织快速分析收集的数据，识别模式，从而在做出决策时尽可能减少人为干预。机器学习始于一组初始数据，即训练数据。这些训练数据有助于提高机器学习模型的预测准确性，也是该模型学习的基础。模型经过验证，准确性达到要求，足以满足业务需求后，便会将其部署到生产中。AWS Step Functions 数据科学软件开发工具包 (SDK) 是一个开源库，允许您使用 Amazon SageMaker 和 Step Functions 轻松创建预处理数据、执行训练，以及发布模型的工作流。

预处理现有数据集是组织创建训练数据的常用方式。这种方法可以添加信息，例如标注图像中的对象、注释文本或处理音频。要预处理数据，您可以使用 AWS Glue，也可以创建一个运行 Jupyter 笔记本应用程序的 SageMaker 笔记本实例。数据准备就绪后，可以将其上传到 Amazon S3 以便于访问。训练机器学习模型完成后，您可以调整每个模型的参数，提高准确性，直到模型可以部署为止。

Step Functions 允许您在 SageMaker 上编排端到端的机器学习工作流。这些工作流可能包括数据预处理、后处理、特征工程、数据验证和模型评估。将模型部署到生产后，您可以改进和测试新方法，持续

改善业务成果。您可以直接在 Python 中创建生产就绪型工作流，也可以使用 Step Functions 数据科学开发工具包复制该工作流，体验新选项，然后将改进后的工作流投入到生产中。

客户使用 Step Functions 的机器学习工作流类型包括：

欺诈侦测

- 识别并防止欺诈性交易（例如信用欺诈）的发生。
- 使用经过训练的机器学习模型侦测并预防账户盗用。
- 识别促销滥用行为，包括创建虚假账户，以便您可以快速采取行动。

个性化和推荐

- 根据对目标客户兴趣的预测，向其推荐产品。
- 预测客户是否会将其账户从免费套餐升级为付费订阅。

数据扩充

- 将数据扩充作为预处理的一部分，为更精确的机器学习模型提供更好的训练数据。
- 为文本和音频摘录添加注释，以便添加语法信息，例如讽刺和俚语。
- 为图像中的其他对象添加标签，为模型学习提供关键信息，例如对象是苹果、篮球、岩石还是动物。

微服务编排

微服务架构将应用程序分解为松耦合的服务。这样做的好处包括提高可扩展性、增强弹性和加快上市时间。每个微服务都是独立的，因此无需扩展整个应用程序，即可轻松纵向扩展单个服务或功能。各个服务皆为松耦合，这让独立团队可以专注于单个业务流程，而无需了解整个应用程序。微服务还允许您选择适合业务需求的单个组件，从而无需重写整个工作流，即可灵活地更改选择。不同的团队可以使用自己选择的编程语言和框架来处理他们的微服务，并且该微服务仍然可以通过应用程序编程接口 (API) 与应用程序中的任何其他微服务进行通信。

Step Functions 为您提供了多种管理微服务工作流的方法。对于长时间运行的工作流，您可以将标准工作流与 AWS Fargate 集成一起使用，来编排在容器中运行的应用程序。对于需要即时响应的短期、高容量工作流，同步快速工作流是理想选择。该工作流可用于基于 Web 的应用程序或移动应用程序，此类应用程序的工作流通常持续时间较短，并且需要完成一系列步骤后才能返回响应。您可以直接从 Amazon API Gateway 触发同步快速工作流，在工作流完成或超时之前，连接将保持打开状态。对于不需要即时响应的短期工作流，Step Functions 还提供了异步快速工作流。

使用 Step Functions 的 API 编排示例包括：

同步或实时工作流

- 更改记录中的值，例如更新员工的姓氏，并立即在屏幕上显示更改。
- 在结账时更新订单，例如添加、移除或更改商品数量，然后立即将更新内容反馈给客户。
- 运行快速处理任务，并立即将结果返回给请求者。

容器编排

- 使用 Amazon Elastic Kubernetes Service 在 Kubernetes 上运行作业，或者使用 Fargate 在Amazon Elastic Container Service (ECS) 上运行作业，并与其他 AWS 服务集成，例如通过 Amazon SNS 发送通知，作为同一工作流的一部分。

IT 和安全自动化

IT 自动化可以帮助管理日益复杂和耗时的操作，例如升级和修补软件、部署安全更新以修复漏洞、选择基础架构、同步数据、路由支持票证等。重复耗时任务的自动化操作可以让您的组织快速一致地大规模完成例行操作。这样，您就可以在满足这些日益增长的需求的同时，专注于特征开发、复杂的支持请求和创新等战略性工作。

Step Functions 允许您创建自动扩展的工作流，满足您的业务需求，而无需人工干预。如果您的工作流中出现错误，通常不需要人工干预。Step Functions 可以自动重试失败的任务并通过指数回退来管理工作流中的错误。

在某些情况下，可能需要人工干预才能推进工作流。例如，批准大幅提高信用额度可能需要人工审批。要处理这种情况，您可以在 Step Functions 中定义分支逻辑，这样只有超过规定金额的申请才需要人工批准，而所有其他申请则会自动完成。在需要人工批准的情况下，Step Functions 允许您在特定步骤暂停工作流，等待响应，然后在收到响应后继续工作流。

客户使用 Step Functions 实现的自动化工作流类型示例：

IT 自动化

- 自动修复诸如打开 SSH 端口、磁盘空间不足或授予对 Amazon S3 存储桶的公开访问权限之类的事件。
- 自动部署 AWS CloudFormation StackSets

安全自动化

- 自动响应用户和用户访问密钥被泄露的场景。
- 根据定义的策略操作（如限制对特定 ARN 的操作或应用其他操作），自动修复安全事件响应。
- 在员工收到网络钓鱼电子邮件几秒钟内发出警告。

人工审批

- 自动训练机器学习模型，然后要求数据科学家对模型进行人工审批，然后根据收到的响应自动部署或拒绝模型。
- 根据情绪分析对收到的客户反馈进行自动路由，以便立即将具有负面情绪的反馈升级为人工审核。

Step Functions 的工作原理

本部分介绍帮助您熟悉 AWS Step Functions 并了解其工作原理的重要概念。

主题

- 标准和快速工作流
- 状态
- Map 状态处理模式
- 分布式 Map 状态容许的故障阈值
- 转换
- 状态机数据
- Step Functions 中的输入和输出处理
- 数据流模拟器
- 使用版本与别名功能管理持续部署
- Step Functions 中的执行
- Step Functions 中的错误处理
- 从其他服务调用 AWS Step Functions
- Step Functions 中的读取一致性
- Step Functions 中的标记

标准和快速工作流

创建状态机时，您必须选择标准或快速类型。状态机的默认类型为标准。类型为标准的状态机称为标准工作流，类型为快速的状态机被称为快速工作流。

对于标准工作流和快速工作流，您都可以使用 Amazon States Language 来定义状态机。根据您选择的类型，状态机的执行行为会有所不同。

> **Important**
>
> 创建状态机后，无法更改您选择的类型。

> **Note**
>
> 如果您在 Step Functions 控制台之外定义状态机，例如在自己选择的编辑器中，则必须以 .asl.json 扩展名保存状态机定义。

标准工作流非常适合长时间运行（最长一年）、持久且可审计的工作流。在执行完成后，您可以使用 Step Functions API 检索 90 天内的完整执行历史记录。标准工作流遵循仅一次 模型，在这种模型中，除非您在 ASL 中指定了 `Retry` 行为，否则任务和状态的运行次数永远不会超过一次。这使得标准工作流适合于编排非幂等操作，例如启动 Amazon EMR 集群或处理付款。标准工作流执行将根据处理的状态转换次数计费。

快速工作流是适用于大批量事件处理工作负载，例如 IoT 数据摄取、流数据处理和转换以及移动应用程序后端。它们最多可以运行五分钟。快速工作流采用至少一次 模型，在这种模式下，一次执行可能会运行多次。这使得快速工作流非常适合编排幂等操作，例如转换输入数据和通过 PUT 操作存储到 Amazon DynamoDB 中。快速工作流的执行按执行次数、执行持续时间和执行运行时消耗的内存计费。

标准和快速工作流可以自动启动以响应事件，例如来自 Amazon API Gateway 进行的 HTTP 请求（大规模完全托管的 API）、IoT 规则以及 Amazon EventBridge 中超过 140 个事件源。

> **Tip**
>
> 要将快速工作流的示例部署到 AWS 账户，请参阅《AWS Step Functions 研讨会》中的模块 7 - API Gateway、Parallel 状态、快速工作流。

有关标准和快速工作流执行的控制台体验的信息，请参阅控制台中的标准和快速工作流执行。

标准和快速工作流

	标准工作流	快速工作流程：同步和异步
最长持续时间	一年	五分钟
支持的执行启动率	有关与支持的执行启动率相关的配额信息，请参阅与 API 操作限制相关的配额。	有关与支持的执行启动率相关的配额信息，请参阅与 API 操作限制相关的配额。

	标准工作流	快速工作流程：同步和异步
支持的状态转换速率	有关与支持的状态转换速率相关的配额信息，请参阅与状态限制相关的配额。	无限制
定价	按状态转换次数定价。每完成一个执行步骤，就会计算一次状态转换。	按运行的执行次数、执行时长和内存消耗量来定价。
执行历史记录	可以使用 Step Functions API 列出和描述执行情况。您可通过控制台对执行进行可视化调试。您也可以通过在状态机上启用日志能在 CloudWatch Logs 中检查执行情况。 有关在控制台中调试标准工作流执行的更多信息，请参阅控制台中的标准和快速工作流执行和查看和调试执行。	无限制的执行历史记录，即在 5 分钟内生成多少条执行历史记录，就会保留多少条执行历史记录。 通过在状态机上启用日志记录功，可在 CloudWatch Logs 或 Step Functions 控制台中检查执行情况。 有关在控制台中调试快速工作流执行的更多信息，请参阅控制台中的标准和快速工作流执行和查看和调试执行。
执行语义	仅一次 工作流执行。	异步快速工作流：至少一次 工作流执行。 同步快速工作流程：至多一次工作流执行。

	标准工作流	快速工作流程：同步和异步
服务集成	支持所有服务集成和模式。	支持所有服务集成。 **Note** 快速工作流不支持任务运行 (`.sync`) 或回调 (`.waitForTaskToken`) 服务集成模式。
Step Functions 活动	支持 Step Functions 活动。	不支持 Step Functions 活动。

同步和异步快速工作流

您可以选择两种类型的快速工作流：异步快速工作流和同步快速工作流。

- 异步快速工作流会返回工作流已启动的确认信息，但不会等待工作流完成。要获得结果，您必须轮询该服务的 CloudWatch Logs。如果您不需要即时响应输出，例如其他服务不依赖的消息服务或数据处理，则可以使用异步快速工作流。您可以通过 Step Functions 中的嵌套工作流或使用 `StartExecution` API 调用来启动异步快速工作流，响应事件。
- 同步快速工作流会启动一个工作流，等待工作流完成，然后返回结果。同步快速工作流可用于编排微服务。使用同步快速工作流，您可以开发应用程序，而无需开发额外的代码来处理错误、重试或运行并行任务。您可以运行从 Amazon API Gateway 或 AWS Lambda 调用的同步快速工作流，也可以使用 `StartSyncExecution` API 调用来运行同步快速工作流。

> **Note**
>
> 如果您从控制台同步运行 Step Functions 快速工作流，则 `StartSyncExecution` 请求经过 60 秒。若要同步运行长达五分钟的快速工作流，请不要使用 Step Functions 控制台，而要使用 AWS 开发工具包或 AWS Command Line Interface (AWS CLI) 发出 `StartSyncExecution` 请求。

同步快速执行 API 调用不会影响现有的账户容量限制。Step Functions 按需提供容量，并根据持续的工作负载自动扩展。在容量扩展之前，可以限制工作负载激增。

执行保障

标准工作流	异步快速工作流	同步快速工作流	
仅一次 工作流执行	至少一次 工作流执行	至多一次 工作流执行	
在状态转换之间，执行状态在内部保持不变。	在状态转换之间，执行状态不会持续存在。	在状态转换之间，执行状态不会持续存在。	
在启动与当前运行的工作流同名的执行时，自动返回幂等响应。当前运行的工作流完成后，新工作流不会启动，并且会引发异常。	不会自动管理幂等性。启动多个同名工作流会导致并发执行。如果状态机逻辑不幂等，则可能导致内部工作流状态丢失。	不会自动管理幂等性。Step Functions 会在执行开始后等待，并在执行完成时返回状态机的结果。如果发生异常，工作流不会重新启动。	
执行历史数据会在 90 天后删除。删除过时的执行数据后，可以重复使用工作流名称。 为了满足合规性、组织或监管要求，您可以通过发送配额请求，将执行历史记录的保留期缩短至 30 天。为实现该操作，请使用 AWS Support Center Console并创建一个新案例。	Step Functions 不会捕获执行历史记录。必须通过 Amazon CloudWatch Logs 启用日志记录。	Step Functions 不会捕获执行历史记录。必须通过 Amazon CloudWatch Logs 启用日志记录。	

使用快速工作流实现成本优化

Step Functions 根据您用来构建状态机的工作流类型来确定标准和快速工作流的定价。要优化无服务器工作流的成本，您可以遵循以下一项或两项建议：

主题

- 提示 #1：在标准工作流中嵌套快速工作流
- 提示 #2：将标准工作流转换为快速工作流

有关选择标准或快速工作流类型如何影响账单的信息，请参阅AWS Step Functions 定价。

提示 #1：在标准工作流中嵌套快速工作流

Step Functions 运行的工作流具有有限的持续时间和步骤数量。有些工作流可能会在短时间内完成执行。另一些工作流可能需要将长时间运行的工作流和高事件率的工作流结合起来。借助 Step Functions，您可以利用多个更小、更简单的工作流来构建大型复杂的工作流。

例如，要构建订单处理工作流，您可以将所有非幂等操作包含到标准工作流中。这可能包括通过人际互动批准订单和处理付款等操作。然后，您可以在快速工作流中组合一系列幂等操作，例如发送付款通知和更新产品库存。您可以将此快速工作流嵌套在标准工作流中。在此示例中，标准工作流被称为父状态机。嵌套的快速工作流被称为子状态机。

提示 #2：将标准工作流转换为快速工作流

如果现有标准工作流满足以下要求，则可以将其转换为快速工作流：

- 工作流必须在五分钟内完成执行。
- 该工作流符合至少一次的执行模型。这意味着工作流中的每个步骤都可能运行不止一次。
- 工作流不使用 `.waitForTaskToken` 或 `.sync` 服务集成模式。

> **Important**
>
> 快速工作流使用 Amazon CloudWatch Logs 来记录执行历史记录。使用 CloudWatch Logs 时，将产生额外费用。

使用控制台将标准工作流转换为快速工作流

1. 打开 Step Functions 控制台。
2. 在状态机页面上，选择一个标准类型的状态机将其打开。

   > **Tip**
   >
   > 从任意类型下拉列表中，选择标准以筛选状态机列表，并仅查看标准工作流。

3. 选择复制到新项目。

   Workflow Studio 在设计模式下打开，显示所选状态机的工作流。
4. （可选）更新工作流设计。
5. 为状态机指定一个名称。要执行此操作，请选择默认状态机名称 MyStateMachine 旁边的编辑图标。然后，找到状态机配置，在状态机名称框中指定一个名称。
6. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   确保在类型中选择快速。保留状态机设置中的所有其他默认选项。

   > **Note**
   >
   > 如果您要转换之前在 AWS CDK 或 AWS SAM 中定义的标准工作流，则必须更改 `Type` 和 `Resource` 名称的值。

7. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

   > **Note**
   >
   > 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

有关管理工作流成本优化时的最佳实操和指南的更多信息，请参阅构建经济高效的 AWS Step Functions 工作流。

状态

单个状态可以根据其输入做出决定、执行操作，并将输出传递给其他状态。在 AWS Step Functions 中，您可以使用 Amazon States Language (ASL) 定义工作流。Step Functions 控制台提供了状态机的图形表示，以帮助可视化您的应用程序逻辑。

> **Note**
>
> 如果您在 Step Functions 控制台之外定义状态机，例如在自己选择的编辑器中，则必须以 .asl.json 扩展名保存状态机定义。

状态是状态机中的元素。状态通过其名称 来引用，这可以是任意字符串，但在整个状态机的范围内必须唯一。

在状态机中，状态可以执行各种函数：

- 在您的状态机中执行一些工作（Task 状态）
- 在执行分支之间进行选择（Choice 状态）
- 停止执行，返回失败或成功（Fail 或 Succeed 状态）
- 将其输入传递到其输出，或者向工作流注入一些固定数据（Pass 状态）
- 提供一定时间量的延迟或直至指定时间和日期（Wait 状态）
- 开始执行的并行分支（Parallel 状态）
- 动态迭代步骤（Map 状态）

以下是一个名为 `HelloWorld` 的执行 AWS Lambda 函数的示例状态。

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

状态有许多共同特征：

- `Type` 字段指示状态是什么类型。

- 可选的 Comment 字段以人类可读的格式存放有关状态的评论或说明。
- 每个状态 (除了 Succeed 或 Fail 状态) 需要一个 Next 字段，或者也可以通过指定 End 字段成为最终状态。

> **Note**
>
> 一个 Choice 状态可能会有多个 Next，但每个选项规则中只能有一个。Choice 状态无法使用 End。

某些状态类型需要额外的字段，也可能重新定义常见字段用法。

创建并运行标准流后，您可以通过在 Step Functions 控制台中查看执行详细信息页面，来访问有关各个状态、其输入和输出、何时活动以及活动时长的信息。有关更多信息，请参阅在 Step Functions 控制台上查看和调试执行。

创建并运行快速工作流执行后，如果快速工作流启用了日志记录，则可在Step Functions 控制台或 Amazon CloudWatch Logs 中访问有关执行的信息。有关更多信息，请参阅在 Step Functions 控制台上查看和调试执行。

主题

- Amazon States Language
- Pass
- 任务状态
- Choice
- Wait
- Succeed
- Fail
- Parallel
- Map

Amazon States Language

Amazon States Language 是一种基于 JSON 的结构化语言，用于定义状态机（一个状态集合），可以执行工作（Task 状态），确定哪些状态转换为下一个状态（Choice 状态），在出错的情况下停止执行（Fail 状态）等等。

有关更多信息，请参阅 Amazon States Language 规范和 Statelint，后者是一个用于验证 Amazon 状态语言代码的工具。

要使用 Amazon States Language 在 Step Functions 控制台上创建状态机，请参阅入门。

> **Note**
>
> 如果您在 Step Functions 控制台之外定义状态机，例如在自己选择的编辑器中，则必须以 .asl.json 扩展名保存状态机定义。

Amazon States Language 规范示例

```
{
  "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
    }
  }
}
```

主题

- 状态机结构
- 内置函数
- 常见状态字段

状态机结构

状态机使用 JSON 文本定义，该文本表示的结构包含以下字段。

Comment（可选）

一个人类可读格式的状态机描述。

StartAt（必填）

一个字符串，必须与某个状态对象的名称完全匹配（区分大小写）。

TimeoutSeconds（可选）。

状态机执行可运行的最大秒数。如果运行时间超过指定时间，则执行失败并返回 States.Timeout 错误名称。

Version（可选）

状态机中使用的 Amazon States Language 版本（默认为“1.0”）。

States（必填）

一个对象，其中包含逗号分隔的状态集合。

States 字段包含状态。

```
{
    "State1" : {
    },

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

状态机由其包含的状态以及状态之间的关系所定义。

以下是示例。

```
{
  "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
    }
  }
}
```

启动此状态机的执行时，系统从 StartAt 字段中引用的状态开始 ("HelloWorld")。如果此状态有一个 "End": true 字段，则执行停止并返回结果。否则，系统会查找 "Next": 字段并使用下一个

状态继续。此过程不断重复，直至系统达到最终状态 (`"Type": "Succeed"`、`"Type": "Fail"` 或 `"End": true` 状态) 或者出现运行时错误。

以下规则适用于状态机中的状态：

- 状态在封闭块中可以按任意顺序出现，但它们列出的顺序不会影响到它们的运行顺序。运行顺序由状态的内容决定。
- 在状态机中，只能有一个状态指定为 `start` 状态，由顶级结构中 `StartAt` 字段的值指定。此状态是在执行启动时首先执行的状态。
- `End` 字段为 `true` 的任意状态将被视为 `end`（或 `terminal`）状态。根据状态机逻辑（例如状态机有多个执行分支），您可能会有多个 `end` 状态。
- 如果状态机只包含一个状态，该状态可以为 `start` 状态和 `end` 状态。

内置函数

Amazon States Language 提供了多个内置函数 (intrinsic functions)，也称为 intrinsics，可帮助您在不使用 `Task` 状态的情况下执行基本的数据处理操作。内置函数是看起来与编程语言中的函数相似的结构。它们可以用来帮助有效载荷生成器处理进出 `Task` 状态 `Resource` 字段的数据。

在 Amazon States Language 中，根据您要执行的数据处理任务的类型，将内置函数分为以下类别：

- 数组的内置函数
- 用于数据编码和解码的内置函数
- 用于哈希计算的内置函数
- 用于 JSON 数据操作的内置函数
- 用于数学运算的内置函数
- 用于字符串操作的内置函数
- 用于生成唯一标识符的内置函数
- 用于泛型操作的内置函数

> **Note**
>
> - 要使用内置函数，必须在状态机定义的键值中指定 `.$`，如以下示例所示：
>
> ```
> "KeyId.$": "States.Array($.Id)"
> ```

- 在工作流的字段中，您最多可以嵌套 10 个内置函数。以下示例显示了一个名为 *myArn* 的字段，其中包含九个嵌套的内置函数：

```
"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**
>
> 如果您在本地开发环境中使用 Step Functions，请确保您使用的是 1.12.0 或更高版本，以便能够在工作流中包含所有内置函数。

支持内置函数的字段

下面显示了每个状态的哪些字段支持内置函数。

支持内置函数的字段

State	Pass	Task	Choice	Wait	Succeed	Fail	Parallel	Map
InputPath								
Parameters	✓	✓					✓	✓
ResultSelector		✓					✓	✓
ResultPath								
OutputPath								

State	Pass	Task	Choice	Wait	Succeed	Fail	Parallel	Map
Variable								
<Comparison Operator> Path								
TimeoutSecondsPath								
HeartbeatSecondsPath								
凭证		✓						

数组的内置函数

使用以下内置函数来执行数组操作。

`States.Array`

`States.Array` 内置函数采用零个或多个参数。解释器返回一个 JSON 数组，其中包含按提供的顺序排列的参数值。例如，给定以下输入：

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

您可以使用

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

返回以下响应：

```
“BuildId”: [123456]
```

States.ArrayPartition

使用 States.ArrayPartition 内置函数对大型数组进行分区。您也可以使用这个内置函数对数据进行切片，然后将有效负载分成较小的区块发送。

这个内置函数需要两个参数。第一个参数是一个数组，而第二个参数定义了区块大小。解释器将输入数组分成多个数组，其大小由区块大小指定。如果数组中剩余的项目数小于区块大小，则最后一个数组区块的长度可能小于之前的数组区块的长度。

输入验证

- 必须指定一个数组作为函数第一个参数的输入值。
- 必须为代表区块大小值的第二个参数指定一个非零的正整数。

  如果您为第二个参数指定了非整数值，Step Functions 会将其四舍五入为最接近的整数。
- 输入数组不能超过 Step Functions 的有效载荷大小限制，即 256 KB。

例如，给定以下输入数组：

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

您可以使用 States.ArrayPartition 函数将数组分成四个值的区块：

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

将返回以下数组区块：

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

在前面的示例中，States.ArrayPartition 函数输出了三个数组。前两个数组根据区块大小的定义，各包含四个值。第三个数组包含剩余的值，并且小于定义的区块大小。

States.ArrayContains

使用 States.ArrayContains 内置函数来确定数组中是否存在特定值。例如，您可以使用此函数来检测 Map 状态迭代中是否存在错误。

这个内置函数需要两个参数。第一个参数是数组，而第二个参数是要在数组中搜索的值。

输入验证

- 必须指定一个数组作为函数第一个参数的输入值。
- 您必须指定一个有效的 JSON 对象作为第二个参数。
- 输入数组不能超过 Step Functions 的有效载荷大小限制，即 256 KB。

例如，给定以下输入数组：

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

您可以使用 States.ArrayContains 函数在以下位置中 inputArray 查找 lookingFor 值：

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

由于 lookingFor 中存储的值包含在 inputArray 中，因此 States.ArrayContains 会返回以下结果：

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

States.ArrayRange

使用 States.ArrayRange 内置函数创建一个包含特定元素范围的新数组。新数组最多可以包含 1000 个元素。

这个函数需要三个参数。第一个参数是新数组的第一个元素，第二个参数是新数组的最后一个元素，第三个参数是新数组中元素之间的增量值。

输入验证

- 必须为所有参数指定整数值。

  如果您为任何参数指定了非整数值，Step Functions 会将其四舍五入为最接近的整数。
- 必须为第三个参数指定一个非零值。
- 新生成的数组不能包含超过 1000 个项目。

例如，以下使用 States.ArrayRange 函数将创建一个数组，其第一个值为 1，最后一个值为 9，并且第一个值和最后一个值之间的值，增量皆为二：

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

将返回以下数组：

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

States.ArrayGetItem

此内置函数返回指定索引的值。这个函数需要两个参数。第一个参数是值数组，第二个参数是要返回的值的数组索引。

例如，使用以下 inputArray 和 index 值：

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

根据这些值，您可以使用 States.ArrayGetItem 函数返回数组中 index 位置 5 的值：

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

在本示例中，States.ArrayGetItem 返回以下结果：

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

States.ArrayLength

States.ArrayLength 内置函数返回数组的长度。它有一个参数，即要返回长度的数组。

例如，给定以下输入数组：

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

您可以使用 States.ArrayLength 返回 inputArray 的长度：

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

在本示例中，States.ArrayLength 返回以下 JSON 对象，表示数组长度：

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

States.ArrayUnique

States.ArrayUnique 内置函数从数组中删除重复的值，并返回仅包含唯一元素的数组。该函数的唯一参数是一个数组，该数组可以不排序。

例如，以下 inputArray 包含一系列重复的值：

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

您可以使用 States.ArrayUnique 函数指定要从中删除重复值的数组：

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

States.ArrayUnique 函数将返回以下仅包含唯一元素的数组，删除所有重复的值：

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

用于数据编码和解码的内置函数

使用以下内置函数根据 Base64 编码方案对数据进行编码或解码。

States.Base64Encode

使用 States.Base64Encode 内置函数根据 MIME Base64 编码方案对数据进行编码。您可以使用此函数将数据传递给其他 AWS 服务，而无需使用 AWS Lambda 函数。

此函数将最多 1 万个字符的数据字符串作为其唯一参数进行编码。

例如，考虑以下 input 字符串：

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

您可以使用 States.Base64Encode 函数将 input 字符串编码为 MIME Base64 字符串：

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

States.Base64Encode 函数会返回以下编码数据：

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

States.Base64Decode

使用 States.Base64Decode 内置函数根据 MIME Base64 解码方案对数据进行解码。您可以使用此函数将数据传递给其他 AWS 服务，而无需使用 Lambda 函数。

此函数将最多 1 万个字符的 Base64 编码数据字符串作为其唯一参数进行解码。

例如，给定以下输入：

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

您可以使用 States.Base64Decode 函数将 base64 字符串解码为便于阅读的字符串：

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

States.Base64Decode function 会返回以下解码数据：

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

用于哈希计算的内置函数

States.Hash

使用 States.Hash 内置函数计算给定输入的哈希值。您可以使用此函数将数据传递给其他 AWS 服务，而无需使用 Lambda 函数。

这个函数需要两个参数。第一个参数是您要计算哈希值的数据。第二个参数是用于执行哈希计算的哈希算法。您提供的数据必须是包含 1 万个或更少字符的对象字符串。

您指定的哈希算法可以是以下任何算法：

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

- SHA-512

例如，您可以使用此函数，使用指定 Algorithm 来计算 Data 字符串的哈希值：

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

您可以使用 States.Hash 函数来计算哈希值：

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

States.Hash 函数会返回以下哈希值：

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

用于 JSON 数据操作的内置函数

使用这些函数对 JSON 对象执行基本的数据处理操作。

States.JsonMerge

使用 States.JsonMerge 内置函数将两个 JSON 对象合并为一个对象。这个函数需要三个参数。前两个参数是您要合并的 JSON 对象。第三个参数是布尔值 false。此布尔值决定是否启用深合并模式。

目前，Step Functions 仅支持浅合并模式；因此，必须将布尔值指定为 false。在浅模式下，如果两个 JSON 对象中存在相同的键，则后一个对象的键将覆盖第一个对象中的相同键。此外，当您使用浅合并时，嵌套在 JSON 对象中的对象不会合并。

例如，您可以使用 States.JsonMerge 函数合并以下两个相同键 a 的 JSON 对象。

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

您可以在 States.JsonMerge 函数中指定 json1 和 json2 对象作为输入，将它们合并在一起：

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

States.JsonMerge 返回以下合并的 JSON 对象结果。在合并的 JSON 对象 output 中，json2 对象的键 a 替换了 json1 对象的键 a。此外，json1 对象键 a 中的嵌套对象被丢弃，因为浅模式不支持合并嵌套对象。

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

States.StringToJson

States.StringToJson 函数将转义的 JSON 字符串的引用路径作为其唯一参数。

解释器应用 JSON 分析器并返回输入分析后的 JSON 表单。例如，您可以使用此函数来转义以下输入字符串：

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

使用 States.StringToJson 函数并指定 escapedJsonString 作为输入参数：

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

States.StringToJson 函数返回以下结果：

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

States.JsonToString

States.JsonToString 函数只接受一个参数，即包含要作为未转义字符串返回的 JSON 数据的路径。解释器返回一个字符串，其中包含表示路径指定数据的 JSON 文本。例如，您可以提供以下包含转义值的 JSON 路径：

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

为 States.JsonToString 函数提供 unescapedJson 中包含的数据：

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

States.JsonToString 函数会返回以下响应：

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

用于数学运算的内置函数

使用这些函数来执行数学运算。

States.MathRandom

使用 States.MathRandom 内置函数返回介于指定起始数字（包括）和结束数字（不包括）之间的随机数。

您可以使用此函数在两个或多个资源之间分配特定任务。

这个函数需要三个参数。第一个参数是起始数字，第二个参数是结束数字，最后一个参数控制种子值。种子值参数是可选的。如果您使用具有相同种子值的函数，它将返回相同的数字。

> **Important**
>
> 由于 States.MathRandom 函数不返回加密安全的随机数，因此我们建议您不要将其用于安全敏感型应用程序。

输入验证

- 必须为起始数字和结束数字参数指定整数值。

如果您为起始数字或结束数字参数指定非整数值，Step Functions 会将其四舍五入为最接近的整数。

例如，要生成介于 1 到 999 之间的随机数，可以使用以下输入值：

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

要生成随机数，请为 States.MathRandom 函数提供 start 和 end 值：

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

States.MathRandom 函数返回以下随机数：

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

States.MathAdd

使用 States.MathAdd 内置函数返回两个数字的总和。例如，您可以使用此函数在循环内递增值，而无需调用 Lambda 函数。

输入验证

- 必须为所有参数指定整数值。

  如果您为一个或两个参数指定了非整数值，Step Functions 会将其四舍五入为最接近的整数。
- 必须指定介于 -2147483648 和 2147483647 范围内的整数值。

例如，您可以使用以下值从 111 中减去 1：

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

然后，使用 States.MathAdd 函数，定义 value1 为起始值，step 为 value1 的增量值：

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

States.MathAdd 函数将在响应中返回下面的数字。

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

用于字符串操作的内置函数

States.StringSplit

使用 States.StringSplit 内置函数将字符串拆分成一个值数组。这个函数需要两个参数。第一个参数是一个字符串，第二个参数是该函数用来拆分字符串的分隔字符。

Example - 使用单个分隔字符拆分输入字符串

在本示例中，使用 States.StringSplit 对下方的 inputString 进行划分，其中包含一系列以逗号分隔的值：

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

使用 States.StringSplit 函数并将inputString 定义为第一个参数，将分隔字符 splitter 用作第二个参数：

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

States.StringSplit 函数返回以下字符串数组作为结果：

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

Example - 使用多个分隔字符拆分输入字符串

在本示例中，使用 States.StringSplit 来划分下方的 inputString，其中包含多个分隔字符：

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

按如下方式使用 States.StringSplit 函数：

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

States.StringSplit 函数返回以下字符串数组作为结果：

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

用于生成唯一标识符的内置函数

States.UUID

使用 States.UUID 内置函数返回使用随机数生成的版本 4 通用唯一标识符 (v4 UUID)。例如，您可以使用此函数调用其他需要 UUID 参数的 AWS 服务或资源，或者在 DynamoDB 表中插入项目。

调用 States.UUID 函数时不指定任何参数：

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

该函数返回一个随机生成的 UUID，如下例所示：

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

用于泛型操作的内置函数

States.Format

使用 States.Format 内置函数根据文本值和插值构造字符串。此函数采用一个或多个参数。第一个参数的值必须是字符串，并且可以包含字符序列 {} 的零个或多个实例。内置函数的调用中，其余参数的个数必须与出现的 {} 个数一样多。解释器会返回第一个参数中定义的字符串，每个 {} 都会被内置函数调用中位置对应的参数值替换。

例如，您可以使用以下输入，输入一个人的 name，并在 template 句子中插入其姓名：

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

使用 States.Format 函数，指定 template 字符串和要插入以代替 {} 字符的字符串：

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

或

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

对于前面的任何一个输入，States.Format 函数都会返回已完成的字符串：

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

内置函数中的预留字符

以下字符是内置函数的预留字符，必须使用反斜杠 ('\') 转义，才会出现在值中：'{} 和 \。

如果字符 \ 需要作为值的一部分出现而不用作转义字符，则必须使用反斜杠对其进行转义。以下转义字符序列与内置函数一起使用：

- 文字字符串 \' 表示 '。
- 文字字符串 \{ 表示 {。
- 文字字符串 \} 表示 }。
- 文字字符串 \\ 表示 \。

在 JSON 中，字符串文字值中包含的反斜杠必须使用另一个反斜杠进行转义。JSON 的等效列表是：

- 转义后的字符串 \\\' 表示 \'。
- 转义后的字符串 \\\{ 表示 \{。
- 转义后的字符串 \\\} 表示 \}。
- 转义后的字符串 \\\\ 表示 \\。

> **Note**
>
> 如果在内置函数调用字符串中发现了开放的转义反斜杠 \，则解释器将返回运行时错误。

常见状态字段

Type（必填）

状态的类型。

Next

当前状态完成时将运行的下一个状态的名称。某些状态类型 (例如 Choice) 允许多个转换状态。

如果当前状态是工作流中的最后一个状态或终端状态（例如 Succeed 或 Fail），则无需指定 Next 字段。

End

在设置为 true 时，指定此状态作为终端状态（结束执行）。每个状态机可以有任意数量的终端状态。在一个状态中只能使用一个 Next 或 End。某些状态类型（例如 Choice）或终端状态（例如 Succeed 和 Fail ）不支持或使用 End 字段。

Comment（可选）

保存状态的人类可读格式的描述。

InputPath（可选）

一个路径，用于选择要传递到状态任务进行处理的状态输入的一部分。如果省略，它的值为 $，表示指定整个输入。有关更多信息，请参阅输入和输出处理。

OutputPath（可选）

一个路径，用于选择要传递到下一个状态的状态输出的一部分。如果省略，它的值为 $，表示指定整个输出。有关更多信息，请参阅输入和输出处理。

Pass

Pass 状态 ("Type": "Pass") 将其输入传递到其输出，不执行任何工作。Pass 状态在构造和调试状态机时非常有用。

您还可以使用 Pass 状态通过筛选条件转换 JSON 状态输入，然后将转换后的数据传递到工作流中的下一个状态。有关输入转换的信息，请参阅 InputPath、参数和 ResultSelector。

除了常用状态字段之外，Pass 状态还允许以下字段。

Result（可选）

指传递到下一个状态的虚拟任务的输出。如果您在状态机定义中包含 ResultPath 字段，Result 则按 ResultPath 指定的方式放置，并传递到下一个状态。

ResultPath（可选）

指定用于放置 Result 中指定虚拟任务的输出（相对于输入）的位置。输入将进一步按照 OutputPath 字段（如果存在）指定的内容进行筛选，然后再用作状态输出。有关更多信息，请参阅输入和输出处理。

Parameters（可选）

创建将作为输入传递的键值对集合。您可以指定 Parameters 为静态值，也可以使用路径从输入中进行选择。有关更多信息，请参阅 InputPath、参数和 ResultSelector。

传递状态示例

以下示例说明的是 Pass 状态，该状态将一些固定数据注入到状态机中，可用于测试用途。

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

假设此状态的输入为以下内容。

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

然后，输出将为：

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

任务状态

Task 状态 ("Type": "Task") 代表状态机执行的一个工作单元。任务通过使用活动或函数、与其他支持的活动或 AWS Lambda 函数集成 AWS 服务，或者通过调用第三方 API（例如 Stripe）来执行工作。

Amazon States Language 通过将状态类型设置为 Task 并且向任务提供活动、Lambda 函数或第三方 API 端点的 Amazon 资源名称 (ARN) 来呈现任务。以下 Task 状态定义调用名为 *HelloFunction* 的 Lambda 函数。

```
"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
}
```

本主题内容

- 任务类型
- Task 状态字段
- Task 状态定义示例
- 活动

任务类型

Step Functions 支持以下任务类型，您可以在 Task 状态定义中指定这些类型：

- 活动
- Lambda 函数
- A 支持的 AWS 服务
- HTTP 任务

您可以通过在 Task 状态定义的 `Resource` 字段中提供其 ARN 来指定任务类型。以下示例显示了 `Resource` 字段的语法。除调用第三方 API 的任务类型外，所有任务类型均使用以下语法。有关 HTTP 任务语法的信息，请参阅调用第三方 API。

在任务状态定义中，将以下语法中的斜体文本替换为 AWS 资源特定的信息。

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

下面的列表解释了此语法中的各个组件：

- `partition`是最常用的 AWS Step Functions 分区`aws`。
- `service`表示 AWS 服务 用于执行任务的，可以是以下值之一：
  - `states` 表示活动。
  - `lambda` 表示 Lambda 函数。如果您与其他 AWS 服务（例如 Amazon SNS 或 Amazon DynamoDB）集成，请使用或。`sns` `dynamodb`
- `region`是在其中创建 Step Functions 活动或状态机类型、Lambda 函数或任何其他 AWS 资源的AWS 区域代码。
- `account`是您定义资源时使用的 AWS 账户 ID。
- `task_type` 是要运行的任务的类型。它可能为下列值之一：
  - `activity` – 一个活动。
  - `function` – 一个 Lambda 函数。
  - *`servicename`* – 支持的连接服务的名称（请参阅 Step Functions 的优化集成）。
- `name` 是已注册的资源名称（活动名称、Lambda 函数名称或服务 API 操作）。

> **Note**
>
> Step Functions 不支持跨分区或区域引用 ARN。例如，`aws-cn` 无法调用 `aws` 分区中的任务，反之亦然。

以下各部分提供有关每种类型的更多详细信息。

活动

活动表示由您实现和托管的、用于执行某个特定任务的工作线程 (进程或线程)。它们只受标准工作流支持，而不受快速工作流支持。

活动 `Resource` ARN 使用下面的语法。

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

> **Note**
>
> 在首次使用 Step Functions（使用CreateActivity、API 操作或 Step Functions 控制台）之前，您必须使用 Step Fun ctions 创建活动。

有关创建活动和实现工作线程的更多信息，请参阅活动。

Lambda 函数

Lambda 任务使用执行函数。 AWS Lambda要指定 Lambda 函数，请在 `Resource` 字段中使用 Lambda 函数的 ARN。

根据您用于指定 Lambda 函数的集成类型（优化集成或 AWS 软件开发工具包集成），Lambda 函数 `Resource` 字段的语法会有所不同。

以下 `Resource` 字段语法是 Lambda 函数的优化集成的示例。

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

以下`Resource`字段语法是 AWS 软件开发工具包与 Lambda 函数集成的示例。

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

以下 `Task` 状态定义显示了名为 *`HelloWorld`* 的 Lambda 函数的优化集成的示例。

```
"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"
}
```

在Resource字段中指定的 Lambda 函数完成后，其输出将发送到Next字段中标识的状态 (" NextState ")。

A 支持的 AWS 服务

当您引用连接的资源时，Step Functions 直接调用受支持服务的 API 操作。在 Resource 字段中指定服务和操作。

所连接服务 Resource ARN 使用下面的语法。

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

> **Note**
>
> 要创建与所连接资源的同步连接，请将 .sync 追加到 ARN 中的 *APIname* 条目。有关更多信息，请参阅 使用其他服务。

例如：

```
{
 "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
    }
  }
}
```

Task 状态字段

除了常见状态字段之外，Task 状态还具有以下字段。

`Resource`（必填）

一个 URI，尤其是唯一标识要执行的特定任务的 ARN。

`Parameters`（可选）

用于将信息传递给所连接资源的 API 操作。这些参数可以混合使用静态 JSON 和JsonPath。有关更多信息，请参阅 将参数传递给服务 API。

`Credentials`（可选）

指定状态机的执行角色在调用指定 `Resource` 前必须承担的目标角色。或者，您也可以指定 JSONPath 值或内置函数，在运行时根据执行输入解析为 IAM 角色 ARN。如果指定 JSONPath 值，则必须在其前缀加上`$.`表示法。

有关在 Task 状态内使用此字段的示例，请参阅 Task 状态的“凭证”字段示例。有关使用此字段从状态机访问跨账户 AWS 资源的示例，请参阅教程：访问跨账户资源 AWS。

> **Note**
>
> 使用 Lambda 函数任务类型的和支持的服务支持 AWS此字段。

`ResultPath`（可选）

指定（输入中）用于放置 `Resource` 中所指定任务的执行结果的位置。接下来，输入将按照 `OutputPath` 字段（如果存在）指定的内容进行筛选，然后再用作状态输出。有关更多信息，请参阅输入和输出处理。

`ResultSelector`（可选）

传递键值对集合，其中，值为静态值或从结果中选择的值。有关更多信息，请参阅 ResultSelector。

Retry（可选）

一个称为重试器的对象的数组，定义在状态遇到运行时错误时的重试策略。有关更多信息，请参阅使用 Retry 和使用 Catch 的状态机示例。

Catch（可选）

一个称为捕获器的对象数组，用于定义回退状态。如果状态遇到运行时错误并且其重试策略已耗尽或者未定义，则执行该状态。有关更多信息，请参阅回退状态。

TimeoutSeconds（可选）

指定活动或任务在因 States.Timeout 错误而超时并失败之前可以运行的最长时间。超时值必须是非零的正整数。默认值为 99999999。

超时计数从任务启动后开始，例如，从 `ActivityStarted` 或 `LambdaFunctionStarted` 事件记录到执行事件历史记录时开始。对于活动，当 `GetActivityTask` 收到令牌并且 `ActivityStarted` 记录在执行事件历史记录中时开始计数。

Step Functions 会在指定的 `TimeoutSeconds` 时长内等待任务或活动工作线程的成功或失败响应。如果任务或活动工作线程未能在这段时间内做出响应，Step Functions 会将工作流执行标记为失败。

TimeoutSecondsPath（可选）

如果要使用参考路径从状态输入中动态提供超时值，请使用 `TimeoutSecondsPath`。解析后，参考路径必须选择值为正整数的字段。

> **Note**
>
> 一个 `Task` 状态不能同时包含 `TimeoutSeconds` 和 `TimeoutSecondsPath`。

HeartbeatSeconds（可选）

确定活动工作线程在任务执行期间发送的检测信号的频率。检测信号表示任务仍在运行，需要更多时间才能完成。检测信号可防止活动或任务在 `TimeoutSeconds` 持续时间内超时。

`HeartbeatSeconds` 必须是小于 `TimeoutSeconds` 字段值的正非零整数值。默认值为 99999999。如果任务检测信号之间的时间超过了指定秒数，则 Task 状态将失败，并返回 States.Timeout 错误。

对于活动，当 GetActivityTask 收到令牌并且 ActivityStarted 记录在执行事件历史记录中时开始计数。

HeartbeatSecondsPath（可选）

如果要使用参考路径从状态输入中动态提供检测信号值，请使用 HeartbeatSecondsPath。解析后，参考路径必须选择值为正整数的字段。

> **Note**
>
> 一个 Task 状态不能同时包含 HeartbeatSeconds 和 HeartbeatSecondsPath。

Task 状态必须将 End 字段设置为 true（如果状态结束执行），或者必须在 Next 字段中提供一个状态（该状态将在 Task 状态完成时运行）。

Task 状态定义示例

以下示例说明如何根据要求指定 Task 状态定义。

- 指定 Task 状态超时和检测信号间隔
  - 静态超时和检测信号通知示例
  - 动态任务超时和检测信号通知示例
- 使用“凭证”字段
  - 指定硬编码的 IAM 角色 ARN
  - 指定 JSONPath 为 IAM 角色 ARN
  - 指定一个内置函数为 IAM 角色 ARN

Task 状态超时和检测信号间隔

为长时间运行的活动设置超时值和检测信号间隔是一个好的做法。这可以通过指定超时和检测信号值或通过动态设置来完成。

静态超时和检测信号通知示例

当 HelloWorld 完成后，将运行下一个状态（此处名为 NextState）。

如果此任务在 300 秒内未能完成，或者未在 60 秒的间隔内发送检测信号通知，则任务会被标记为 failed。

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

动态任务超时和检测信号通知示例

在此示例中，当 AWS Glue 作业完成时，下一个状态将运行。

如果此任务未能在 AWS Glue 作业动态设置的时间间隔内完成，则该任务将被标记为 `failed`。

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

  "Next": "NextState"
}
```

Task 状态的“凭证”字段示例

指定硬编码的 IAM 角色 ARN

以下示例指定了一个目标 IAM 角色，状态机的执行角色必须担任该角色才能访问名为 `Echo` 的跨账户 Lambda 函数。在此示例中，目标角色 ARN 被指定为一个硬编码值。

```
{
  "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
    }
  }
}
```

指定 JSONPath 为 IAM 角色 ARN

以下示例指定了一个 JSONPath 值，该值将在运行时解析为 IAM 角色 ARN。

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

指定一个内置函数为 IAM 角色 ARN

以下示例使用 States.Format 内置函数，该函数在运行时解析为 IAM 角色 ARN。

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

```
}
```

活动

活动是 AWS Step Functions 的一项特征，它使您能够在状态机中设置任务，由工作线程执行，工作线程可以托管在 Amazon Elastic Compute Cloud (Amazon EC2)、Amazon Elastic Container Service (Amazon ECS)、移动设备上（基本上可以在任何地方托管）。

概述

在 AWS Step Functions 中，可以将正在某处运行的代码（称为活动工作线程）与状态机中的特定任务关联起来。您可以在 Step Functions 控制台中或通过调用 `CreateActivity` 创建一个活动。这样会为任务状态提供 Amazon 资源名称 (ARN)。使用此 ARN 可以轮询活动工作线程中的工作的任务状态。

> **Note**
>
> 活动没有版本控制，并应向后兼容。如果您必须对活动执行无法向后兼容的更改，则应通过 Step Functions 使用唯一名称创建一个新 活动。

活动工作线程可以是在 Amazon EC2 实例上运行的应用程序、AWS Lambda 函数、移动设备：任何可以进行 HTTP 连接、在任何位置托管的应用程序。当 Step Functions 达到某种活动任务状态，工作流程就等待活动工作线程轮询任务。活动工作线程通过使用 `GetActivityTask` 并发送相关活动的 ARN 来轮询 Step Functions，`GetActivityTask` 返回响应，其中包括 `input`（任务的 JSON 字符串输入）和 `taskToken`（任务的唯一标识符）。活动工作线程完成其工作后，可以使用 `SendTaskSuccess` 或 `SendTaskFailure` 提供成功或失败报告。这两个调用使用 `GetActivityTask` 提供的 `taskToken` 将结果与该任务关联起来。

与活动任务相关的 API

Step Functions 提供用于创建和列出活动、请求任务和根据工作线程结果管理状态机流程的 API。

下面是与活动相关的 Step Functions API：

- `CreateActivity`
- `GetActivityTask`
- `ListActivities`
- `SendTaskFailure`

- SendTaskHeartbeat
- SendTaskSuccess

> **Note**
>
> 在某些实现中，通过 GetActivityTask 轮询活动任务可能会导致延迟。请参阅避免轮询活动任务时发生延迟。

等待完成活动任务

通过在任务定义中设置 TimeoutSeconds 配置状态等待时长 要使任务保持为活动和等待状态，请在 TimeoutSeconds 中配置的时间内，定期使用 SendTaskHeartbeat 从活动工作线程发送检测信号。通过配置较长的超时时间和积极发送检测信号，Step Functions 中的活动最长可以等待一年时间以完成执行。

例如，如果需要工作流程等待长时间进程的结果，请执行以下操作：

1. 使用控制台或者使用 CreateActivity 创建活动。记下活动 ARN。
2. 在状态机定义中的活动任务状态中引用该 ARN 并设置 TimeoutSeconds。
3. 使用 GetActivityTask 并引用该活动 ARN 实现用于轮询工作的活动工作线程。
4. 在状态机任务定义的 HeartbeatSeconds 所设置的时间内，定期使用 SendTaskHeartbeat，以防止任务超时。
5. 启动状态机执行。
6. 启动活动工作进程。

执行在相应活动任务状态时暂停，等待活动工作线程轮询任务。一旦 taskToken 提供给活动工作线程，工作流程将等待 SendTaskSuccess 或 SendTaskFailure 提供状态。如果执行在 TimeoutSeconds 中配置的时间之前未收到上述任一状态或 SendTaskHeartbeat 调用，则执行将失败，执行历史记录将包含 ExecutionTimedOut 事件。

后续步骤

要更详细地了解如何创建使用活动工作线程的状态机，请参阅：

- 使用 Step Functions 创建活动状态机

- 使用 Ruby 编写的示例活动工作线程

使用 Ruby 编写的示例活动工作线程

下面是使用AWS SDK for Ruby的示例活动工作线程，介绍如何使用最佳实操实现您自己的活动工作线程。

这段代码实现了消费者/生产者模型，并且允许对轮询器和活动工作线程的线程数进行配置。轮询器线程不断长轮询活动任务。一旦检索到活动任务，则将其传入一个有界的阻塞队列，供活动线程领取任务。

- 有关使用 AWS SDK for Ruby 的更多信息，请参阅 AWS SDK for Ruby API reference。
- 要下载此代码及相关资源，请参阅 GitHub 上的 step-functions-ruby-activity-worker 存储库。

以下 Ruby 代码是此示例 Ruby 活动工作线程的主入口点。

```
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
```

这段代码包含默认设置，您可以更改为引用您的活动并使其适应您的特定实现。这段代码将实际实现逻辑作为输入，允许您引用特定的活动和凭证，并允许您配置线程数量和检测信号延迟。要获取更多信息和下载代码，请参阅 Step Functions Ruby 活动工作线程。

项目	描述
require_relative	下面示例活动工作线程代码的相对路径。
region	活动的 AWS 区域。
workers_count	活动工作线程的线程数。对于大多数实现来说，10 到 20 个线程就足够了。活动的处理时间越长，可能需要的线程就越多。您可以使用以下方式进行估算：将每秒处理活动数乘以第 99 个百分点的活动处理延迟 (以秒为单位)。
pollers_count	轮询器的线程数。对于大多数实现来说，10 到 20 个线程就足够了。
heartbeat_delay	检测信号间的延迟 (以秒为单位)。
input	活动的实现逻辑。

下面是 Ruby 活动工作线程，在代码中由 ../lib/step_functions/activity 引用。

```
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
```

Choice

Choice 状态 ("Type": "Choice") 向状态机添加条件逻辑。

除了大多数常用状态字段之外，Choice 状态还包含以下额外字段。

Choices（必填）

选项规则数组，确定状态机接下来转换为什么状态。在选择规则中使用比较运算符可将输入变量与特定值进行比较。例如，使用选择规则，您可以比较输入变量是大于还是小于 100。

运行 Choice 状态时，它会将每个选择规则评估为 true 或 false。根据评估结果，Step Functions 会转换到工作流中的下一个状态。

您必须在 Choice 状态中至少定义一条规则。

Default（可选，建议）

在 Choices 中没有进行任何转换时，要转换为的状态的名称。

> **Important**
>
> Choice 状态不支持 End 字段。此外，它们仅在其 Choices 字段内部使用 Next。

> **Tip**
>
> 要在您的 AWS 账户中部署使用 Choice 状态的工作流示例，请参阅《AWS Step Functions 研讨会》中的模块 5 - Choice 状态和 Map 状态。

选项规则

一个 Choice 状态必须有一个值为非空数组的 Choices 字段。此数组中的每个元素都是一个名为“选项规则”的对象，其中包含以下内容：

- 比较 – 两个字段，指定要比较的输入变量、比较的类型以及与变量进行比较的值。选项规则支持两个变量之间的比较。在选项规则中，通过在支持的比较运算符名称后附加 Path，可以将变量值与状态输入中的另一个值进行比较。比较中的 Variable 和路径字段的值必须是有效的参考路径。
- Next 字段 – 此字段的值必须与状态机中的状态名称匹配。

以下示例检查数值是否等于 1。

```
{
```

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

以下示例检查字符串是否等于 MyString。

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

以下示例检查字符串是否大于 MyStringABC。

```
{
  "Variable": "$.foo",
  "StringGreaterThan": "MyStringABC",
  "Next": "FirstMatchState"
}
```

以下示例检查字符串是否为空。

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

以下示例显示了由于前面的 IsPresent 选项规则，StringEquals 规则仅 $.keyThatMightNotExist 存在时才会被评估。

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

以下示例检查带有通配符的模式是否匹配。

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

以下示例检查时间戳是否等于 2001-01-01T12:00:00Z。

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

以下示例将变量与状态输入中的另一个值进行比较。

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

Step Functions 按照 Choices 字段中列出的顺序检查每个选项规则。然后，根据比较运算符确定变量与值匹配的第一条选项规则，并转换为该规则中的 Next 字段指定的状态。

支持下列比较运算符：

- 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

对于每个上述运算符，对应值必须为合适的类型：字符串、数字、布尔值或时间戳。Step Functions 不尝试将数值字段与字符串值匹配。但是，由于时间戳字段在逻辑上是字符串，视为时间戳的字段可能会由 StringEquals 运算符匹配。

> **Note**
>
> 为了实现互操作性，请不要假定数值比较可用于超出 IEEE 754-2008 binary64 数据类型所提供量级或精度的值。特别是，超出范围 $[-2^{53}+1, 2^{53}-1]$ 的整数可能会无法按预期方式比较。
> 时间戳 (例如，2016-08-18T17:33:00Z) 必须遵从 RFC3339 配置文件 ISO 8601，此外还有更多限制：
>
> - 必须使用大写 T 分隔日期和时间部分。
> - 大写 Z 必须表示没有数字时区偏移。
>
> 要了解字符串比较的行为，请参阅 Java compareTo 文档。

And 和 Or 运算符的值必须为非空的选项规则数组，自身不能包含 Next 字段。与此类似，Not 运算符的值必须为单个选项规则，不能包含 Next 字段。
您可以使用 And、Not 和 Or 创建复杂的嵌套选项规则。但是，Next 字段只能显示在顶级选项规则中。
使用 StringMatches 比较运算符可以对带有一个或多个通配符（“*”）的模式进行字符串比较。通配符使用标准的 \\（Ex：“*”）对进行转义。在匹配过程中，除了“*”以外，其他字符没有任何特殊含义。

选择状态示例

以下示例说明的是 Choice 状态及其转换为的其他状态。

Note

您必须指定 $.type 字段。如果状态输入不包含 $.type 字段，则执行失败，执行历史记录中显示错误。您只能在 StringEquals 字段中指定与文本值相匹配的字符串。例如，"StringEquals": "Buy"。

```
"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!"
}
```

在此示例中，状态机使用以下输入值启动。

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

Step Functions 基于 `value` 字段转换为 `ValueInTwenties` 状态。

如果 Choice 状态的 Choices 没有匹配，将改为运行 Default 字段中提供的状态。如果未指定 Default 状态，执行将失败，出现错误。

Wait

Wait 状态 ("Type": "Wait") 使状态机延迟指定的时间，然后再继续。您可以选择一个相对时间，指定从状态开始后的秒数；也可以选择以时间戳的方式指定绝对结束时间。

除了常用状态字段之外，Wait 状态还具有以下字段之一。

Seconds

启动 Next 字段中指定状态之前等待的时间，以秒为单位。时间必须指定为介于 0 到 99999999 之间的正整数值。

Timestamp

一个绝对时间，等到该时间后再启动 Next 字段中指定的状态。

时间戳必须遵循 ISO 8601 的 RFC3339 配置文件，并有进一步的限制，需要使用大写 T 分隔日期和时间部分，大写 Z 必须表示没有数字时区偏移，例如 2024-08-18T17:33:00Z。

> **Note**
>
> 当前，如果您将等待时间指定为时间戳，Step Functions 会选取秒以内的时间值并截断毫秒。

SecondsPath

启动在 Next 字段中指定的状态所等待的时间，以秒为单位，使用状态输入数据中的路径指定。

必须为此字段指定一个整数值。

TimestampPath

一个绝对时间，等到该时间后再启动在 Next 字段中指定的状态，使用状态输入数据中的路径指定。

> **Note**
>
> 您必须且只能指定以下之一：Seconds、Timestamp、SecondsPath 或 TimestampPath。此外，您可以为标准工作流和快速工作流指定的最长等待时间分别为一年和五分钟。

Wait 状态示例

以下 Wait 状态向状态机中引入 10 秒的延迟。

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

在下一个示例中，Wait 状态等待直至绝对时间：2024 年 3 月 14 日凌晨 1:59 (UTC 时间)。

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

您不必对等待持续时间进行硬编码。例如，给定以下输入数据：

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

您可以使用引用路径从输入中选择“expirydate”值，以便从输入数据选择该值。

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

Succeed

Succeed 状态 ("Type": "Succeed") 可成功停止执行。Succeed 状态是 Choice 状态分支的一个非常有用的目标，不执行任何操作，只是停止执行。

由于 Succeed 状态是终端状态，它们没有任何 Next 字段，也不需要 End 字段，如以下示例所示。

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

Fail

Fail 状态 ("Type": "Fail") 将停止状态机的执行并将其标记为故障，除非被 Catch 块捕获。。

Fail 状态仅允许使用常用状态字段集合中的 Type 和 Comment 字段。此外，Fail 状态允许以下字段。

Cause（可选）

描述错误原因的自定义字符串。您可以指定此字段用于操作或诊断目的。

CausePath（可选）

如果要使用参考路径从状态输入中动态提供有关错误原因的详细描述，请使用 CausePath。解决后，参考路径必须选择包含字符串值的字段。

您也可以使用返回字符串的内置函数来指定 CausePath。这些内置函数包括：States.Format、States.JsonToString、States.ArrayGetItem、States.Base64Encode、States.Base64 和 States.UUID.

> **Important**
>
> - 您可以在 Fail 状态定义中指定 Cause 或 CausePath，但二者不能同时指定。
> - 作为信息安全最佳实操，我们建议您从原因描述中删除任何敏感信息或内部系统详细信息。

Error（可选）

使用 Retry 或 Catch 字段执行错误处理时可以提供的错误名称。您也可以提供错误名称用于操作或诊断目的。

ErrorPath（可选）

如果要使用参考路径从状态输入中动态提供错误的名称，请使用 ErrorPath。解决后，参考路径必须选择包含字符串值的字段。

您也可以使用返回字符串的内置函数来指定 ErrorPath。这些内置函数包括：States.Format、States.JsonToString、States.ArrayGetItem、States.Base64Encode、States.Base64 和 States.UUID.

> **Important**
>
> - 您可以在 Fail 状态定义中指定 Error 或 ErrorPath，但二者不能同时指定。
> - 作为信息安全最佳实操，我们建议您从错误名称中删除任何敏感信息或内部系统详细信息。

由于 Fail 状态始终会退出状态机，它们没有 Next 字段，也不需要 End 字段。

Fail 状态定义示例

以下 Fail 状态定义示例指定了静态 Error 和 Cause 字段值。

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

以下 Fail 状态定义示例使用参考路径动态解析 Error 和 Cause 字段值。

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

以下 Fail 状态定义示例使用 States.Format 内置函数动态指定 Error 和 Cause 字段值。

```
"FailState": {
```

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

Parallel

`Parallel` 状态 (`"Type": "Parallel"`) 可用于在状态机中添加单独的执行分支。

除了常用状态字段之外，`Parallel` 状态还包括这些额外字段。

`Branches`（必填）

一个对象数组，指定要并行执行的状态机。每个此类状态机对象都必须具有名为 `States` 和 `StartAt` 的字段，其含义与在状态机顶级中的字段完全相同。

`ResultPath`（可选）

指定 (输入中) 用于放置分支输出的位置。接下来，输入将按照 `OutputPath` 字段（如果存在）指定的内容进行筛选，然后再用作状态输出。有关更多信息，请参阅输入和输出处理。

`ResultSelector`（可选）

传递键值对集合，其中，值为静态值或从结果中选择的值。有关更多信息，请参阅ResultSelector。

`Retry`（可选）

一个称为重试器的对象的数组，定义在状态遇到运行时错误的情况下的重试策略。有关更多信息，请参阅使用 Retry 和使用 Catch 的状态机示例。

`Catch`（可选）

一个称为捕获器的对象数组，定义在状态遇到运行时错误并且其重试策略已耗尽或者未定义的情况下执行的回退状态。有关更多信息，请参阅回退状态。

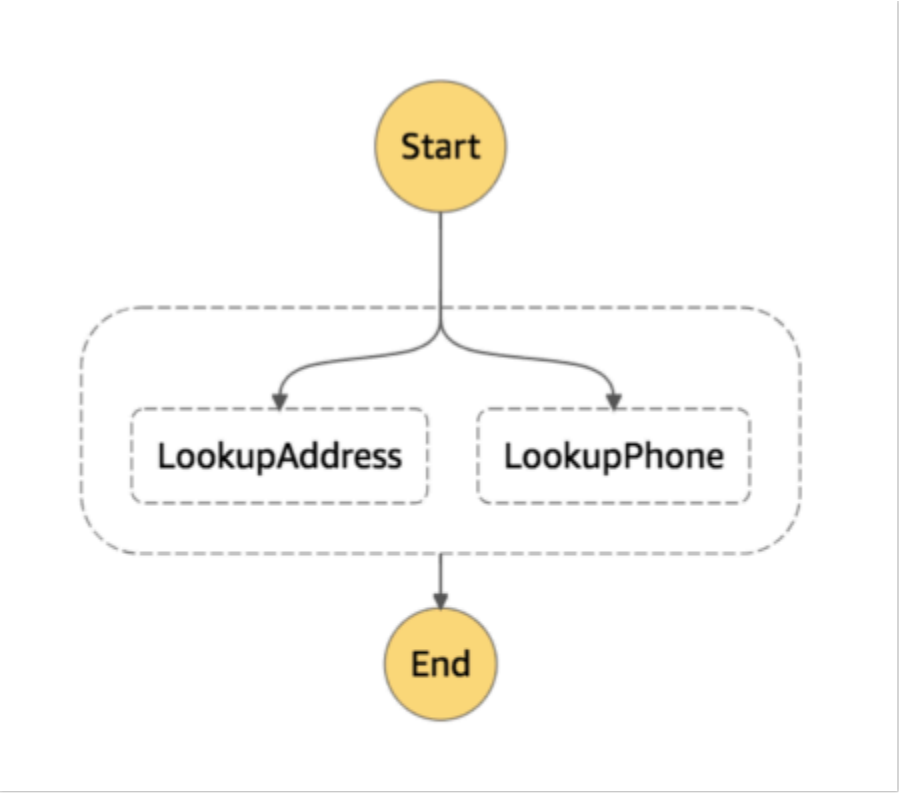
`Parallel`状态会 AWS Step Functions 导致尽可能并发地执行每个分支，从该分支的`StartAt`字段中命名的状态开始，然后等到所有分支终止（达到终端状态）后再处理该`Parallel`状态的`Next`字段。

并行状态示例

```
{
```

```
  "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
           }
         }
       }
      ]
    }
  }
}
```

在本示例中，LookupAddress 和 LookupPhone 分支并行执行。以下是可视工作流在 Step Functions 控制台中的显示效果。



每个分支必须是自包含的。对于 Parallel 状态的一个分支中的状态，不能具有将该分支之外的字段作为目标的 Next 字段，该分支之外的任何其他状态也不能转换到该分支中。

并行状态输入和输出处理

Parallel 状态向每个分支提供其自己输入数据的副本 (需要由 InputPath 字段修改)。它会生成一个数组输出，在该数组中每个分支有一个元素，包含来自该分支的输出。不要求所有元素具有相同类型。输出数组可以按照常规方法，使用 ResultPath 字段插入到输入数据（并且整体作为 Parallel 状态的输出发送）（请参阅输入和输出处理）。

```
{
  "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
            }
          }
        }
      ]
    }
  }
}
```

如果向 FunWithMath 状态提供了数组 [3, 2] 作为输入，则 Add 和 Subtract 状态均接收该数组作为输入。Add 和 Subtract 任务的输出将是数组元素 3 和 2 的和与差，即 5 和 1，而 Parallel 状态的输出将是一个数组。

```
[ 5, 1 ]
```

> **Tip**
>
> 如果您在状态机中使用的 Parallel 或 Map 状态返回由数组组成的数组，您可以使用 ResultSelector 字段将他们转换为一个平面数组。有关更多信息，请参阅展平由数组组成的数组。

错误处理

如果任何分支由于未处理的错误或者转换为 `Fail` 状态而失败，则整个 `Parallel` 状态将视为已失败，并且会停止所有其分支。如果 `Parallel` 状态本身不处理错误，Step Functions 将停止执行，并出现错误。

> **Note**
>
> 当 Parallel 状态失败后，调用的 Lambda 函数将继续运行并且处理任务令牌的活动工作线程不会停止。
>
> - 要停止长时间运行的活动，请使用检测信号检测 Step Functions 是否已停止其分支并停止正在处理任务的工作线程。如果状态已失败，则调用 `SendTaskHeartbeat`、`SendTaskSuccess` 或 `SendTaskFailure` 将引发错误。请参阅检测信号错误。
> - 运行中的 Lambda 函数无法停止。如果您已实施回退，请使用 `Wait` 状态，以便清理工作在 Lambda 函数完成后出现。

Map

使用 `Map` 状态，为数据集中的每个项目运行一组工作流步骤。`Map` 状态的迭代是并行运行的，从而可以快速处理数据集。`Map` 状态可以使用多种输入类型，包括 JSON 数组、Amazon S3 对象列表或 CSV 文件。

Step Functions 为在工作流中使用 `Map` 状态提供了两种类型的处理模式：内联模式和分布式模式。

有关这些模式以及如何在这些模式下使用 `Map` 状态的信息，请参阅以下主题：

- Map 状态处理模式
  - 在内联模式下使用 Map 状态
  - 使用分布式模式下的 Map 状态

> **Tip**
>
> 要在您的 AWS 账户中部署使用 `Map` 状态的工作流示例，请参阅《AWS Step Functions 研讨会》中的模块 5 - Choice 状态和 Map 状态。

Map 状态处理模式

根据您希望如何处理数据集中的项目，Step Functions 为 Map 状态提供以下处理模式。

- 内联 – 限制并发模式。在此模式下，Map 状态的每次迭代都在包含 Map 状态的工作流的上下文中运行。Step Functions 会将这些迭代的执行历史记录添加到父工作流的执行历史记录中。默认情况下，Map 状态以内联模式运行。

  在此模式下，Map 状态仅接受 JSON 数组作为输入。此外，此模式最多支持 40 次并发迭代。

  有关更多信息，请参阅在内联模式下使用 Map 状态。
- 分布式 – 高并发模式。在此模式下，Map 状态将每次迭代作为子工作流执行运行，从而实现多达 1 万个并行子工作流执行的高并发数。每个子工作流执行都有自己的、独立于父工作流的执行历史记录。

  在此模式下，Map 状态可以接受 JSON 数组或 Amazon S3 数据来源（例如 CSV 文件）作为其输入。

  有关更多信息，请参阅使用分布式模式下的 Map 状态。

应使用的模式取决于您希望如何处理数据集中的项目。如果工作流的执行历史记录不超过 2.5 万个条目，或者需要的并发迭代次数不超过 40 次，则可以使用内联模式的 Map 状态。

当您需要编排满足以下任意条件组合的大规模并行工作负载时，请使用分布式模式的 Map 状态：

- 数据集的大小超过 256 KB。
- 该工作流程的执行事件历史记录超过 2.5 万个条目。
- 您需要一个超过 40 次并行迭代的并发数。

主题

- 内联模式和分布式模式的区别
- 在内联模式下使用 Map 状态
- 使用分布式模式下的 Map 状态编排大规模并行工作负载

内联模式和分布式模式的区别

下表突出显示了内联模式和分布式模式之间的区别。

内联模式	分布式模式
Supported data sources	
接受从工作流中上一步传递的 JSON 数组作为输入。	接受以下数据来源作为输入： • 从工作流中上一步传递的 JSON 数组 • 包含数组的 Amazon S3 存储桶中的 JSON 文件 • Amazon S3 存储桶中的 CSV 文件 • Amazon S3 对象列表 • Amazon S3 清单
Map iterations	
在此模式下，Map 状态的每次迭代都在包含 Map 状态的工作流的上下文中运行。Step Functions 会将这些迭代的执行历史记录添加到父工作流的执行历史记录中。	在此模式下，Map 状态将每次迭代作为子工作流执行运行，从而实现多达 1 万个并行子工作流执行的高并发数。每个子工作流执行都有自己的、独立于父工作流的执行历史记录。
Maximum concurrency for parallel iterations	
允许尽可能同时运行多达 40 次迭代。	允许运行多达 1 万个并行子工作流执行，一次处理数百万个数据项。
Input payload and event history sizes	
强制限制 256 KB 的输入有效负载大小，执行事件历史记录中最多包含 2.5 万个条目。	允许您无视有效负载大小限制，因为 Map 状态可以直接从 Amazon S3 数据来源读取输入。 在此模式下，您还可以无视执行历史记录的限制，因为 Map 状态启动的子工作流执行会保留自己的、独立于父工作流的执行历史记录。
Monitoring and observability	
您可以通过控制台或调用 GetExecutionHistory API 操作来查看工作流的执行历史记录。	当您运行分布式模式下的 Map 状态时，Step Functions 会创建一个 Map Run 资源。Map Run 是指分布式 Map 状态启动的一组子工作流

内联模式	分布式模式
您还可以通过 CloudWatch 和 X-Ray 查看执行历史记录。	执行。您可以在 Step Functions 控制台中查看 Map Run。您也可以调用 `DescribeMapRun` API 操作。Map Run 还会向 CloudWatch 发出指标。 有关更多信息，请参阅检查分布式 Map 状态的 Map Run。

在内联模式下使用 Map 状态

默认情况下，Map 状态以内联模式运行。在内联模式下，Map 状态仅接受 JSON 数组作为输入。它接收来自工作流中上一步的数组。在此模式下，Map 状态的每次迭代都在包含 Map 状态的工作流的上下文中运行。Step Functions 会将这些迭代的执行历史记录添加到父工作流的执行历史记录中。

在此模式下，Map 状态最多支持 40 次并发迭代。

设置为内联的 Map 状态称为内联 Map 状态。如果工作流的执行历史记录不超过 2.5 万个条目，或者需要的并发迭代次数不超过 40 次，则可以使用内联模式的 Map 状态。

有关使用内联 Map 状态的介绍，请参阅教程使用内联 Map 状态重复操作。

内容

- 本主题中的关键概念
- 内联 Map 状态字段
- 已弃用的字段
- 内联 Map 状态示例
- 使用 ItemSelector 的内联 Map 状态示例
- 内联 Map 状态输入和输出处理

本主题中的关键概念

内联模式

Map 状态的有限并发模式。在此模式下，Map 状态的每次迭代都在包含 Map 状态的工作流的上下文中运行。Step Functions 会将这些迭代的执行历史记录添加到父工作流的执行历史记录中。默认情况下，Map 状态在内联模式下运行。

此模式仅接受 JSON 数组作为输入，最多支持 40 次并发迭代。

内联 Map 状态

设置为内联模式的 Map 状态。

Map 工作流

Map 状态为每次迭代运行的一组步骤。

Map 状态迭代

在 Map 状态内部定义的重复工作流。

内联 Map 状态字段

要在工作流中使用内联 Map 状态，请指定以下字段中的一个或多个。除了公共状态字段外，您还可以指定以下字段。

Type（必填）

设置状态的类型，例如 Map。

ItemProcessor（必填）

包含以下 JSON 对象，用于指定 Map 状态处理模式和定义。

该定义包含处理每个数组项目时要重复的一组步骤。

- ProcessorConfig – 一个可选的 JSON 对象，用于指定 Map 状态的处理模式。此对象包含 Mode 子字段。此字段默认为 INLINE，即在内联模式下使用 Map 状态。

  在此模式下，任何迭代失败都会导致 Map 状态失败。当 Map 状态失败时，所有迭代都会停止。
- StartAt– 指定表示工作流中第一个状态的字符串。该字符串区分大小写，必须与某个状态对象的名称相匹配。此状态首先针对数据集中的每个项目运行。您向 Map 状态提供的任何执行输入都将首先传递给 StartAt 状态。

- `States` – 一个 JSON 对象，其中包含逗号分隔的状态集合。在此对象中，您可以定义 Map workflow。

  > **Note**
  >
  > - `ItemProcessor` 字段中的状态只能相互转换。`ItemProcessor` 字段外的任何状态都不能转换到字段内的状态。
  > - `ItemProcessor` 字段取代了现已弃用的 `Iterator` 字段。尽管您可以继续包含使用 `Iterator` 字段的 `Map` 状态，但我们强烈建议您将此字段替换为 `ItemProcessor`。
  >
  >   Step Functions Local 目前不支持 `ItemProcessor` 字段。我们建议对 Step Functions Local 使用 `Iterator` 字段。

`ItemsPath`（可选）

使用 JsonPath 语法指定参考路径。此路径选择包含状态输入内项目数组的 JSON 节点。有关更多信息，请参阅ItemsPath。

`ItemSelector`（可选）

在输入数组项的值传递到每次 `Map` 状态迭代之前，覆盖这些值。

在此字段中，指定包含键值对集合的有效 JSON。这些对可包含以下任何内容：

- 在状态机定义中定义的静态值。
- 使用路径从状态输入中选择的值。
- 从上下文对象中访问的值。

有关更多信息，请参阅ItemSelector。

`ItemSelector` 字段取代了现已弃用的 `Parameters` 字段。尽管您可以继续包含使用 `Parameters` 字段的 `Map` 状态，但我们强烈建议您将此字段替换为 `ItemSelector`。

`MaxConcurrency`（可选）

指定一个整数值，该值提供可以并行运行的 `Map` 状态迭代次数的上限。例如，`MaxConcurrency` 值为 10 将限制您的 `Map` 状态同时运行 10 次并发迭代。

> **Note**
>
> 并发迭代可能会受到限制。发生这种情况时，有些迭代要等到之前的迭代完成后才会开始。当输入数组中的项目超过 40 个时，发生这种情况的可能性就会增加。

要实现更高的并发数，请考虑使用分布式模式下的 Map 状态。

默认值为 0，这对并发数没有限制。Step Functions 尽可能同时调用迭代。

MaxConcurrency 值为 1 会对每个数组元素调用一次 ItemProcessor。数组中的项目按其在输入中的出现顺序进行处理。Step Functions 在完成前一次迭代之后才会开始新的迭代。

MaxConcurrencyPath（可选）

如果要使用参考路径从状态输入中动态提供最大并发数值，请使用 MaxConcurrencyPath。解决后，参考路径必须选择一个值为非负整数的字段。

> **Note**
>
> 一个 Map 状态不能同时包含 MaxConcurrency 和 MaxConcurrencyPath。

ResultPath（可选）

指定输入中存储 Map 状态迭代输出的位置。然后，Map 状态按照 OutputPath 字段（如果已指定）的指定筛选输入。然后，它使用筛选后的输入作为状态的输出。有关更多信息，请参阅输入和输出处理。

ResultSelector（可选）

传递键值对集合，其中，键值为静态值或从结果中选择。有关更多信息，请参阅ResultSelector。

> **Tip**
>
> 如果您在状态机中使用的 Parallel 或 Map 状态返回由数组组成的数组，您可以使用 ResultSelector 字段将他们转换为一个平面数组。有关更多信息，请参阅展平由数组组成的数组。

Retry（可选）

一个称为重试器的对象数组，用于定义重试策略。状态在遇到运行时错误时会使用重试策略。有关更多信息，请参阅使用 Retry 和使用 Catch 的状态机示例。

> **Note**
>
> 如果您为内联 Map 状态定义了重试器，则重试策略将应用于所有 Map 状态迭代，而不仅仅是失败的迭代。例如，Map 状态包含两次成功的迭代和一次失败的迭代。如果您已经为 Map 状态定义了 Retry 字段，则重试策略将应用于所有三次 Map 状态迭代，而不仅仅是失败的迭代。

Catch（可选）

一个称为捕获器的对象数组，用于定义回退状态。如果状态遇到了运行时错误，且没有应用重试策略，或者重试策略已用尽，则会运行捕获器。有关更多信息，请参阅回退状态。

已弃用的字段

> **Note**
>
> 尽管您可以继续包含使用以下字段的 Map 状态，但我们强烈建议您将 Iterator 替换为 ItemProcessor，将 Parameters 替换为 ItemSelector。

Iterator

指定一个 JSON 对象，该对象定义了一组处理数组中每个元素的步骤。

Parameters

指定键值对集合，其中，值可包含以下任何内容：

- 在状态机定义中定义的静态值。
- 使用路径从输入中选择的值。

内联 Map 状态示例

考虑以下以内联模式运行的 Map 状态的输入数据。

```
{
  "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 }
    ]
  }
}
```

给定前面的输入，以下示例中的 Map 状态将为 shipped 字段中数组的每个项目调用一次名为 ship-val 的 AWS Lambda 函数。

```
"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 状态的每次迭代都会将数组中的一个项目（通过 ItemsPath 字段选择）作为输入发送到 ship-val Lambda 函数。以下值是 Map 状态发送给调用 Lambda 函数的输入示例：

```
{
  "prod": "R31",
  "dest-code": 9511,
  "quantity": 1344
}
```

完成后，Map 状态的输出是一个 JSON 数组，其中每个项目都是迭代的输出。在本例中，此数组包含 ship-val Lambda 函数的输出。

使用 **ItemSelector** 的内联 Map 状态示例

假设上一个示例中的 ship-val Lambda 函数还需要有关货件快递员的信息。该信息是对每次迭代的数组中项目的补充。您可以包括来自输入的信息，以及特定于 Map 状态的当前迭代的信息。请注意下面示例中的 ItemSelector 字段：

```
"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 块用一个 JSON 节点替换了迭代的输入。此节点既包含来自上下文对象的当前项目数据，也包含来自 Map 状态输入 delivery-partner 字段的快递员信息。以下是一个单次迭代输入示例。Map 状态会将此输入传递给 ship-val Lambda 函数的调用。

```
{
  "parcel": {
    "prod": "R31",
    "dest-code": 9511,
    "quantity": 1344
   },
   "courier": "UQS"
}
```

在前面的内联 Map 状态示例中，ResultPath 字段以与输入相同的格式生成输出。但是，它会用一个数组覆盖 detail.shipped 字段，数组中每个元素都是每次迭代的 ship-val Lambda 调用的输出。

有关使用内联 Map 状态及其字段的更多信息，请参阅以下内容。

- 使用内联 Map 状态重复操作
- Step Functions 中的输入和输出处理
- ItemsPath
- Map 状态的上下文对象数据

内联 Map 状态输入和输出处理

对于给定的 Map 状态，InputPath 选择该状态输入的子集。

Map 状态的输入必须包含 JSON 数组。Map 状态为数组中的每个项目运行一次 ItemProcessor 部分。如果指定了 ItemsPath 字段，则 Map 状态会选择在输入中的哪个位置查找要迭代的数组。如果未指定，ItemsPath 的值为 $，而 ItemProcessor 部分预期此数组是唯一的输入。如果指定了 ItemsPath 字段，则其值必须为参考路径。Map 状态在应用 InputPath 之后，会将此路径应用于有效输入。ItemsPath 必须标识值为 JSON 数组的字段。

默认情况下，每次迭代的输入都是由 ItemsPath 值标识的数组字段的单个元素。您可以使用 ItemSelector 字段覆盖此值。

完成后，Map 状态的输出是一个 JSON 数组，其中每个项目都是迭代的输出。

有关内联 Map 状态输入和输出的更多信息，请参阅以下内容：

- 使用内联 Map 状态重复操作

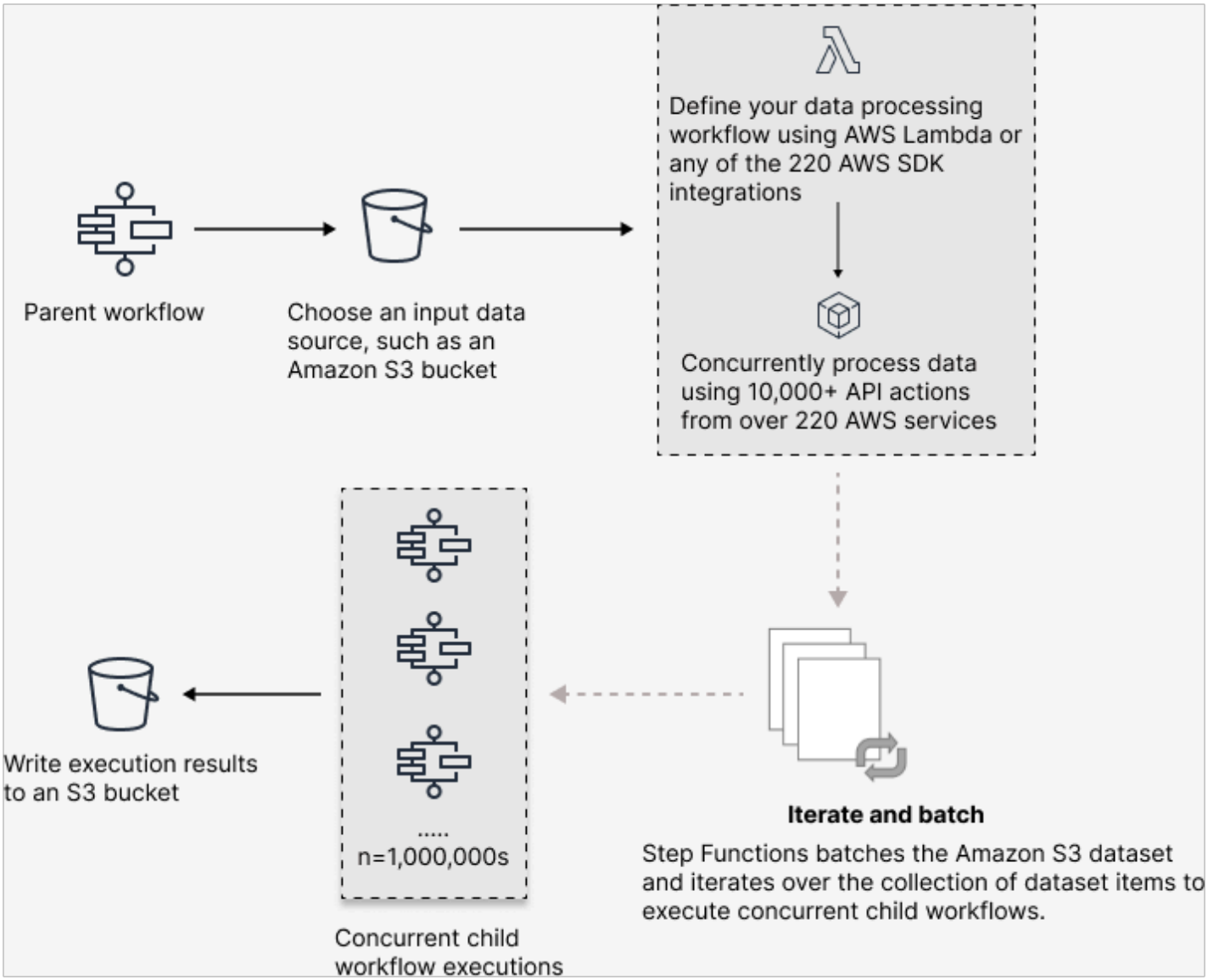
- 使用 ItemSelector 的内联 Map 状态示例
- Step Functions 中的输入和输出处理
- Map 状态的上下文对象数据
- 使用 Map 状态动态处理数据

使用分布式模式下的 Map 状态编排大规模并行工作负载

借助 Step Functions，您可以编排大规模的并行工作负载来执行任务，例如按需处理半结构化数据。这些并行工作负载允许您同时处理存储在 Amazon S3 中的大规模数据来源。例如，您可以处理包含大量数据的单个 JSON 或 CSV 文件。或者，您也可以处理一大组 Amazon S3 对象。

要在工作流中设置大规模的并行工作负载，请添加一个分布式模式下的 Map 状态。Map 状态 可同时处理数据集中的项目。设置为分布式的 Map 状态被称为分布式 Map 状态。在分布式模式下，Map 状态允许高并发处理。在分布式模式下，Map 状态在称为子工作流执行 迭代中处理数据集中的项目。您可以指定可并行运行的子工作流执行的数量。每个子工作流执行都有自己的、独立于父工作流的执行历史记录。如果您未指定，Step Functions 将并行运行 1 万个并行子工作流执行。

下图说明了如何在工作流中设置大规模的并行工作负载。



本主题内容

- 关键术语
- 分布式 Map 状态定义示例
- 运行分布式 Map 的权限
- 分布式 Map 状态字段
- 后续步骤

关键术语

分布式模式

Map 状态的一种处理模式。在此模式下，Map 状态的每次迭代都作为子工作流执行来运行，从而实现高并发数。每个子工作流执行都有自己的执行历史记录，独立于父工作流的执行历史。该模式支持从大规模 Amazon S3 数据来源读取输入。

分布式 Map 状态

设置为分布式处理模式的 Map 状态。

Map 工作流

Map 状态运行的一组步骤。

父工作流

包含一个或多个分布式 Map 状态的工作流。

子工作流执行

分布式 Map 状态的一次迭代。子工作流执行有自己的执行历史记录，独立于父工作流的执行历史。

Map Run

当您运行分布式模式下的 Map 状态时，Step Functions 会创建一个 Map Run 资源。Map Run 是指分布式 Map 状态 启动的一组子工作流执行，以及控制这些执行的运行时设置。Step Functions 会为 Map Run 分配一个 Amazon 资源名称 (ARN)。您可以在 Step Functions 控制台中查看 Map Run。您也可以调用 DescribeMapRun API 操作。Map Run 还会向发送指标。 CloudWatch

有关更多信息，请参阅 检查 Map Run。

分布式 Map 状态定义示例

当您需要编排满足以下任意条件组合的大规模并行工作负载时，请使用分布式模式的 Map 状态：

- 数据集的大小超过 256 KB。
- 该工作流程的执行事件历史记录超过 2.5 万个条目。
- 您需要一个超过 40 次并行迭代的并发数。

下面的分布式 Map 状态 定义示例将数据集指定为存储在 Amazon S3 存储桶中的 CSV 文件。它还指定了一个 Lambda 函数，用于处理 CSV 文件每一行中的数据。由于此示例使用了一个 CSV 文件，因

此它还指定了 CSV 列标题的位置。要查看此示例的完整状态机定义，请参阅教程使用分布式 Map 复制大规模 CSV 数据。

```
{
  "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"
      }
    }
  }
}
```

运行分布式 Map 的权限

当您在工作流中包含分布式 Map 状态 时，Step Functions 需要适当的权限才能允许状态机角色为分布式 Map 状态 调用 StartExecution API 操作。

以下 IAM 策略示例授予您的状态机角色运行分布式 Map 状态 所需的最低权限。

> **Note**
>
> 确保将 *stateMachineName* 替换为使用分布式 Map 状态 的状态机的名称。例如，arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*。

```
{
  "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:*"
    }
  ]
```

```
}
```

此外，您需要确保您拥有访问分布式地图状态下使用的 AWS 资源（例如 Amazon S3 存储桶）所需的最低权限。有关信息，请参阅使用分布式 Map 状态的 IAM 策略。

分布式 Map 状态字段

要在工作流中使用分布式 Map 状态，请指定其中一个或多个字段。除了公共状态字段外，您还可以指定以下字段。

`Type`（必填）

设置状态的类型，例如 `Map`。

`ItemProcessor`（必填）

包含以下 JSON 对象，用于指定 `Map` 状态处理模式和定义。

- `ProcessorConfig` – 一个 JSON 对象，用于指定 `Map` 状态的配置。此对象包含以下子字段：
  - `Mode` – 设置为 **`DISTRIBUTED`**，以在分布式模式下使用 `Map` 状态。

    > **Note**
    >
    > 目前，如果您在快速工作流中使用 `Map` 状态，则无法将 `Mode` 设置为 `DISTRIBUTED`。但是，如果您在标准工作流中使用 `Map` 状态，则可以将 `Mode` 设置为 `DISTRIBUTED`。

  - `ExecutionType` – 将 Map 工作流的执行类型指定为标准或快速。如果您为 `Mode` 子字段指定了 `DISTRIBUTED`，则必须提供此字段。有关工作流类型的更多信息，请参阅标准和快速工作流。
- `StartAt`– 指定表示工作流中第一个状态的字符串。该字符串区分大小写，必须与某个状态对象的名称相匹配。此状态首先针对数据集中的每个项目运行。您向 `Map` 状态提供的任何执行输入都将首先传递给 `StartAt` 状态。
- `States` – 一个 JSON 对象，其中包含逗号分隔的状态集合。在此对象中，您可以定义 Map workflow。

`ItemReader`

指定一个数据集及其位置。`Map` 状态接收来自指定数据集的输入数据。

在分布式模式下，您可以使用从先前状态传递的 JSON 有效负载，也可以使用大规模的 Amazon S3 数据来源作为数据集。有关更多信息，请参阅 ItemReader。

`ItemsPath`（可选）

使用JsonPath语法指定参考路径，以选择包含状态输入内项目数组的 JSON 节点。

在分布式模式下，只有使用上一步中的 JSON 数组作为状态输入时，才能指定此字段。有关更多信息，请参阅ItemsPath。

`ItemSelector`（可选）

在将单个数据集项目的值传递到每次 Map 状态迭代之前，覆盖这些值。

在此字段中，指定包含键值对集合的有效 JSON 输入。这些对可以是您在状态机定义中定义的静态值，使用路径从状态输入中选择的值，或者从上下文对象中获取的值。有关更多信息，请参阅 ItemSelector。

`ItemBatcher`（可选）

指定批量处理数据集项目。然后，每个子工作流执行都会收到一批这些项目作为输入。有关更多信息，请参阅 ItemBatcher。

`MaxConcurrency`（可选）

指定可并行运行的子工作流数量。解释器只允许执行指定数量的并行子工作流。如果您未指定并发数或将其设置为零，Step Functions 不会限制并发数，并且会运行 1 万个并行子工作流执行。

> **Note**
>
> 虽然您可以为并行子工作流程执行指定更高的并发限制，但我们建议您不要超过下游 AWS 服务的容量，例如 AWS Lambda。

`MaxConcurrencyPath`（可选）

如果要使用参考路径从状态输入中动态提供最大并发数值，请使用 `MaxConcurrencyPath`。解决后，参考路径必须选择一个值为非负整数的字段。

> **Note**
>
> 一个 Map 状态不能同时包含 `MaxConcurrency` 和 `MaxConcurrencyPath`。

`ToleratedFailurePercentage`（可选）

定义在 Map Run 中允许的失败项目的百分比。如果超过此百分比，Map Run 将自动失败。Step Functions 通过失败或超时项目总数除以项目总数计算失败项目的百分比。您可以指定 0 到 100 之间的值。有关更多信息，请参阅 分布式 Map 状态容许的故障阈值。

`ToleratedFailurePercentagePath`（可选）

如果要使用参考路径从状态输入中动态提供容许的故障百分比值，请使用 `ToleratedFailurePercentagePath`。解决后，参考路径必须选择一个值介于 0 到 100 之间的字段。

`ToleratedFailureCount`（可选）

定义 Map Run 中允许的失败项目数量。如果超过此数量，Map Run 将自动失败。有关更多信息，请参阅 分布式 Map 状态容许的故障阈值。

`ToleratedFailureCountPath`（可选）

如果要使用参考路径从状态输入中动态提供容许的故障计数值，请使用 `ToleratedFailureCountPath`。解决后，参考路径必须选择一个值为非负整数的字段。

`Label`（可选）

唯一标识 Map 状态的字符串。对于每个 Map Run，Step Functions 都会将标签添加到 Map Run ARN。以下是带有名为 `demoLabel` 的自定义标签的 Map Run ARN 的示例：

```
arn:aws:states:us-east-1:123456789012:mapRun:demoWorkflow/
demoLabel:3c39a231-69bb-3d89-8607-9e124eddbb0b
```

如果您未指定标签，则 Step Functions 会自动生成一个唯一标签。

> **Note**
>
> 标签长度不能超过 40 个字符，在状态机定义中必须唯一，并且不能包含任何以下字符：
>
> - 空格字符
> - 通配符 (? *)
> - 方括号字符 (< > { } [ ])
> - 特殊字符 (: ; , \ | ^ ~ $ # % & ` ")
> - 控制字符（\\u0000 - \\u001f 或 \\u007f - \\u009f）

Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

`ResultWriter`（可选）

指定 Step Functions 写入所有子工作流执行结果的 Amazon S3 位置。

Step Functions 整合了所有子工作流执行数据，例如执行输入和输出、ARN 和执行状态。然后，它将状态相同的执行导出到指定 Amazon S3 位置的相应文件中。有关更多信息，请参阅 ResultWriter。

如果您不导出 Map 状态结果，它将返回一个包含所有子工作流执行结果的数组。例如：

```
[1, 2, 3, 4, 5]
```

`ResultPath`（可选）

指定输入中放置迭代输出的位置。接下来，输入将按照 `OutputPath` 字段（如果存在）指定的内容进行筛选，然后再用作状态输出传递。有关更多信息，请参阅输入和输出处理。

`ResultSelector`（可选）

传递一个键值对集合，其中，值为静态值或从结果中选择的值。有关更多信息，请参阅 ResultSelector。

> **Tip**
>
> 如果您在状态机中使用的 Parallel 或 Map 状态返回由数组组成的数组，您可以使用 ResultSelector 字段将他们转换为一个平面数组。有关更多信息，请参阅 展平由数组组成的数组。

`Retry`（可选）

一个称为重试器的对象数组，用于定义重试策略。如果状态遇到运行时错误，则执行将使用重试策略。有关更多信息，请参阅 使用 Retry 和使用 Catch 的状态机示例。

> **Note**
>
> 如果您为分布式 Map 状态 定义了重试器，则重试策略将应用于 Map 状态启动的所有子工作流执行。例如，假设 Map 状态启动了三个子工作流执行，其中一个失败了。失败发生时，将执行将使用 Retry 字段（如果已定义），用于 Map 状态。重试策略应用于所有子工作流执行，而不仅仅是失败的执行。如果一个或多个子工作流执行失败，则 Map Run 将失败。
>
> 当您重试某个 Map 状态时，它会创建一个新的 Map Run。

Catch（可选）

一个称为捕获器的对象数组，用于定义回退状态。如果状态遇到运行时错误，Step Functions 将使用 Catch 中定义的捕获器。发生错误时，执行会首先使用 Retry 中定义的任何重试器。如果重试策略未定义或已用尽，则执行将使用其捕获器（如果已定义）。有关更多信息，请参阅回退状态。

后续步骤

要继续了解有关分布式 Map 状态的更多信息，请参阅以下资源：

- 输入和输出处理

  为了配置分布式 Map 状态接收的输入及其生成的输出，Step Functions 提供了以下字段：

  - ItemReader
  - ItemsPath
  - ItemSelector
  - ItemBatcher
  - ResultWriter
  - 解析输入 CSV 文件

  除了这些字段外，Step Functions 还可以为分布式 Map 定义容许的故障阈值。此值可用于指定失败项目的最大数量或百分比作为 Map Run 的失败阈值。有关配置容许的故障阈值的更多信息，请参阅分布式 Map 状态容许的故障阈值。

- 使用分布式 Map 状态

  要开始使用分布式 Map 状态，请参阅以下教程和示例项目。

  - 开始使用分布式 Map 状态

- 使用 Lambda 函数处理整批数据
- 使用 Lambda 函数处理单个数据项
- 示例项目：使用分布式 Map 处理 CSV 文件
- 示例项目：使用分布式 Map 处理 Amazon S3 存储桶中的数据

- 检查分布式 Map 状态执行

  Step Functions 控制台提供 Map Run 详细信息 页面，其中显示了与分布式 Map 状态 执行相关的所有信息。有关如何检查此页面上显示的信息的信息，请参阅检查 Map Run。

分布式 Map 状态容许的故障阈值

在编排大规模并行工作负载时，您还可以定义一个容许的故障阈值。此值可用于指定失败项目的最大数量或百分比作为 Map Run 的失败阈值。根据您指定的数量值或百分比值，如果超出阈值，Map Run 将自动失败。如果同时指定两个值，则当工作流超过任一值时，工作流就会失败。

指定阈值可以帮助您在整个 Map Run 失败之前允许特定定数量的项目失败。当 Map Run 因超过指定阈值而失败时，Step Functions 会返回 `States.ExceedToleratedFailureThreshold` 错误。

> **Note**
>
> 即使在超过容许的失败阈值之后，Step Functions 仍可能在 Map Run 失败之前继续运行 Map Run 中的子工作流。

要在 Workflow Studio 中指定阈值，请在运行时设置字段下的其他配置中选择设置容许的故障阈值。

容许的失败百分比

定义容许的失败项目所占的百分比。如果超过此值，Map Run 将失败。Step Functions 通过失败或超时项目总数除以项目总数计算失败项目的百分比。您可以指定 0 到 100 之间的值。默认百分比值为零，意味着如果任何一个子工作流执行失败或超时，工作流就会失败。如果将百分比指定为 100，则即使所有子工作流执行都失败，该工作流也不会失败。

或者，您可以将百分比指定为分布式 Map 状态 输入中现有键值对的参考路径。在运行时，此路径必须解析为 0 到 100 之间的正整数。您可以在 `ToleratedFailurePercentagePath` 子字段中指定参考路径。

例如，给定以下输入：

```
{
  "percentage": 15
}
```

您可以使用该输入的参考路径来指定百分比，如下所示：

```
{
  ...
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailurePercentagePath": "$.percentage"
    ...
  }
}
```

> **Important**
>
> 您可以在分布式 Map 状态定义中指定 ToleratedFailurePercentage 或 ToleratedFailurePercentagePath，但不能同时指定两者。

容许的失败数

定义容许的失败项目数。如果超过此值，Map Run 将失败。

或者，您可以将该数值指定为分布式 Map 状态 输入中现有键值对的参考路径。在运行时，此路径必须解析为正整数。您可以在 ToleratedFailureCountPath 子字段中指定参考路径。

例如，给定以下输入：

```
{
  "count": 10
}
```

您可以使用该输入的参考路径来指定数值，如下所示：

```
{
  ...
```

```
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailureCountPath": "$.count"
    ...
  }
}
```

> **Important**
>
> 您可以在分布式 Map 状态定义中指定 ToleratedFailureCount 或 ToleratedFailureCountPath，但不能同时指定两者。

转换

在启动状态机的新执行时，系统以在顶级 StartAt 字段中引用的状态开始。此字段为字符串形式，必须完全匹配工作流中某个状态的名称，并且区分大小写。

运行状态之后，AWS Step Functions 使用 Next 字段的值来确定要前进到的下一个状态。

Next 字段还将状态名称指定为字符串。此字符串区分大小写，并且必须完全匹配状态机定义中指定的状态名称。

例如，以下状态包括到 NextState 的转换。

```
"SomeState" : {
  ...,
  "Next" : "NextState"
}
```

大部分状态只允许使用具有 Next 字段的一个转换规则。不过，某些流控制状态（例如，Choice 状态）允许您指定多个转换规则，每个具有各自的 Next 字段。Amazon States Language 提供有关您可指定的各个状态类型的详细信息，包括如何指定转换的信息。

状态可以有多个来自其他状态的传入转换。

该过程重复，直至到达最终状态 ("Type": Succeed、"Type": Fail 或 "End": true 状态) 或者出现运行时错误。

当您redrive执行时，则视为一种状态转换。此外，在redrive中重新运行的所有状态也视为状态转换。

以下规则适用于状态机中的状态：

- 状态在封闭块中可以按任意顺序出现。但是，它们列出的顺序不会影响到它们的运行顺序。运行顺序由状态的内容决定。
- 在状态机中，只能有一个状态指定为 `start` 状态。`start` 状态由顶级结构中 `StartAt` 字段的值定义。
- 根据您的状态机逻辑，您可能会有多个 `end` 状态 (例如状态机有多个逻辑分支时)。
- 如果状态机只包含一个状态，该状态可以为 start 状态和 end 状态。

分布式 Map 状态下的转换

当您在分布式模式下使用 `Map` 状态时，对于分布式 Map 状态 启动的每个子工作流执行，都会收取一次状态转换的费用。当您在内联模式下使用 `Map` 状态时，您无需为内联 Map 状态 的每次迭代支付状态转换费用。

您可以在分布式模式下使用 `Map` 状态，并在 `Map` 状态定义中包含嵌套工作流，进而优化成本。当您启动快速类型的子工作流执行时，分布式 Map 状态还会增加更多价值。Step Functions 存储快速子工作流执行的响应和状态，从而减少了在 CloudWatch Logs 中存储执行数据的需求。您还可以访问分布式 Map 状态 下的可用流量控制，例如定义错误阈值或批处理一组项目。有关 Step Functions 定价的信息，请参阅 AWS Step Functions 定价。

状态机数据

状态机数据采用以下形式：

- 状态机的初始输入
- 在状态之间传递的数据
- 状态机的输出

此部分介绍在 AWS Step Functions 中状态机数据的格式及如何使用。

主题

- 数据格式
- 状态机输入/输出

- 状态输入/输出

数据格式

状态机数据由 JSON 文本表示。您可以使用 JSON 支持的任何数据类型向状态机提供值。

> **Note**
>
> - JSON 文本格式的数字符合 JavaScript 语义。这些数字通常对应于双精度 IEEE-854 值。
> - 以下是有效的 JSON 文本：
>   - 以引号分隔的独立字符串
>   - 对象
>   - 数组
>   - 数字
>   - 布尔值
>   - `null`
> - 一个状态的输出成为下一个状态的输入。但是，您可以使用输入和输出处理，将状态限制为处理输入数据的子集。

状态机输入/输出

您可以通过以下两种方式之一将初始输入数据提供给 AWS Step Functions 状态机。开始执行时，可以将数据传递给一个 `StartExecution` 操作。您也可以从 Step Functions 控制台将数据传递到状态机。初始数据传递到状态机的 `StartAt` 状态。如果未提供输入，默认值为空对象 (`{}`)。

执行的输出由最后一个状态 (`terminal`) 返回。此输出以 JSON 文本形式显示在执行结果中。

对于标准工作流，您可以使用外部调用方（例如，在 `DescribeExecution` 操作中），从执行历史记录中检索执行结果。您可以在 Step Functions 控制台上查看执行结果。

对于快速工作流，如果您启用了日志记录，则可以从 CloudWatch Logs 中检索结果，或者在 Step Functions 控制台中查看和调试执行。有关更多信息，请参阅使用 CloudWatch Logs 进行日志记录和在 Step Functions 控制台上查看和调试执行。

您还应该考虑与状态机相关的配额。有关更多信息，请参阅配额。

状态输入/输出

每个状态的输入包含来自前一个状态的 JSON 文本，对于 StartAt 状态来说，是对执行的输入。某些流控制状态的输出与输入相同。

在以下示例中，状态机将两个数字加到一起。

1. 定义 AWS Lambda 函数。

```
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. 定义状态机。

```
{
  "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. 使用以下 JSON 文本开始执行。

```
{ "numbers": [3, 4] }
```

   Add 状态接收 JSON 文本并将其传递到 Lambda 函数

   Lambda 函数将计算结果返回给状态。

状态在其输出中返回以下值。

```
{ "result": 7 }
```

由于 Add 还是状态机中的最后一个状态，此值作为状态机的输出返回。

如果最后一个状态未返回输出，则状态机返回空对象 ({})。

有关更多信息，请参阅Step Functions 中的输入和输出处理。

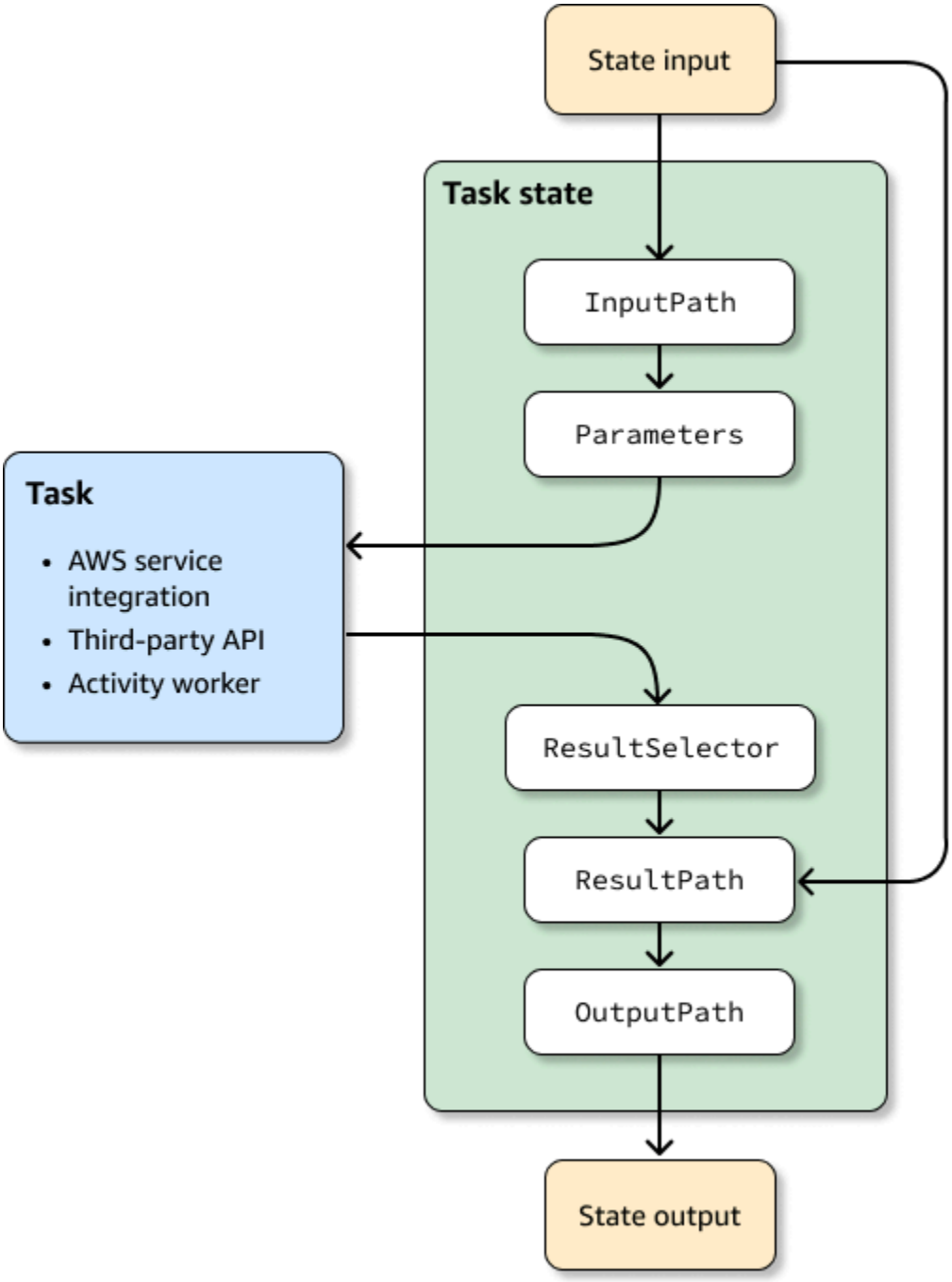
Step Functions 中的输入和输出处理

Step Functions 执行接收 JSON 文本作为输入并将该输入传递到工作流中的第一个状态。各个状态将接收输入形式的 JSON 并通常将 JSON 以输出形式传递到下一个状态。了解此信息如何在各个状态之间流动，并了解如何筛选和操作此数据，这对在 AWS Step Functions中高效设计和实施工作流程至关重要。

在 Amazon States Language 中，这些字段筛选和控制 JSON 在各个状态之间的流动：

- `InputPath`
- `Parameters`
- `ResultSelector`
- `ResultPath`
- `OutputPath`

下图显示了 JSON 信息如何在任务状态中移动。 `InputPath`选择 JSON 输入的哪些部分要传递给`Task`状态任务（例如， AWS Lambda 函数）。 `ResultPath`然后选择要传递给输出的状态输入和任务结果的组合。 `OutputPath`可以过滤 JSON 输出以进一步限制传递给输出的信息。



`InputPath`、`Parameters`、`ResultSelector`、`ResultPath` 和 `OutputPath` 均在 JSON 经过工作流程中的每个状态时操纵 JSON。

每个都可以使用路径从输入或结果中选择 JSON 的某些部分。路径是以 $ 开头的字符串，标识 JSON 文本内的节点。Step Functions 路径使用JsonPath语法。

> **Tip**
>
> 使用 Step Functions 控制台中的数据流模拟器来测试 JSON 路径语法，以更好地了解在状态下如何操作数据，并查看数据在状态之间是如何传递的。

> **Tip**
>
> 要将包含输入和输出处理的工作流程示例部署到您的中 AWS 账户，请参阅 AWS Step Functions 研讨会的模块 6-输入和输出处理。

主题

- 路径
- InputPath、参数和 ResultSelector
- ResultPath
- OutputPath
- InputPath、ResultPath 和 OutputPath 示例
- Map 状态输入和输出字段
- Context 对象

路径

在 Amazon States Language中，路径是以 $ 开头的字符串，您可使用该字符串来标识 JSON 文本中的组件。路径遵循JsonPath语法。在指定 `InputPath`、`ResultPath` 和 `OutputPath` 的值时，可以指定访问输入子集的路径。有关更多信息，请参阅 Step Functions 中的输入和输出处理。

> **Note**
>
> 您还可以使用状态定义的 `Parameters` 字段中的路径指定输入或上下文对象的 JSON 节点。请参阅 将参数传递给服务 API。

如果您的字段名包含任何未包含在 JsonPath ABNF 规则`member-name-shorthand`定义中的字符，则必须使用方括号表示法。因此，要对标点符号（不包括_）等特殊字符进行编码，必须使用方括号表示法。例如，`$.abc['def ghi']`。

引用路径

引用路径 是一种语法存在限制的路径，它只能标识 JSON 结构中的单个节点：

- 您可以使用句点 (`.`) 和方括号 (`[ ]`) 表示法访问对象字段。
- 不支持像 `length()` 这样的函数。
- 不支持非符号的词法运算符，例如 `subsetof`。
- 不支持按正则表达式或引用 JSON 结构中的其他值进行筛选。
- 不支持运算符`@,:`、、和？

例如，如果状态输入数据包含以下值：

```
{
  "foo": 123,
  "bar": ["a", "b", "c"],
  "car": {
      "cdr": true
  }
}
```

以下引用路径将返回以下内容。

```
$.foo => 123
$.bar => ["a", "b", "c"]
$.car.cdr => true
```

某些状态使用路径和引用路径来控制状态机的流，或者配置状态的设置或选项。有关更多信息，请参阅中的使用数据流模拟器对工作流输入和输出路径处理进行建模和有效使用 JSONPath。 AWS Step Functions

展平由数组组成的数组

如果状态机中的 Parallel 或 Map 状态返回由数组组成的数组，则可以使用 ResultSelector 字段将他们转换为一个平面数组。您可以将此字段包含在 Parallel 或 Map 状态定义中，以操纵这些状态的结果。

要展平数组，请在 `ResultSelector` 字段中使用 JMESPath 语法 `[*]`，如下方的示例所示。

```
"ResultSelector": {
```

```
    "flattenArray.$": "$[*][*]"
  }
```

有关演示如何展平数组的示例，请参阅以下教程中的第 3 步：

- 使用 Lambda 函数处理整批数据
- 使用 Lambda 函数处理单个数据项

InputPath、参数和 ResultSelector

InputPath、Parameters 和 ResultSelector 字段提供了一种在 JSON 流经工作流时操纵它的方法。InputPath 可以通过使用路径筛选 JSON 表示法来限制传递的输入（请参阅路径）。Parameters 字段使您可以传递键值对的集合，其中值是您在状态机定义中定义的静态值，或者是使用路径从输入中选择的值。ResultSelector 字段提供了一种在应用 ResultPath 状态之前操作状态结果的方法。

AWS Step Functions 首先应用InputPath字段，然后应用字Parameters段。您可以先使用 InputPath 将原始输入筛选为所选内容，然后应用 Parameters 进一步操作该输入，或添加新值。然后，您可以在应用 ResultPath 状态之前使用 ResultSelector 字段来操作状态的输出。

> **Tip**
>
> 使用 Step Functions 控制台中的数据流模拟器来测试 JSON 路径语法，以更好地了解在状态下如何操作数据，并查看数据在状态之间是如何传递的。

InputPath

使用 InputPath 选择状态输入的一部分。

例如，假设状态输入包括以下内容。

```
{
  "comment": "Example for InputPath.",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
```

```
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

您可以应用 InputPath。

```
"InputPath": "$.dataset2",
```

借助之前的 InputPath，以下内容是作为输入传递的 JSON。

```
{
  "val1": "a",
  "val2": "b",
  "val3": "c"
}
```

> **Note**
>
> 一条路径可以生成一系列值。考虑以下示例。
>
> ```
> { "a": [1, 2, 3, 4] }
> ```
>
> 如果您应用路径 $.a[0:2]，结果如下。
>
> ```
> [ 1, 2 ]
> ```

参数

本部分介绍您可以使用 Parameters 字段的不同方式。

键值对

使用 Parameters 字段创建作为输入传递的键值对集合。每个键值对的值可以是您在状态机定义中包含的静态值，也可以是使用路径从输入或上下文对象中选择的值。对于使用路径选择值的键值对，键名必须以 .$ 结尾。

例如，假设您提供以下输入。

```
{
  "comment": "Example for Parameters.",
  "product": {
    "details": {
       "color": "blue",
       "size": "small",
       "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

要选择一些信息，您可以在状态机定义中指定这些参数。

```
"Parameters": {
        "comment": "Selecting what I care about.",
        "MyDetails": {
          "size.$": "$.product.details.size",
          "exists.$": "$.product.availability",
          "StaticValue": "foo"
        }
      },
```

给定上一个输入和 Parameters 字段，这是传递的 JSON。

```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
      "size": "small",
      "exists": "in stock",
      "StaticValue": "foo"
  }
},
```

除了输入之外，您还可以访问一个特殊的 JSON 对象，称为上下文对象。上下文对象包含有关状态机执行的信息。请参阅 Context 对象。

连接的资源

Parameters 字段还可以将信息传递给连接的资源。例如，如果您的任务状态为编排 AWS Batch 任务，则可以将相关的 API 参数直接传递给该服务的 API 操作。有关更多信息，请参阅：

- 将参数传递给服务 API
- 使用其他服务

Amazon S3

如果您在各状态之间传递的 Lambda 函数数据可能增长到超过 262,144 字节，我们建议使用 Amazon S3 来存储数据，并实施以下方法之一：

- 在工作流中使用分布式 Map 状态，以便 Map 状态可以直接从 Amazon S3 数据来源读取输入。有关更多信息，请参阅 使用分布式模式下的 Map 状态。
- 解析 Payload 参数中存储桶的 Amazon 资源名称 (ARN)，以获取存储桶名称和键值。有关更多信息，请参阅 使用 Amazon S3 ARN 而不是传递大量有效负载。

或者，您也可以调整实施，以便在执行中传递较少的有效负载。

ResultSelector

在应用 ResultPath 之前，使用 ResultSelector 字段来操作状态的结果。ResultSelector 字段可用于创建键值对的集合，其中值为静态值或从状态的结果中选择。使用 ResultSelector 字段，您可以选择要将状态结果的哪些部分传递给 ResultPath 字段。

> **Note**
>
> 使用 ResultPath 字段，您可以将 ResultSelector 字段的输出添加到原始输入中。

ResultSelector 是处于以下状态的可选字段：

- Map
- 任务状态
- Parallel

例如，除了结果中的有效负载外，Step Functions 服务集成还会返回元数据。`ResultSelector` 可以选择结果的某些部分并使用 `ResultPath` 将其与状态输入合并。在此示例中，我们只想选择 `resourceType` 和 `ClusterId`，然后将其与来自 Amazon EMR createCluster.sync 的状态输入合并。给定以下内容：

```
{
  "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"
  }
}
```

然后，您可以使用 `ResultSelector` 选择 `resourceType` 和 `ClusterId`：

```
"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"
}
```

使用给定的输入，使用 `ResultSelector` 会产生：

```
{
  "OtherDataFromInput": {},
  "EMROutput": {
    "ResourceType": "elasticmapreduce",
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

展平由数组组成的数组

如果状态机中的 Parallel 或 Map 状态返回由数组组成的数组，则可以使用 ResultSelector 字段将他们转换为一个平面数组。您可以将此字段包含在 Parallel 或 Map 状态定义中，以操纵这些状态的结果。

要展平数组，请在 `ResultSelector` 字段中使用 JMESPath 语法 `[*]`，如下方的示例所示。

```
"ResultSelector": {
    "flattenArray.$": "$[*][*]"
  }
```

有关演示如何展平数组的示例，请参阅以下教程中的第 3 步：

- 使用 Lambda 函数处理整批数据
- 使用 Lambda 函数处理单个数据项

ResultPath

状态的输出可以是其输入的副本、其生成的结果（例如，`Task` 状态的 Lambda 函数的输出）或其输入和结果的组合。使用 `ResultPath` 可控制传递到状态输出的上述两种内容的组合。

以下状态类型可以生成结果且包含 `ResultPath`：

- Pass
- 任务状态
- Parallel
- Map

使用 `ResultPath` 可将任务结果与任务输入组合，或可选择二者之一。提供给 `ResultPath` 的路径控制将传递到输出的信息。

> **Note**
>
> ResultPath 限制为使用限制范围的引用路径，以便它可以仅标识 JSON 中的单一节点。请参阅 Amazon States Language 中的 引用路径。

这些示例基于创建使用 Lambda 的 Step Functions 状态机教程中所述的状态机和 Lambda 函数。演练此教程并通过在 ResultPath 字段中尝试不同的路径来测试不同的输出。

可使用 ResultPath 执行以下操作：

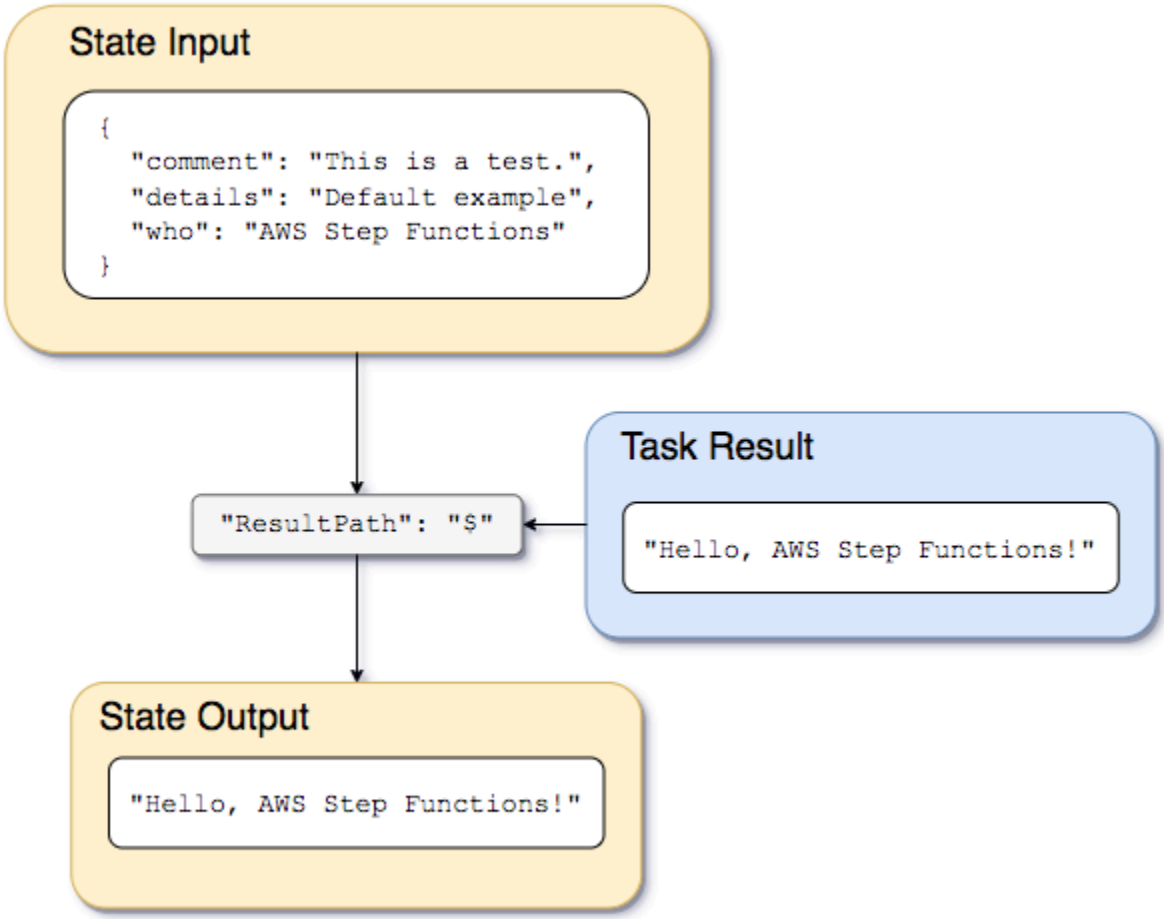
- 用于 ResultPath 用结果替换输入
- 丢弃结果并保留原始输入
- 用于 ResultPath 将结果包含在输入中
- ResultPath 用于使用结果更新输入中的节点
- ResultPath 用于在 a 中同时包含错误和输入 Catch

> **Tip**
>
> 使用 Step Functions 控制台中的数据流模拟器来测试 JSON 路径语法，以更好地了解在状态下如何操作数据，并查看数据在状态之间是如何传递的。

用于 ResultPath 用结果替换输入

如果没有指定 ResultPath，则默认行为是已指定 "ResultPath": "$"。由于这将指示状态将整个输入替换为结果，因此状态输入将由来自任务结果的结果完全替换。

下图显示了 ResultPath 如何将输入完全替换为任务的结果。



使用创建使用 Lambda 的 Step Functions 状态机中所述的状态机和 Lambda 函数，并将 Lambda 函数的服务集成类型更改为 AWS 开发工具包集成。为此，请在 Task 状态 Resource 字段中指定 Lambda 函数 Amazon 资源名称 (ARN)，如下例所示。使用 AWS SDK 集成可确保Task状态结果仅包含 Lambda 函数输出，不包含任何元数据。

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
         "Resource":"arn:aws:lambda:us-east-2:123456789012:function:HelloFunction",
         "End": true
      }
   }
}
```

然后，传递以下输入：

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 函数提供以下结果。

```
"Hello, AWS Step Functions!"
```

> **Tip**
>
> 您可以在 Step Functions 控制台上查看此结果。为此，请在控制台的执行详细信息 页面上，在图表视图中选择 Lambda 函数。然后，在 步骤详细信息 窗格中选择输出选项卡以查看此结果。

如果状态中未指定 `ResultPath`，或者已设置 `"ResultPath": "$"`，则状态的输入将由 Lambda 函数的结果取代，并且状态的输出为以下内容。

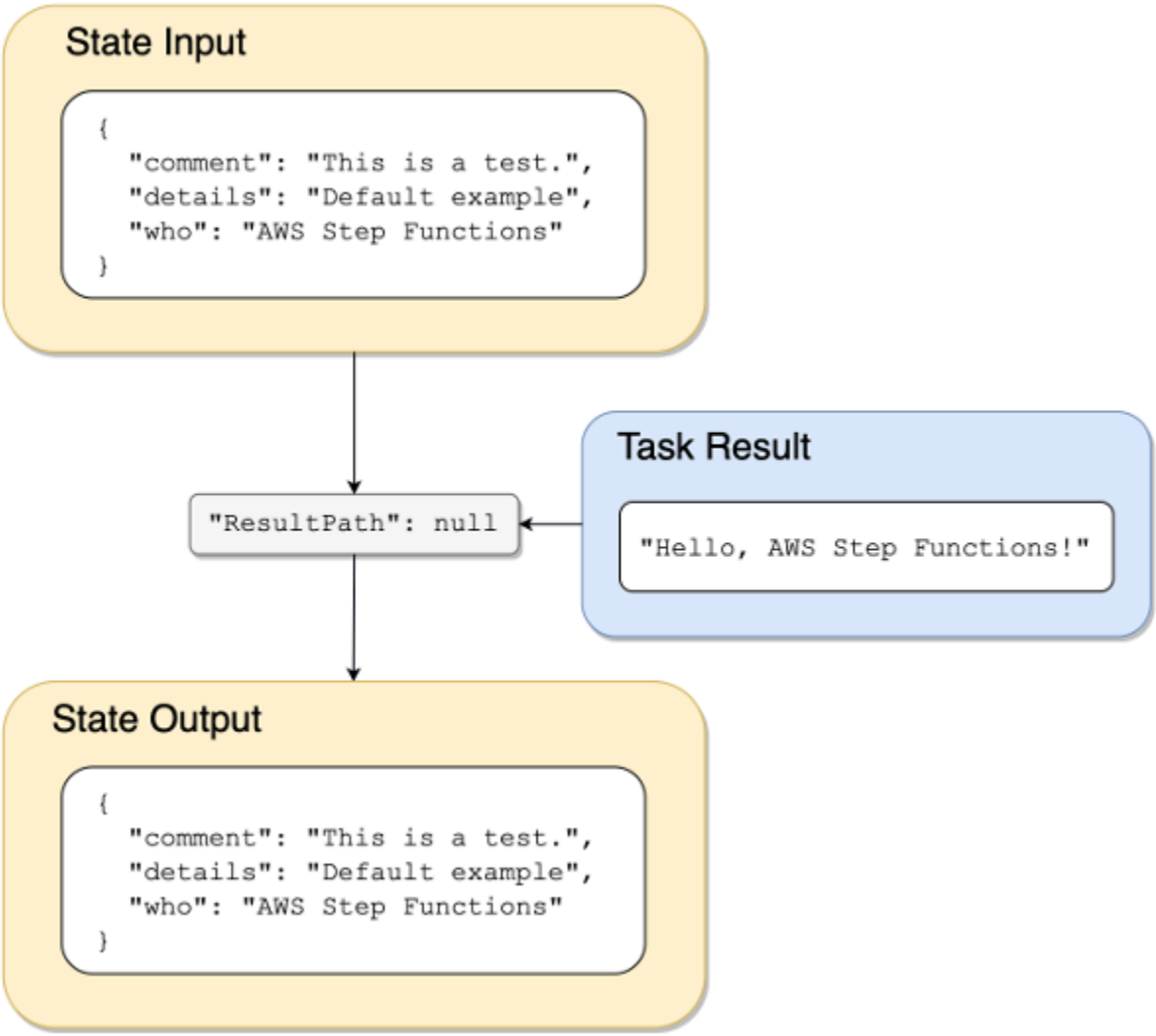
```
"Hello, AWS Step Functions!"
```

> **Note**
>
> `ResultPath` 用于包含结果内容与输入，然后再将其传递到输出。但是，如果未指定 `ResultPath`，则默认为替换整个输入。

丢弃结果并保留原始输入

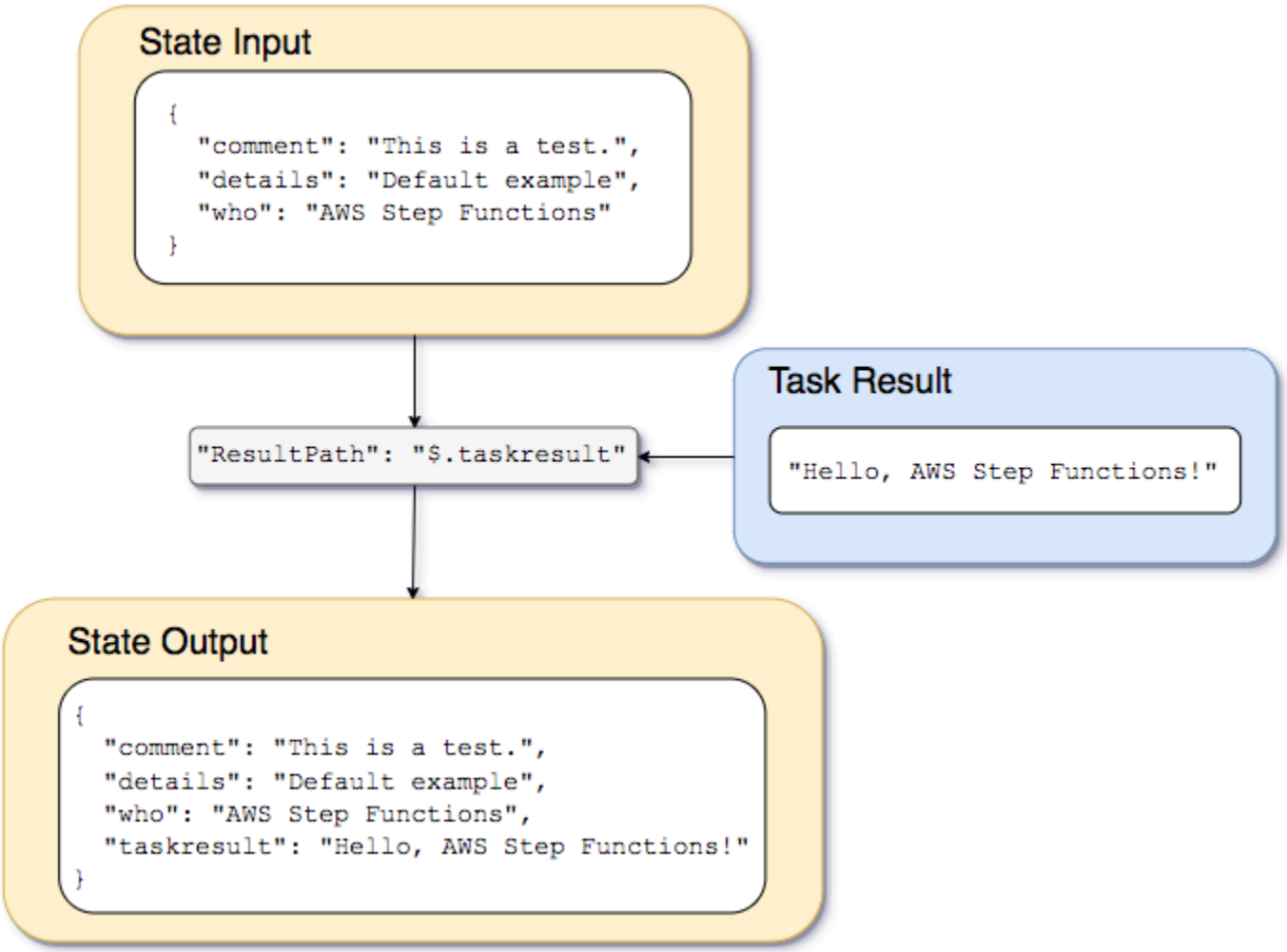
如果将 `ResultPath` 设置为 `null`，它会将原始输入传递给输出。使用 `"ResultPath": null` 时，状态的输入有效负载将直接复制到输出中，而不考虑结果。

下图显示了 null `ResultPath` 如何将输入直接复制到输出。



用于 ResultPath 将结果包含在输入中

下图显示了 `ResultPath` 如何包含结果与输入。



使用在创建使用 Lambda 的 Step Functions 状态机教程中介绍的状态机和 Lambda 函数，我们可以传递以下输入。

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 函数的结果为以下内容。

```
"Hello, AWS Step Functions!"
```

要保留输入，插入 Lambda 函数的结果，然后将组合的 JSON 传递到下一个状态，我们可以将 ResultPath 设置如下。

```
"ResultPath": "$.taskresult"
```

这包括 Lambda 函数的结果与原始输入。

```
{
  "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 函数的输出将作为 taskresult 的值追加到原始输入中。输入（包括新插入的值）将传递到下一个状态。

还可以将结果插入输入的子节点。将 ResultPath 设置为以下内容。

```
"ResultPath": "$.strings.lambdaresult"
```

使用以下输入开始执行。

```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
    "string2": "bar",
    "string3": "baz"
  },
  "who": "AWS Step Functions"
}
```

Lambda 函数的结果将作为输入中 strings 节点的子级插入。

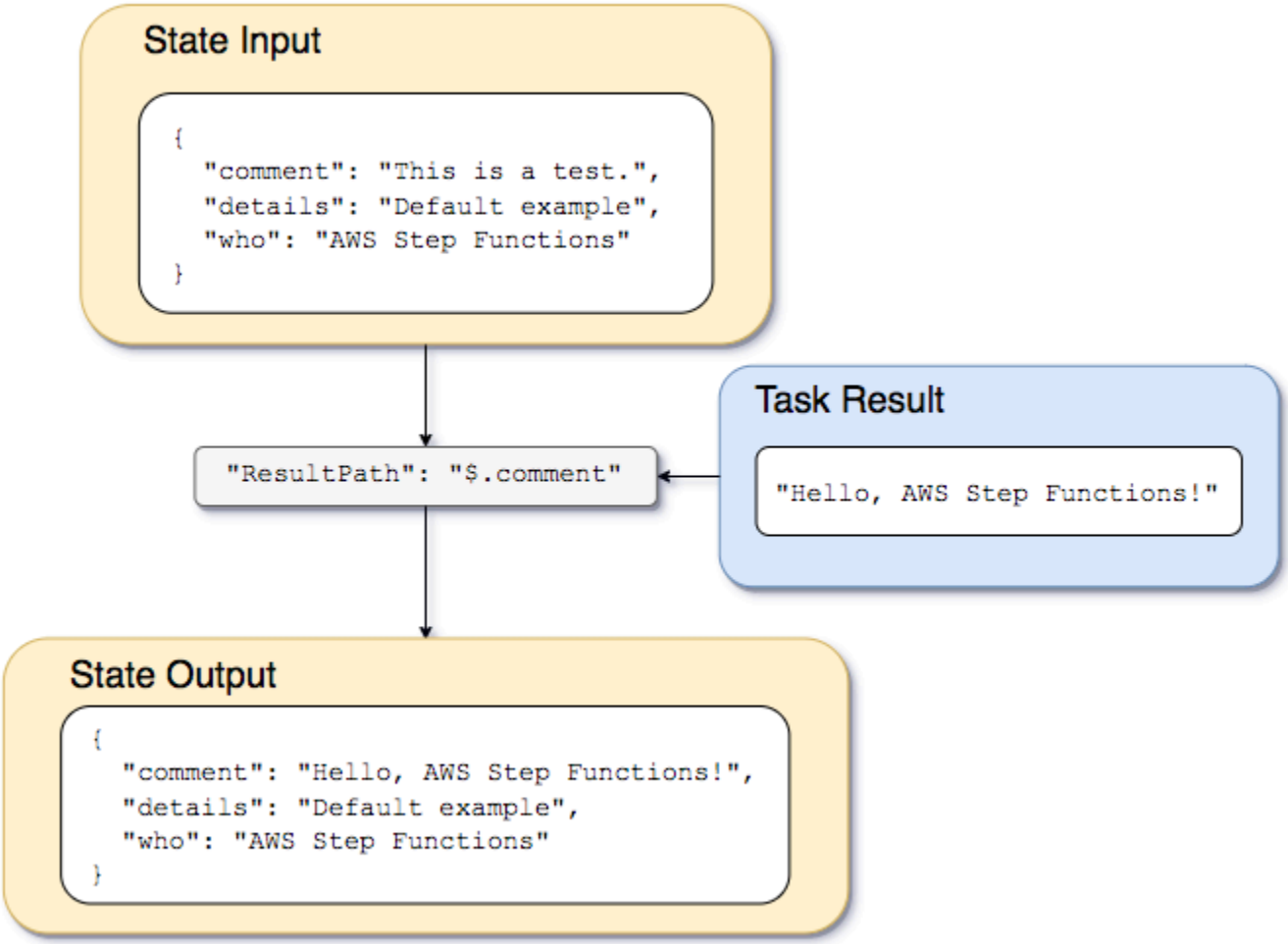
```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
    "string2": "bar",
```

```
    "string3": "baz",
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "who": "AWS Step Functions"
}
```

状态输出现在包含原始输入 JSON 与作为子节点的结果。

ResultPath 用于使用结果更新输入中的节点

下图显示了 `ResultPath` 如何将输入中现有 JSON 节点的值更新为任务结果中的值。



使用在创建使用 Lambda 的 Step Functions 状态机教程中介绍的状态机和 Lambda 函数示例，我们可以传递以下输入。

```
{
```

```
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 函数的结果为以下内容。

```
Hello, AWS Step Functions!
```

我们可覆盖现有节点，从而代替保留输入和将结果作为新节点插入 JSON。

例如，正如忽略或设置 "ResultPath": "$" 可覆盖整个节点一样，您可以指定单个节点来覆盖结果。

```
"ResultPath": "$.comment"
```

由于 comment 节点已存在于状态输入中，因此将 ResultPath 设置为 "$.comment" 会将输入中的该节点替换为 Lambda 函数的结果。如果未按 OutputPath 进一步筛选，则以下信息将传递到输出。

```
{
  "comment": "Hello, AWS Step Functions!",
  "details": "Default behavior example",
  "who": "AWS Step Functions",
}
```

comment 节点的值 "This is a test of the input and output of a Task state." 将由状态输出中 Lambda 函数的结果 "Hello, AWS Step Functions!" 替换。

ResultPath 用于在 a 中同时包含错误和输入 **Catch**

使用状态机处理 Step Functions 函数错误情形教程演示如何使用状态机捕获错误。在某些情况下，您可能希望保留原始输入与错误。在 Catch 中使用 ResultPath 可包含错误与原始输入，而不是替换它。

```
"Catch": [{
  "ErrorEquals": ["States.ALL"],
  "Next": "NextTask",
  "ResultPath": "$.error"
}]
```

如果上一 Catch 语句捕获错误，则它将在状态输入的 error 节点中包含结果。例如，对于以下输入：

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

捕获错误时，状态输出为以下内容。

```
{
  "foo": "bar",
  "error": {
    "Error": "Error here"
  }
}
```

有关错误处理的更多信息，请参阅以下内容：

- Step Functions 中的错误处理
- 使用状态机处理 Step Functions 函数错误情形

OutputPath

OutputPath 允许您选择状态输出的一部分以传递到下一个状态。这可让您筛选出不需要的信息，并且仅传递所需的 JSON 部分。

如果您未指定 OutputPath，则默认值是 $。这会将整个 JSON 节点（由状态输入、任务结果和 ResultPath 确定）传递到下一个状态。

> **Tip**
>
> 使用 Step Functions 控制台中的数据流模拟器来测试 JSON 路径语法，以更好地了解在状态下如何操作数据，并查看数据在状态之间是如何传递的。

有关更多信息，请参阅下列内容：

- Amazon States Language 中的路径
- InputPath、ResultPath 和 OutputPath 示例
- 将静态 JSON 作为参数传递

- Step Functions 中的输入和输出处理

InputPath、ResultPath 和 OutputPath 示例

除 Fail 或 Succeed 状态外的任何状态，都可以包含输入和输出处理字段，如 InputPath、ResultPath 或 OutputPath。此外，Wait 和 Choice 状态不支持 ResultPath 字段。通过这些字段，您可以使用 JsonPath 来筛选在工作流中移动的 JSON 数据。

您还可以使用 Parameters 字段在 JSON 数据在工作流中移动时对其进行操作。有关使用 Parameters 的信息，请参阅 InputPath、参数和 ResultSelector。

例如，从AWS Lambda教程中所述的 创建使用 Lambda 的 Step Functions 状态机 函数和状态机开始。修改状态机，以便它包含以下 InputPath、ResultPath 和 OutputPath。

```
{
  "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
  "StartAt": "HelloWorld",
  "States": {
    "HelloWorld": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
      "InputPath": "$.lambda",
      "ResultPath": "$.data.lambdaresult",
      "OutputPath": "$.data",
      "End": true
    }
  }
}
```

使用以下输入开始执行。

```
{
  "comment": "An input comment.",
  "data": {
    "val1": 23,
    "val2": 17
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
```

```
  }
}
```

假设 comment 和 extra 节点可以丢弃，但我们想要包含 Lambda 函数的输出以及保留 data 节点中的信息。

在更新后的状态机中，Task 状态将更改以处理任务输入。

```
"InputPath": "$.lambda",
```

状态机定义中的此行将任务输入限制为仅状态输入中的 lambda 节点。Lambda 函数仅接收输入形式的 JSON 对象 {"who": "AWS Step Functions"}。

```
"ResultPath": "$.data.lambdaresult",
```

此 ResultPath 指示状态机将 Lambda 函数的结果插入名为 lambdaresult、作为原始状态机输入中 data 节点的子级的节点中。因为我们没有使用 OutputPath 对原始输入和结果执行任何其他操作，所以现在状态的输出包括 Lambda 函数与原始输入的结果。

```
{
  "comment": "An input comment.",
  "data": {
    "val1": 23,
    "val2": 17,
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
  }
}
```

但是，我们的目标是仅保留 data 节点，并且包含 Lambda 函数的结果。OutputPath 将先筛选此组合的 JSON，然后再将其传递到状态输出。

```
"OutputPath": "$.data",
```

这将仅选择原始输入中要传递到输出的 data 节点（包括 ResultPath 插入的 lambdaresult 子节点）。状态输出将筛选为以下内容。

```
{
  "val1": 23,
  "val2": 17,
  "lambdaresult": "Hello, AWS Step Functions!"
}
```

在该 Task 状态中：

1. InputPath 仅将输入中的 lambda 节点发送至 Lambda 函数。
2. ResultPath 将结果作为 data 节点的子级插入原始输入。
3. OutputPath 将筛选状态输入（现在包含 Lambda 函数的结果），以便它仅将 data 节点传递到状态输出。

Example 使用 JsonPath 操作原始状态机输入、结果和最终输出

考虑以下状态机器，它可以验证保险申请人的身份和地址。

> **Note**
>
> 要查看完整的示例，请参阅 How to use JSON Path in Step Functions。

```
{
  "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
            }
          }
        }
      ]
    }
  }
}
```

如果您使用以下输入运行此状态机，则执行将失败，因为执行验证的 Lambda 函数只期望需要验证的数据作为输入。因此，您必须使用适当的 JsonPath 指定包含待验证信息的节点。

```
{
  "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
      },
    ]
  }
}
```

要指定 *check-identity* Lambda 函数必须使用的节点，请使用 InputPath 字段，如下所示：

```
"InputPath": "$.data.identity"
```

要指定 *check-address* Lambda 函数必须使用的节点，请使用 InputPath 字段，如下所示：

```
"InputPath": "$.data.address"
```

现在，如果要将验证结果存储在原始状态机输入中，请使用 ResultPath 字段，如下所示：

```
"ResultPath": "$.results"
```

但是，如果您只需要身份和验证结果并丢弃原始输入，请使用 OutputPath 字段，如下所示：

```
"OutputPath": "$.results"
```

有关更多信息，请参阅Step Functions 中的输入和输出处理。

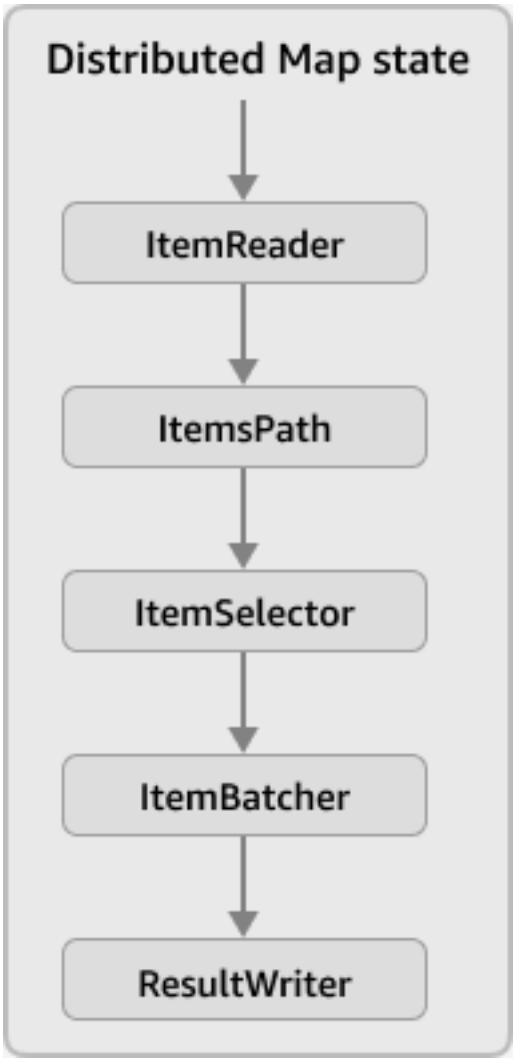
Map 状态输入和输出字段

Map 状态同时迭代数据集中的一个项集合，例如 JSON 数组、Amazon S3 对象列表或 Amazon S3 存储桶中 CSV 文件的行。它会为集合中的每个项重复一组步骤。您可以使用这些字段配置 Map 状态接收

的输入及其生成的输出。Step Functions 按以下列表和插图所示的顺序应用分布式 Map 状态中的每个字段：

> **Note**
>
> 根据您的使用案例，您可能不需要应用所有这些字段。

1. ItemReader
2. ItemsPath
3. ItemSelector
4. ItemBatcher
5. ResultWriter



> **Note**
>
> 这些 Map 状态输入和输出字段目前在 Step Functions 控制台的数据流模拟器中不可用。

ItemReader

`ItemReader` 字段是一个 JSON 对象，用于指定数据集及其位置。分布式 Map 状态使用此数据集作为其输入。以下示例显示了如果数据集是存储在 Amazon S3 存储桶中的 CSV 文件时，`ItemReader` 字段的语法内容。

```
"ItemReader": {
    "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "FIRST_ROW"
```

```
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
        "Bucket": "myBucket",
        "Key": "csvDataset/ratings.csv"
    }
}
```

> **Tip**
>
> 在 Workflow Studio 中，您可以在项目来源字段中指定数据集及其位置。

内容

- ItemReader 字段的内容
- 数据集示例
- 适用于数据集的 IAM 策略

ItemReader 字段的内容

根据您的数据集，`ItemReader` 字段的内容会有所不同。例如，如果您的数据集是从工作流的上一个步骤传递的 JSON 数组，则 `ItemReader` 字段将被省略。如果您的数据集是 Amazon S3 数据来源，则此字段包含以下子字段。

`ReaderConfig`

一个 JSON 对象，用于指定以下详细信息：

- `InputType`

  指定 Amazon S3 数据来源的类型，例如 CSV 文件、对象、JSON 文件或 Amazon S3 清单列表。在 Workflow Studio 中，您可以从项目来源字段下的 Amazon S3 项目来源下拉列表中选择一种输入类型。

- `CSVHeaderLocation`

  > **Note**
  >
  > 仅当使用 CSV 文件作为数据集时，才必须指定此字段。

接受以下值之一来指定列标题的位置：

> **Important**
>
> 目前，Step Functions 支持最大 10 KB 的 CSV 标题。

- `FIRST_ROW` – 如果文件的第一行是标题，则使用此选项。
- `GIVEN` – 使用此选项在状态机定义中指定标题。例如，如果您的 CSV 文件包含以下内容。

```
1,307,3.5,1256677221
1,481,3.5,1256677456
1,1091,1.5,1256677471
...
```

提供以下 JSON 数组作为 CSV 标题。

```
"ItemReader": {
    "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "GIVEN",
        "CSVHeaders": [
            "userId",
            "movieId",
            "rating",
            "timestamp"
        ]
    }
}
```

> **Tip**
>
> 在 Workflow Studio 中，您可以在项目来源字段的其他配置下找到此选项。

- `MaxItems`

限制传递给 Map 状态的数据项数量。例如，假设您提供的一个包含 1000 行的 CSV 文件并指定限制为 100。然后，解释器将只向 Map 状态传递 100 行。Map 状态从标题行之后开始，按顺序处理项。

默认情况下，Map 状态会迭代指定数据集中的所有项。

> **Note**
>
> 目前，您可以将上限指定为 1 亿。分布式 Map 状态将停止读取超过此限制的项。

> **Tip**
>
> 在 Workflow Studio 中，您可以在项目来源字段的其他配置下找到此选项。

或者，您可以在分布式 Map 状态输入中指定现有键值对的参考路径。此路径必须解析为正整数。您可以在 `MaxItemsPath` 子字段中指定参考路径。

> **Important**
>
> 您可以指定 `MaxItems` 或 `MaxItemsPath` 子字段，但不能同时指定两者。

`Resource`

Step Functions 必须根据指定的数据集调用的 Amazon S3 API 操作。

`Parameters`

一个 JSON 对象，用于指定存储数据集的 Amazon S3 存储桶名称和对象密钥。

> **Important**
>
> 确保您的 Amazon S3 存储桶与您的状态机处在同一个 AWS 账户和 AWS 区域下。

数据集示例

您可以指定下列选项之一作为数据集：

- 上一个步骤中的 JSON 数组
- Amazon S3 对象列表
- Amazon S3 存储桶中的 JSON 文件
- Amazon S3 存储桶中的 CSV 文件
- Amazon S3 清单列表

> **Important**
>
> Step Functions 需要适当的权限，才能访问您使用的Amazon S3 数据集。有关数据集的 IAM 政策信息，请参阅适用于数据集的 IAM 策略。

上一个步骤中的 JSON 数组

分布式 Map 状态可以接受从工作流的上一个步骤中传递的 JSON 输入。此输入必须是数组，或者必须包含特定节点内的数组。要选择包含数组的节点，可以使用 `ItemsPath` 字段。

要处理数组中的单个项，分布式 Map 状态会为每个数组项启动子工作流执行。以下选项卡显示了传递给 `Map` 状态的输入以及子工作流执行的相应输入的示例。

> **Note**
>
> 当您的数据集是上一个步骤中的 JSON 数组时，Step Functions 会省略 `ItemReader` 字段。

Input passed to the Map state

思考以下由三个项组成的 JSON 数组。

```
"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

分布式 Map 状态将启动三个子工作流执行。每次执行都会接收一个数组项作为输入。以下示例显示了子工作流执行接收的输入。

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 对象示例

分布式 Map 状态 可以迭代存储在 Amazon S3 存储桶中的对象。当工作流执行到达 Map 状态时，Step Functions 会调用 ListObjectsV2 API 操作，该操作会返回 Amazon S3 对象元数据的数组。在此数组中，每个项目都包含存储在存储桶中的数据，例如 ETag 和 Key。

要处理数组中的各个项目，分布式 Map 状态 会启动一个子工作流执行。例如，假设您的 Amazon S3 存储桶包含 100 张图片。然后，调用 ListObjectsV2 API 操作后返回的数组包含 100 个项目。然后，分布式 Map 状态 将启动 100 个子工作流执行，以处理每个数组项。

> **Note**
>
> - 目前，Step Functions 还会为您使用 Amazon S3 控制台在特定 Amazon S3 存储桶中创建的每个文件夹都提供一个项目。这会导致由分布式 Map 状态 启动额外的子工作流执行。为避免为该文件夹创建额外的子工作流执行，我们建议您使用 AWS CLI 来创建文件夹。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的高级别 Amazon S3 命令。
> - Step Functions 需要适当的权限，才能访问您使用的Amazon S3 数据集。有关数据集的 IAM 政策信息，请参阅适用于数据集的 IAM 策略。

以下选项卡显示了该数据集的 ItemReader 字段语法和传递给子工作流执行的输入的示例。

ItemReader syntax

在此示例中，您已将数据（包括图像、JSON 文件和对象）组织在名为 myBucket 的 Amazon S3 存储桶中名为 processData 的前缀内。

```
"ItemReader": {
    "Resource": "arn:aws:states:::s3:listObjectsV2",
    "Parameters": {
        "Bucket": "myBucket",
        "Prefix": "processData"
    }
}
```

Input passed to a child workflow execution

分布式 Map 状态 启动的子工作流执行与 Amazon S3 存储桶中存在的项目数量一样。以下示例显示了子工作流执行接收的输入。

```
{
  "Etag": "\"05704fbdccb224cb01c59005bebbad28\"",
  "Key": "processData/images/n02085620_1073.jpg",
  "LastModified": 1668699881,
  "Size": 34910,
  "StorageClass": "STANDARD"
}
```

Amazon S3 存储桶中的 JSON 文件

分布式 Map 状态 可以接受存储在 Amazon S3 存储桶中的 JSON 文件作为数据集。JSON 文件必须包含一个数组。

当工作流执行到达 Map 状态时，Step Functions 会调用 GetObject API 操作来获取指定的 JSON 文件。然后，Map 状态会迭代数组中的每个项目，并开始对每个项目执行子工作流。例如，如果您的 JSON 文件包含 1000 个数组项，则 Map 状态将启动 1000 个子工作流执行。

> **Note**
>
> - 用于启动子工作流执行的执行输入不能超过 256 KB。但是，如果您随后应用可选的 `ItemSelector` 字段来减小项目的大小，Step Functions 支持从 CSV 或 JSON 文件中读取最大 8 MB 的项目。
> - 目前，Step Functions 支持 Amazon S3 清单报告中单个文件的最大大小为 10 GB。但是，如果每个文件都小于 10 GB，Step Functions 能够处理大小可以超过 10 GB。
> - Step Functions 需要适当的权限，才能访问您使用的Amazon S3 数据集。有关数据集的 IAM 政策信息，请参阅适用于数据集的 IAM 策略。

以下选项卡显示了该数据集的 `ItemReader` 字段语法和传递给子工作流执行的输入的示例。

在此示例中，假设您有一个名为 *`factcheck.json`* 的 JSON 文件。您已将此文件存储在 Amazon S3 存储桶中的名为 *`jsonDataset`* 的前缀中。以下是 JSON 数据集的示例：

```
[
  {
    "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

分布式 Map 状态 启动的子工作流执行与 JSON 文件中存在的数组项数量一样多。以下示例显示了子工作流执行接收的输入。

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 存储桶中的 CSV 文件

分布式 Map 状态 可以接受存储在 Amazon S3 存储桶中的 CSV 文件作为数据集。如果您使用 CSV 文件作为数据集，则需要指定 CSV 列标题。有关如何指定 CSV 标题的信息，请参阅 ItemReader 字段的内容。

由于在 CSV 文件中没有创建和维护数据的标准化格式，因此 Step Functions 会根据以下规则解析 CSV 文件：

- 逗号 (,) 是用于分隔单个字段的分隔符。
- 换行符是分隔单个记录的分隔符。
- 字段被视为字符串。对于数据类型转换，使用 ItemSelector 中的 `States.StringToJson` 内置函数。
- 不需要双引号 (" ") 将字符串括起来。但是，用双引号括起来的字符串可以包含逗号和换行符，但不能用作分隔符。
- 通过重复双引号来转义双引号。
- 如果一行中的字段数少于标题中的字段数，Step Functions 会为缺失的值提供空字符串。
- 如果一行中的字段数大于标题中的字段数，Step Functions 会跳过多余的字段。

有关 Step Functions 如何解析 CSV 文件的更多信息，请参阅Example of parsing an input CSV file。

当工作流执行到达 Map 状态时，Step Functions 会调用 GetObject 操作来获取指定的 CSV 文件。然后，Map 状态会迭代 CSV 文件中的每一行，并启动一个子工作流执行以处理每行中的项目。例如，假设您提供了一个包含 100 行的 CSV 文件作为输入。然后，解释器将每一行传递给 Map 状态。Map 状态从标题行之后开始，按顺序处理项目。

> **Note**
>
> - 用于启动子工作流执行的执行输入不能超过 256 KB。但是，如果您随后应用可选的 ItemSelector 字段来减小项目的大小，Step Functions 支持从 CSV 或 JSON 文件中读取最大 8 MB 的项目。
> - 目前，Step Functions 支持 Amazon S3 清单报告中单个文件的最大大小为 10 GB。但是，如果每个文件都小于 10 GB，Step Functions 能够处理大小可以超过 10 GB。
> - Step Functions 需要适当的权限，才能访问您使用的Amazon S3 数据集。有关数据集的 IAM 政策信息，请参阅适用于数据集的 IAM 策略。

以下选项卡显示了该数据集的 ItemReader 字段语法和传递给子工作流执行的输入的示例。

ItemReader syntax

例如，假设您有一个名为 *ratings.csv* 的 CSV 文件。然后，您已将此文件存储在 Amazon S3 存储桶中名为 *csvDataset* 的前缀中。

```
{
  "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

分布式 Map 状态 启动的子工作流执行与 CSV 文件中存在的行数一样多，不包括标题行（如果文件有）。以下示例显示了子工作流执行接收的输入。

```
{
  "rating": "3.5",
  "movieId": "307",
  "userId": "1",
  "timestamp": "1256677221"
}
```

S3 清单示例

分布式 Map 状态 可以接受存储在 Amazon S3 存储桶中的 Amazon S3 清单文件作为数据集。

当工作流执行到达 Map 状态时，Step Functions 会调用 GetObject API 操作来获取指定的 Amazon S3 清单文件。然后，Map 状态会迭代清单中的对象，以返回 Amazon S3 清单对象元数据数组。

> **Note**
>
> - 目前，Step Functions 支持 Amazon S3 清单报告中单个文件的最大大小为 10 GB。但是，如果每个文件都小于 10 GB，Step Functions 能够处理大小可以超过 10 GB。
> - Step Functions 需要适当的权限，才能访问您使用的Amazon S3 数据集。有关数据集的 IAM 政策信息，请参阅适用于数据集的 IAM 策略。

以下是 CSV 格式的清单文件示例：此文件包含名为 csvDataset 和 imageDataset 的对象，其中存储在名为 sourceBucket 的 Amazon S3 存储桶中。

```
"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**
>
> 目前，Step Functions 不支持将用户定义的 Amazon S3 清单报告作为数据集。您还必须确保 Amazon S3 清单报告的输出格式为 CSV。有关 Amazon S3 清单及其设置方法的更多信息，请参阅《Amazon S3 用户指南》中的 Amazon S3 清单。

以下清单列表文件示例显示了清单对象元数据的 CSV 标题。

```
{
  "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"
  } ]
}
```

以下选项卡显示了该数据集的 `ItemReader` 字段语法和传递给子工作流执行的输入的示例。

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"
}
```

根据您在配置 Amazon S3 清单报告时选择的字段，您的 `manifest.json` 文件内容可能与所示示例有所不同。

适用于数据集的 IAM 策略

当您使用 Step Functions 控制台创建工作流时，Step Functions 可以根据工作流定义中的资源自动生成 IAM 策略。这些策略包括允许状态机角色调用分布式 Map 状态 的 `StartExecution` API 操作所需的最低权限。这些策略还包括 Step Functions 访问 AWS 资源（例如 Amazon S3 存储桶和对象以及 Lambda 函数）所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。例如，如果您的工作流包含分布式模式下的 `Map` 状态，则将策略范围缩小到包含您的数据集的特定 Amazon S3 存储桶和文件夹。

> **Important**
>
> 如果您在分布式 Map 状态 输入中指定了 Amazon S3 存储桶和对象或前缀，并将参考路径指向现有键值对，请务必更新工作流程的 IAM 策略。将策略范围缩小到运行时该路径解析到的存储桶和对象名称。

以下 IAM 策略示例授予使用 ListObjectsV2 和 GetObject API 操作访问 Amazon S3 数据集所需的最低权限。

Example Amazon S3 对象作为数据集的 IAM 策略

以下示例显示了一个 IAM 策略，该策略可授予访问名为 *myBucket* 的 Amazon S3 存储桶的 *processImages* 中组织的对象的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example 将 CSV 文件作为数据集的 IAM 政策

以下示例显示一个 IAM 策略，该策略授予可授予访问名为 *ratings.csv* 的 CSV 文件的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example Amazon S3 清单作为数据集的 IAM 策略

以下示例显示了一个 IAM 策略，可授予访问 Amazon S3 清单报告的最低权限。

```
{
    "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

使用 ItemsPath 字段在提供给 Map 状态的 JSON 输入中选择一个数组。Map 状态为数组中的每个项目重复一组步骤。默认情况下，Map 状态将 ItemsPath 设置为 $，这将选择整个输入。如果 Map 状态的输入是 JSON 数组，则它将对数组中的每个项目运行一次迭代，并将该项目作为输入传递给迭代。

> **Note**
>
> 只有在工作流中使用从上一个状态传递的 JSON 输入时，才能在分布式 Map 状态下使用 ItemsPath。

您可以使用 ItemsPath 字段指定输入值指向用于迭代的 JSON 数组的位置。ItemsPath 的值必须是一个参考路径，并且该路径必须指向 JSON 数组。例如，考虑 Map 状态的输入，其中包含两个数组，如以下示例所示。

```
{
  "ThingsPiratesSay": [
    {
      "say": "Avast!"
    },
    {
      "say": "Yar!"
    },
    {
```

```
      "say": "Walk the Plank!"
    }
  ],
  "ThingsGiantsSay": [
    {
      "say": "Fee!"
    },
    {
      "say": "Fi!"
    },
    {
      "say": "Fo!"
    },
    {
      "say": "Fum!"
    }
  ]
}
```

在这种情况下，您可以通过 ItemsPath 选择一个数组来指定用于 Map 状态迭代的数组。以下状态机定义使用 ItemsPath 来指定输入中的 ThingsPiratesSay 数组，然后对 ThingsPiratesSay 数组中的每个项目运行 SayWord 传递状态的迭代。

```
{
  "StartAt": "PiratesSay",
  "States": {
    "PiratesSay": {
      "Type": "Map",
      "ItemsPath": "$.ThingsPiratesSay",
      "ItemProcessor": {
         "StartAt": "SayWord",
         "States": {
           "SayWord": {
             "Type": "Pass",
             "End": true
           }
         }
      },
      "End": true
    }
  }
}
```

处理输入时，Map 状态在 InputPath 之后应用 ItemsPath。InputPath 筛选输出后，它将对状态的有效输入进行操作。

有关 Map 状态的更多信息，请参阅以下内容：

- Map 状态
- Map 状态处理模式
- 使用内联 Map 状态重复操作
- 内联 Map 状态输入和输出处理

ItemSelector

默认情况下，Map 状态的有效输入是原始状态输入中的一组单个数据项。使用 ItemSelector 字段，可在数据项的值传递到 Map 状态之前对其进行覆盖。要覆盖这些值，请指定包含键值对集合的有效 JSON 输入。这些键值对可以是状态机定义中提供的静态值，使用路径从状态输入中选择的值，或者从上下文对象访问的值。

如果您使用路径或上下文对象指定键值对，则键名必须以 .$ 结尾。

> **Note**
>
> ItemSelector 字段取代了 Map 状态内的 Parameters 字段。如果您在 Map 状态定义中使用 Parameters 字段来创建自定义输入，我们强烈建议您将其替换为 ItemSelector。

您可以在内联 Map 状态和分布式 Map 状态中指定 ItemSelector 字段。

例如，考虑以下 JSON 输入，其中包含 imageData 节点内三个项目的数组。对于每次 *Map* 状态迭代，都会将一个数组项作为输入传递给迭代。

```
[
  {
    "resize": "true",
    "format": "jpg"
  },
  {
    "resize": "false",
    "format": "png"
  },
  {
```

```
    "resize": "true",
    "format": "jpg"
  }
]
```

使用 ItemSelector 字段，您可以定义一个自定义 JSON 输入以覆盖原始输入，如以下示例所示。然后，Step Functions 将此自定义输入传递给每次 *Map* 状态迭代。自定义输入包含 size 的静态值和 Map 状态的上下文对象数据的值。$$.Map.Item.Value 上下文对象包含每个单独数据项的值。

```
{
  "ItemSelector": {
    "size": 10,
    "value.$": "$$.Map.Item.Value"
  }
}
```

以下示例显示了内联 Map 状态 的一次迭代所接收的输入：

```
{
  "size": 10,
  "value": {
    "resize": "true",
    "format": "jpg"
  }
}
```

> **Tip**
>
> 有关使用 ItemSelector 字段的分布式 Map 状态 的完整示例，请参阅开始使用分布式 Map 状态。

ItemBatcher

ItemBatcher 字段是一个 JSON 对象，用于指定在单个子工作流执行中处理一组项目。可在处理大型 CSV 文件或 JSON 数组，或大型 Amazon S3 对象集时，使用批处理。

以下示例显示了 ItemBatcher 字段的语法。在以下语法中，每个子工作流执行应处理的最大项目数设置为 100。

```
{
```

```
  "ItemBatcher": {
    "MaxItemsPerBatch": 100
  }
}
```

默认情况下，数据集中的每个项目都作为输入传递给单个子工作流执行。例如，假设您指定一个 JSON 文件作为输入，其中包含以下数组：

```
[
  {
    "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"
  },
  ...
]
```

对于给定的输入，每个子工作流执行都会接收一个数组项作为其输入。以下示例显示了一个子工作流执行的输入：

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

为了帮助优化处理任务的性能和成本，请选择能够平衡项目数量和项目处理时间的批次大小。如果使用批处理，Step Functions 会将这些项目添加到一个项目数组中。然后，它将数组作为输入传递给每个子工作流执行。以下示例显示了作为输入传递给子工作流执行的一批两个项目：

```
{
```

```
  "Items": [
    {
      "verdict": "true",
      "statement_date": "6/11/2008",
      "statement_source": "speech"
    },
    {
      "verdict": "false",
      "statement_date": "6/7/2022",
      "statement_source": "television"
    }
  ]
}
```

> **Tip**
>
> 要了解有关在工作流中使用 `ItemBatcher` 字段的更多信息，请尝试以下教程和研讨会：
>
> - 在 Lambda 函数中处理整批数据
> - 在子工作流执行中批量迭代项目
> - 《AWS Step Functions 研讨会》的《模块 14 - 数据处理》中的使用分布式映射进行大规模并行化

内容

- 用于指定项目批处理的字段

用于指定项目批处理的字段

要对项目进行批处理，请指定要批处理的最大项目数、最大批量大小，或者同事指定两者。您至少指定一个值才能批处理项目。

每个批次的最大项目数

指定每个子工作流执行处理的最大项目数。解释器将 `Items` 数组中批处理的项目数限制为该值。如果您同时指定批处理数量和大小，解释器会减少批次中的项目数以避免超过指定的批次大小限制。

如果您未指定此值但提供了最大批处理大小的值，Step Functions 会在不超过以字节为单位的最大批处理大小的情况下，在每个子工作流执行过程中处理尽可能多的项目。

例如，假设您使用一个包含 1130 个节点的输入 JSON 文件运行执行。如果您为每批指定的最大项目数为 100，Step Functions 将创建 12 个批次。其中，11 个批次各包含 100 个项目，而第十二批包含其余 30 个项目。

或者，您也可以将每个批次的最大项目指定为分布式 Map 状态输入中现有键值对的参考路径。此路径必须解析为正整数。

例如，给定以下输入：

```
{
  "maxBatchItems": 500
}
```

您可以使用参考路径指定批处理的最大项目数量，如下所示：

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxItemsPerBatchPath": "$.maxBatchItems"
    }
    ...
    ...
  }
}
```

> **Important**
>
> 您可以指定 `MaxItemsPerBatch` 或 `MaxItemsPerBatchPath` 子字段，但不能同时指定两者。

每批最大 KB 值

以字节为单位指定批次的最大大小，最大为 256 KB。如果您同时指定了最大批次数量和大小，Step Functions 会减少批次中的项目数量以避免超过指定的批量大小限制。

或者，您也可以将最大批量大小指定为分布式地图状态输入中现有键值对的参考路径。此路径必须解析为正整数。

> **Note**
>
> 如果您使用批处理但未指定最大批处理大小，则解释器会在每个子工作流执行中尽可能多地处理不超过 256 KB 的项目。

例如，给定以下输入：

```
{
  "batchSize": 131072
}
```

您可以使用参考路径指定最大批次大小，如下所示：

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxInputBytesPerBatchPath": "$.batchSize"
    }
    ...
    ...
  }
}
```

> **Important**
>
> 您可以指定 `MaxInputBytesPerBatch` 或 `MaxInputBytesPerBatchPath` 子字段，但不能同时指定两者。

批量输入

另外，您也可以指定固定的 JSON 输入，将其包含在传递给每个子工作流执行的每个批次中。Step Functions 将此输入与每个子工作流执行的输入合并。例如，给定以下关于项目数组的事实检查日期的固定输入：

```
"ItemBatcher": {
```

```
    "BatchInput": {
        "factCheck": "December 2022"
    }
}
```

每个子工作流执行都会收到以下内容作为输入：

```
{
  "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` 字段是一个 JSON 对象，用于指定 Step Functions 将分布式 Map 状态 启动的子工作流执行结果写入 Amazon S3 的位置。默认情况下，Step Functions 不会导出这些结果。

> **Important**
>
> 确保您用于导出 Map Run 结果的 Amazon S3 存储桶与状态机位于同一个 AWS 账户和 AWS 区域下。否则，您的状态机执行将因 `States.ResultWriterFailed` 错误而失败。

如果您的输出有效负载大小超过 256 KB，将结果导出到 Amazon S3 存储桶会很有帮助。Step Functions 整合了所有子工作流执行数据，例如执行输入和输出、ARN 和执行状态。然后，它将状态相同的执行导出到指定 Amazon S3 位置的相应文件中。以下示例显示了导出子工作流执行结果时

ResultWriter 字段的语法。在此示例中，您将结果存储在名为 myOutputBucket 存储桶种，该存储桶位于名为 csvProcessJobs 的前缀中。

```
{
  "ResultWriter": {
    "Resource": "arn:aws:states:::s3:putObject",
    "Parameters": {
      "Bucket": "myOutputBucket",
      "Prefix": "csvProcessJobs"
    }
  }
}
```

> **Tip**
>
> 在 Workflow Studio 中，您可以通过选择将 Map 状态结果导出到 Amazon S3 来导出子工作流执行结果。然后，提供您要将结果导出到其中的 Amazon S3 存储桶的名称和前缀。

Step Functions 需要适当的权限才能访问要导出结果的存储桶和文件夹。有关所需 IAM 策略的信息，请参阅 用于 ResultWriter 的 IAM 策略。

如果要导出子工作流执行结果，分布式 Map 状态执行将按以下格式返回 Map Run ARN 以及有关 Amazon S3 导出位置的数据：

```
{
  "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 将具有相同状态的执行结果导出到各自的文件中。例如，如果您的子工作流执行结果为 500 个成功和 200 个失败，则 Step Functions 将在指定的 Amazon S3 位置为成功和失败结果创建两个文件。在此示例中，成功结果文件包含 500 个成功结果，而失败结果文件包含 200 个失败结果。

对于给定的执行尝试，Step Functions 会根据您的执行输出在指定的 Amazon S3 位置创建以下文件：

- `manifest.json` – 包含 Map Run 元数据，例如导出位置、Map Run ARN 以及有关结果文件的信息。

  如果您redriven了 Map Run，`manifest.json` 文件会包含 Map Run 的所有尝试中所有成功子工作流执行的引用。但是，此文件包含对特定redrive的失败和待执行的引用。
- `SUCCEEDED_n.json` – 包含所有成功执行子工作流的合并数据。n 表示文件的索引号。索引号从 0 开始。例如，`SUCCEEDED_1.json`。
- `FAILED_n.json` – 包含所有失败、超时和中止的子工作流执行的合并数据。使用此文件从失败的执行中恢复。n 表示文件的索引。索引号从 0 开始。例如，`FAILED_1.json`。
- `PENDING_n.json` – 包含由于 Map Run 失败或中止而未启动的所有子工作流执行的合并数据。n 表示文件的索引。索引号从 0 开始。例如，`PENDING_1.json`。

Step Functions 支持最大为 5 GB 的单个结果文件。如果文件大小超过 5 GB，Step Functions 会创建另一个文件来写入超出部分的执行结果，并在文件名后面附加一个索引号。例如，如果 `Succeeded_0.json` 文件的大小超过 5 GB，Step Functions 会创建一个 `Succeeded_1.json` 文件来记录超出部分结果。

如果您未指定导出子工作流执行结果，则状态机执行将返回子工作流执行结果数组，如以下示例所示：

> **Note**
>
> 如果返回的输出大小超过 256 KB，则状态机执行失败并返回一个 `States.DataLimitExceeded` 错误。

```
[
  {
    "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"
    }
  },
  ...
]
```

用于 ResultWriter 的 IAM 策略

当您使用 Step Functions 控制台创建工作流时，Step Functions 可以根据工作流定义中的资源自动生成 IAM 策略。这些策略包括允许状态机角色调用分布式 Map 状态 的 StartExecution API 操作所需的最低权限。这些策略还包括 Step Functions 访问 AWS 资源（例如 Amazon S3 存储桶和对象以及 Lambda 函数）所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。例如，如果您的工作流包含分布式模式下的 Map 状态，则将策略范围缩小到包含您的数据集的特定 Amazon S3 存储桶和文件夹。

> **Important**
>
> 如果您在分布式 Map 状态 输入中指定了 Amazon S3 存储桶和对象或前缀，并将参考路径指向现有键值对，请务必更新工作流程的 IAM 策略。将策略范围缩小到运行时该路径解析到的存储桶和对象名称。

下面的 IAM 策略示例授予使用 PutObject API 操作将子工作流执行结果写入 Amazon S3 存储桶中名为 *csvJobs* 的文件夹所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::resultBucket/csvJobs/*"
            ]
```

```
        }
    ]
}
```

如果要将子工作流执行结果写入的 Amazon S3 存储桶使用 AWS Key Management Service (AWS KMS) 密钥加密，则必须在 IAM 策略中包含必要的 AWS KMS 权限。有关更多信息，请参阅AWS KMS key 加密的 Amazon S3 存储桶的 IAM 权限。

解析输入 CSV 文件

由于在 CSV 文件中没有创建和维护数据的标准化格式，因此 Step Functions 会根据以下规则解析 CSV 文件：

- 逗号 (,) 是用于分隔单个字段的分隔符。
- 换行符是分隔单个记录的分隔符。
- 字段被视为字符串。对于数据类型转换，使用 ItemSelector 中的 `States.StringToJson` 内置函数。
- 不需要双引号 (" ") 将字符串括起来。但是，用双引号括起来的字符串可以包含逗号和换行符，但不能用作分隔符。
- 通过重复双引号来转义双引号。
- 如果一行中的字段数少于标题中的字段数，Step Functions 会为缺失的值提供空字符串。
- 如果一行中的字段数大于标题中的字段数，Step Functions 会跳过多余的字段。

Example 解析输入 CSV 文件

假设您提供了一个名为 *myCSVInput.csv* 的 CSV 文件，其中包含一行作为输入。然后，您已将此文件存储在名为 *my-bucket* 的 Amazon S3 存储桶中。该 CSV 文件如下所示。

```
abc,123,"This string contains commas, a double quotation marks (""), and a newline (
)",{""MyKey"":""MyValue""},"[1,2,3]"
```

以下状态机读取此 CSV 文件，并使用 ItemSelector 转换某些字段的数据类型。

```
{
  "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)"
      }
    }
  }
}
```

当您运行这个状态机时，它会产生以下输出。

```
[
  {
    "MyNumbers": 123,
    "MyObject": {
      "MyKey": "MyValue"
    },
    "MyString": "This string contains commas, a double quote (\"), and a newline (\n)",
    "MyLetters": "abc",
    "MyArray": [
      1,
      2,
      3
    ]
  }
]
```

Context 对象

上下文对象是执行期间可用的内部 JSON 结构，包含有关状态机和执行的信息。这允许您的工作流访问有关其特定执行的信息。您可以从以下字段访问上下文对象：

- `InputPath`
- `OutputPath`
- `ItemsPath`（在 Map 状态下）
- `Variable`（在 Choice 状态下）
- `ResultSelector`
- `Parameters`
- 变量到变量的比较运算符

上下文对象格式

上下文对象包括有关状态机、状态、执行和任务的信息。此 JSON 对象包括每种类型数据的节点，并采用以下格式。

```
{
    "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"
    }
}
```

在执行期间，上下文对象将以访问它的 Parameters 字段的相关数据填充。如果 Parameters 字段超出任务状态，则 Task 字段的值为空。

如果您尚未redriven执行，则 RedriveCount 上下文对象的值为 0。此外，RedriveTime 上下文对象只有在redriven执行时才可用。如果您已redriven a Map Run，则 RedriveTime 上下文对象仅适用于标准类型的子工作流。对于使用快速类型的子工作流的redriven Map Run，RedriveTime 不可用。

来自正在运行的执行的内容包括以下格式的具体信息。

```
{
    "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**
>
> 有关与 Map 状态相关的上下文对象数据，请参阅Map 状态的上下文对象数据。

访问上下文对象

要访问上下文对象，首先通过将 `.$` 附加到末尾来指定参数名称，就像选择带路径的状态输入时一样。然后，要访问上下文对象数据（而不是输入），请在路径前加上 `$$.`。这告诉你 AWS Step Functions 使用路径在上下文对象中选择一个节点。

以下示例说明如何访问上下文对象，例如执行 ID、名称、开始时间和redrive计数。

- 检索执行 ARN 并将其传递给下游服务
- 访问处于 Pass 状态的执行开始时间和名称
- 访问执行的redrive次数

检索执行 ARN 并将其传递给下游服务

此示例 Task 状态使用路径进行检索并将执行 Amazon 资源名称 (ARN) 传递给 Amazon Simple Queue Service (Amazon SQS) 消息。

```
{
  "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"
  }
}
```

有关在调用集成服务时使用任务令牌的更多信息，请参阅等待具有任务令牌的回调。

访问处于 Pass 状态的执行开始时间和名称

```
{
  "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
    }
  }
}
```

访问执行的redrive次数

以下 Task 状态定义示例显示了如何访问执行的redrive次数。

```
{
  "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
}
```

Map 状态的上下文对象数据

处理 Map 状态时，上下文对象中还有两个附加项：Index 和 Value。对于每次 Map 状态迭代，Index 包含当前正在处理的数组项的索引号，同时 Value 包含正在处理的数组项。在 Map 状态内，上下文对象包括以下数据：

```
"Map": {
   "Item": {
      "Index": Number,
      "Value": "String"
   }
}
```

这些项仅在 Map 状态下可用，并且可以在 ItemSelector 字段中指定。

> **Note**
>
> 您必须在主要 Map 状态的 ItemSelector 块中的上下文对象中定义参数，而不是在 ItemProcessor 部分中包含的状态中定义参数。

给定具有简单 Map 状态的状态机，我们可以按如下方式从上下文对象中注入信息。

```
{
  "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
    }
  }
}
```

如果使用以下输入执行前一状态机，则将 Index 和 Value 插入到输出中。

```
[
  {
    "who": "bob"
  },
  {
    "who": "meg"
  },
  {
    "who": "joe"
  }
]
```

执行的输出返回三次迭代中每次迭代的 Index 和 Value 项的值，如下所示：

```
[
  {
    "ContextIndex": 0,
    "ContextValue": {
      "who": "bob"
    }
  },
  {
    "ContextIndex": 1,
    "ContextValue": {
      "who": "meg"
    }
  },
  {
```

```
    "ContextIndex": 2,
    "ContextValue": {
      "who": "joe"
    }
  }
]
```

数据流模拟器

您可以在 Step Functions 控制台中设计、实施和调试工作流。您还可以使用 JsonPath 输入和输出数据处理来控制工作流中的数据流。使用数据流模拟器，您可以模拟工作流中的 任务状态 状态在运行时处理数据的顺序。使用模拟器，您可以了解如何在数据从一种状态流向另一种状态时对其进行筛选和操作。它能够模拟 Step Functions 用来处理和控制 JSON 数据流的以下每个字段：

InputPath

选择将整个输入有效负载的哪个 部分用作任务的输入。如果您指定了此字段，Step Functions 将首先应用此字段。

##

指定在调用任务之前输入的外观。使用 Parameters 字段，您可以创建一个键值对的集合，这些键值对作为输入传递给 AWS 服务集成（例如 AWS Lambda 函数）。这些值可以是静态的，也可以从状态输入或工作流上下文对象中动态选择。

ResultSelector

确定从任务的输出中选择什么。使用 ResultSelector 字段，您可以创建一个键值对的集合，这些键值对用于替换状态的结果并将该集合传递给 ResultPath。

ResultPath

确定任务输出的放置位置。使用 ResultPath 来确定状态的输出是其输入的副本、其产生的结果还是两者的组合。

OutputPath

确定发送到下一个状态的内容。使用 OutputPath，您可以筛选出不需要的信息，并且仅传递所需的 JSON 数据。

本主题内容

- 使用数据流模拟器
- 使用数据流模拟器的注意事项

使用数据流模拟器

模拟器可在您应用输入和输出数据处理字段之前和之后，对您的数据进行实时、并排的比较。要使用模拟器，请指定一个 JSON 输入。然后，通过每个输入和输出处理字段对其进行评估。模拟器会自动验证您的 JSON 输入并突出显示所有语法错误。

使用数据流模拟器的操作步骤

在以下步骤中，您将提供 JSON 输入并应用 `InputPath` 和 `##` 字段。您还可以应用其他可用字段并查看其输出。

1. 打开 Step Functions 控制台。
2. 在导航窗格中，选择数据流模拟器。
3. 在状态输入区域中，将预先填充的示例 JSON 数据替换为以下 JSON 数据。然后选择下一步。

```
{
  "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. 在 InputPath 中，输入 **`$.data.address`**，选择输入 JSON 数据的地址节点。

   InputPath 之后的状态输入框将显示以下结果。

```
{
```

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

5. 选择下一步。
6. 应用 Parameters 字段将生成的 JSON 数据转换为字符串。要转换数据，请执行以下操作：
   - 在参数框中，输入以下代码，创建名为 addressString 的字符串。

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

7. 在参数之后的已筛选输入框中查看 Parameters 字段应用的结果。

使用数据流模拟器的注意事项

在使用数据流模拟器之前，请考虑其局限性，包括但不限于：

- 不受支持的筛选表达式

  模拟器中的筛选表达式的行为方式与 Step Functions 服务中的筛选表达式不同。这包括使用以下语法的表达式：[?(expression)]. 以下是不受支持的表达式列表。如果使用这些表达式，在评估后可能不会返回预期的结果。
  - $..book[?(@.isInStock==true)]
  - $..book[?(@.price > 10.0)].title
- 单项数组的 JsonPath 评估不正确

  如果您使用返回单个数组项的 JsonPath 表达式筛选数据，则模拟器会返回不带数组的项目。例如，考虑以下数据数组，名为 items：

```
{
  "items": [
    {
      "name": "shoe",
      "color": "blue",
```

```
      "comment": "nice shoe"
    },
    {
      "name": "hat",
      "color": "red"
    },
    {
      "name": "shirt",
      "color": "yellow"
    }
  ]
}
```

给定这个 `items` 数组，如果您在 InputPath 字段中输入 **`$..comment`**，您会得到以下输出：

```
[
  "nice shoe"
]
```

但是，数据流模拟器会返回以下输出：

```
"nice shoe"
```

对于包含多个项目的数组的 JSONPath 评估，模拟器会返回预期输出。

使用版本与别名功能管理持续部署

您可以使用 Step Functions 通过状态机版本 与别名 功能管理工作流的持续部署。版本 是您可以运行的状态机快照，带编号且不可变。别名 是指向最多两个版本的状态机的指针。

您可以维护状态机的多个版本，并在生产工作流中管理其部署。使用别名，您可以在不同的工作流版本之间路由流量，并逐步将这些工作流部署到生产环境。

此外，您还可以使用版本或别名启动状态机执行。如果您在开始执行状态机时不使用版本或别名，Step Functions 将使用状态机定义的最新版本。

> **状态机修订版**
>
> 状态机可以有一个或多个修订版。使用 UpdateStateMachine API 操作更新状态机时，会创建一个新的状态机修订版。修订版 是状态机定义和配置的不可变的只读快照。您无法从修订

版中开始执行状态机，并且修订版没有 ARN。修订版有一个 `revisionId` 通用唯一标识符 (UUID)。

内容

- 状态机版本
- 状态机别名
- 版本与别名功能的授权
- 将状态机执行与版本或别名关联
- 别名和版本部署示例
- 逐步部署状态机版本

状态机版本

版本 是状态机快照，带编号且不可变。您可以从该状态机的最新修订版开始发布版本。每个版本都具有唯一的 Amazon 资源名称 (ARN)。此 ARN 是状态机 ARN 和版本号的组合，用冒号 (:) 分隔。以下示例显示状态机版本 ARN 的格式。

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:1
```

要开始使用状态机版本，必须发布第一个版本。发布版本后，您可以使用版本 ARN 调用 StartExecutionAPI 操作。您无法编辑版本，但可以更新状态机并发布新版本。您也可以发布多个状态机版本。

发布状态机的新版本时，Step Functions 会为其分配一个版本号。版本号从 1 开始，每个新版本都会单调递增。对于给定的状态机，版本号不会重复使用。如果您删除了状态机的版本 10，然后发布了新版本，Step Functions 会将其发布为版本 11。

状态机的所有版本的以下属性都相同：

- 状态机的所有版本共享相同的类型（标准或快速）。
- 您无法更改不同版本之间状态机的名称或创建日期。
- 标签全局适用于状态机。您可以使用TagResource和 UntagResourceAPI 操作管理状态机的标签。

状态机还包含作为每个版本和revision一部分的属性，但这些属性在两个给定版本或修订版之间可能不同。这些属性包括状态机定义、IAM 角色、跟踪配置和日志配置。

内容

- 发布状态机版本（控制台）
- 使用 Step Functions API 操作管理版本
- 从控制台运行状态机版本

发布状态机版本（控制台）

您最多可以发布 1000 个版本的状态机。要请求提高此软限制，请使用 AWS Management Console中的支持中心页面。您可以从控制台手动删除未使用的版本，也可以通过调用 DeleteStateMachineVersionAPI 操作来删除未使用的版本。

发布状态机版本的操作步骤

1. 打开 Step Functions 控制台，然后选择一个现有的状态机。
2. 在状态机详细信息页面上，选择编辑。
3. 根据需要编辑状态机定义，然后选择保存。
4. 选择发布版本。
5. （可选）在出现的对话框的描述字段中，输入有关状态机版本的简短描述。
6. 选择发布。

> **Note**
>
> 发布状态机的新版本时，Step Functions 会为其分配一个版本号。版本号从 1 开始，每个新版本都会单调递增。对于给定的状态机，版本号不会重复使用。如果您删除了状态机的版本 10，然后发布了新版本，Step Functions 会将其发布为版本 11。

使用 Step Functions API 操作管理版本

Step Functions 提供了以下 API 操作来发布和管理状态机版本：

- PublishStateMachineVersion— 从状态revision机的当前版本发布一个版本。
- UpdateStateMachine— 如果您更新状态机并在同一请求`true`中将`publish`参数设置为，则发布新的状态机版本。
- CreateStateMachine— 如果将`publish`参数设置为，则发布状态机的第一个修订版`true`。
- ListStateMachineVersions— 列出指定状态机 ARN 的版本。
- DescribeStateMachine— 返回中指定的版本 ARN 的状态机版本详情。`stateMachineArn`
- DeleteStateMachineVersion— 删除状态机版本。

要*`myStateMachine`*使用发布名为的状态机的当前版本中的新版本 AWS Command Line Interface，请使用以下`publish-state-machine-version`命令：

```
aws stepfunctions publish-state-machine-version --state-machine-arn arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
```

该响应返回 `stateMachineVersionArn`。例如，前面的命令返回的响应为 `arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:1`。

> **Note**
>
> 发布状态机的新版本时，Step Functions 会为其分配一个版本号。版本号从 1 开始，每个新版本都会单调递增。对于给定的状态机，版本号不会重复使用。如果您删除了状态机的版本 10，然后发布了新版本，Step Functions 会将其发布为版本 11。

从控制台运行状态机版本

要开始使用状态机版本，必须先从当前状态机revision中发布一个版本。要发布版本，请使用 Step Functions 控制台或调用 PublishStateMachineVersionAPI 操作。您还可以使用名为的可选参数来调用 UpdateStateMachineAliasAPI 操作`publish`来更新状态机并发布其版本。

您可以使用控制台或调用 StartExecutionAPI 操作并提供版本 ARN 来开始执行某个版本。您也可以使用别名开始执行某个版本。别名会根据其路由配置，会将流量路由到特定版本。

如果不使用版本就启动状态机执行，Step Functions 将使用状态机的最新版本进行执行。有关 Step Functions 如何将执行与版本关联的信息，请参阅将执行与版本或别名关联。

使用一个状态机版本启动执行

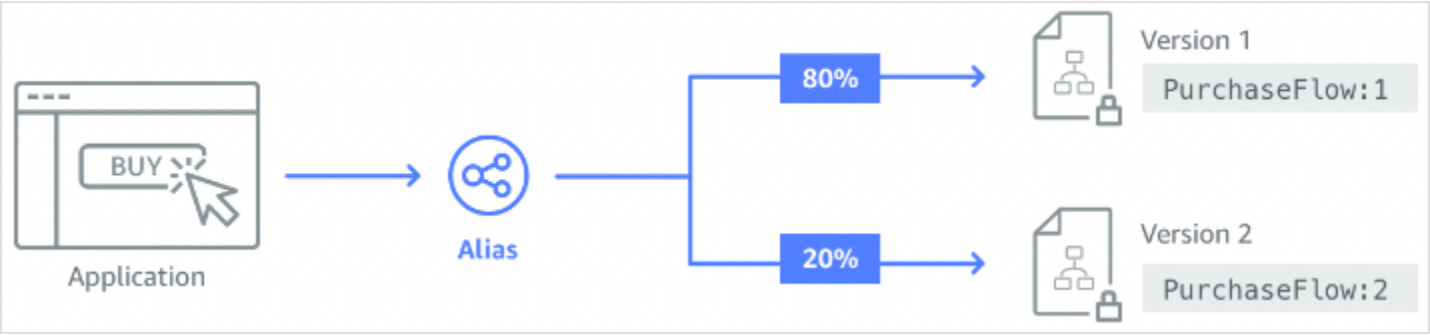
1. 打开 Step Functions 控制台，然后选择已为其发布一个或多个版本的现有状态机。要了解如何发布版本，请参阅发布状态机版本（控制台）。
2. 在状态机详细信息页面上，选择版本选项卡。
3. 在设置部分，执行以下操作：
   a. 选择要启动执行的版本。
   b. 选择启动执行。
4. （可选）在启动执行对话框中，输入执行名称。
5. （可选），输入执行输入，然后选择启动执行。

状态机别名

别名 是指向同一状态机最多两个版本的指针。您可以为状态机创建多个别名。每个别名都有唯一的 Amazon 资源名称 (ARN)。别名 ARN 是状态机的 ARN 和别名名称的组合，用冒号 (:) 分隔。以下示例显示状态机别名 ARN 的格式。

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:aliasName
```

您可以使用别名在两个状态机版本之一之间路由流量。您也可以创建指向单个版本的别名。别名只能指向状态机版本。您不能使用别名指向另一个别名。您也可以更新别名以便指向状态机的其他版本。



内容

- 创建状态机别名（控制台）
- 使用 Step Functions API 操作管理别名

- 别名路由配置
- 使用别名运行状态机（控制台）

创建状态机别名（控制台）

您可以使用 Step Functions 控制台或调用 CreateStateMachineAliasAPI 操作为每台状态机创建最多 100 个别名。要请求提高此软限制，请使用 AWS Management Console中的支持中心页面。从控制台或通过调用 DeleteStateMachineAliasAPI 操作来删除未使用的别名。

创建状态机别名的操作步骤

1. 打开 Step Functions 控制台，然后选择一个现有的状态机。
2. 在状态机详细信息页面上，选择别名选项卡。
3. 选择创建新别名。
4. 在创建别名页面上，执行以下操作：
   a. 输入别名名称。
   b. （可选）输入别名的描述。
5. 要在别名上配置路由，请参阅别名路由配置。
6. 选择创建别名。

使用 Step Functions API 操作管理别名

Step Functions 提供了以下 API 操作，您可以使用这些操作来创建和管理状态机别名或获取有关别名的信息：

- CreateStateMachineAlias— 为状态机创建别名。
- DescribeStateMachineAlias— 返回有关状态机别名的详细信息。
- ListStateMachineAliases— 列出指定状态机 ARN 的别名。
- UpdateStateMachineAlias— 通过修改现有状态机别名的`description`或来更新其配置`routingConfiguration`。
- DeleteStateMachineAlias— 删除状态机版本。

要创建*PROD*指向*myStateMachine*使用命名的状态机的版本 1 的别名 AWS Command Line Interface，请使用`create-state-machine-alias`命令。

```
aws stepfunctions create-state-machine-alias --name PROD  --routing-
configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
```

别名路由配置

您可以使用别名在状态机的两个版本之间路由执行流量。例如，假设您要启动状态机的一个新版本。通过在别名上配置路由，可以降低部署新版本所涉及的风险。通过配置路由，您可以将大部分流量发送到状态机的早期测试过的版本上。然后，新版本可以接收较小的百分比，直到您可以确认可以安全地推出新版本。

要定义路由配置，请确保已发布别名指向的两个状态机版本。从别名开始执行时，Step Functions 会从路由配置中指定的版本中随机选择要运行的状态机版本。选择的依据是您在别名路由配置中为每个版本分配的流量百分比。

在别名上配置路由配置的操作步骤

- 在创建别名页面的路由配置下，执行以下操作：

  a. 对于版本，选择别名指向的第一个状态机版本。

  b. 选中在两个版本之间拆分流量复选框。

  > **Tip**
  >
  > 要指向单个版本，请取消选中在两个版本之间拆分流量复选框。

  c. 对于版本，选择别名必须指向的第二个版本。

  d. 在流量百分比字段中，指定要路由到每个版本的流量百分比。例如，输入**60** 和 **40** 表示将 60% 的执行流量路由到第一个版本，40% 的流量路由到第二个版本。

     合并流量百分比必须等于 100%。

使用别名运行状态机（控制台）

您可以使用控制台中的别名启动状态机执行，也可以使用别名的 ARN 调用 StartExecutionAPI 操作。Step Functions 会运行由别名指定的版本。默认情况下，如果在启动状态机执行时没有指定版本或别名，Step Functions 会使用最新的修订版。

使用别名启动状态机执行的操作步骤

1. 打开 Step Functions 控制台，然后选择一个已创建别名的现有状态机。有关创建别名的信息，请参阅创建状态机别名（控制台）。
2. 在状态机详细信息页面上，选择别名选项卡。
3. 在别名部分中，执行以下操作：
   a. 选择要启动执行的别名。
   b. 选择启动执行。
4. （可选）在启动执行对话框中，输入执行名称。
5. 如果需要，输入执行输入，然后选择启动执行。

版本与别名功能的授权

要使用版本或别名调用 Step Functions API 操作，您需要适当的权限。要授权某个版本或别名调用 API 操作，Step Functions 会使用状态机的 ARN，而不是使用版本 ARN 或别名 ARN。您也可以缩小特定版本或别名的权限范围。有关更多信息，请参阅缩小权限范围。

您可以使用以下名为 *myStateMachine* 的状态机的 IAM 策略示例，调用 CreateStateMachineAlias 操作来创建状态机别名。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "states:CreateStateMachineAlias",
      "Resource": "arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine"
    }
  ]
}
```

在设置允许或拒绝访问使用状态机版本或别名的 API 操作权限时，请考虑以下几点：

- 如果使用 CreateStateMachine 和 UpdateStateMachine API 操作的 `publish` 参数来发布新的状态机版本，还需要在 PublishStateMachineVersion API 操作上设置 `ALLOW` 权限。
- DeleteStateMachine API 操作会删除与状态机关联的所有版本和别名。

本主题内容

- 缩小版本或别名的权限范围

缩小版本或别名的权限范围

您可以使用限定符进一步缩小版本或别名所需的授权权限范围。限定符指的是版本号或别名名称。您可以使用限定符来限定状态机。以下示例是使用名为 PROD 的别名作为限定符的状态机 ARN。

```
arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
```

有关限定和非限定 ARN 的更多信息，请参阅将执行与版本或别名关联。

在 IAM 策略的 Condition 语句中，使用名为 states:StateMachineQualifier 的可选上下文密钥缩小权限范围。例如，以下 IAM 策略针对名为 myStateMachine 的状态机，拒绝调用别名为 PROD 或版本 1 的DescribeStateMachine API 操作。

```
{
  "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"
          ]
        }
      }
    }
  ]
}
```

以下列表说明了您可以使用 StateMachineQualifier 上下文密钥缩小权限范围的 API 操作。

- CreateStateMachineAlias
- DeleteStateMachineAlias

- DeleteStateMachineVersion
- DescribeStateMachine
- DescribeStateMachineAlias
- ListExecutions
- ListStateMachineAliases
- StartExecution
- 开始同步执行
- UpdateStateMachineAlias

将状态机执行与版本或别名关联

Step Functions 根据您用来调用 StartExecutionAPI 操作的亚马逊资源名称 (ARN) 将执行与版本或别名关联起来。Step Functions 在执行开始时进行此操作。

您可以使用限定或非限定的 ARN 来启动状态机。

- 限定 ARN – 指状态机 ARN，后缀为版本号或别名。

  以下限定的 ARN 示例指的是名为 `myStateMachine` 的状态机的版本 3。

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:3
  ```

  以下限定的 ARN 示例指的是名为 `myStateMachine` 的状态机的 `PROD` 别名。

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
  ```

- 非限定 ARN – 指没有版本号或别名后缀的状态机 ARN。

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
  ```

例如，如果您的限定 ARN 指的是版本 3，则 Step Functions 会将执行与该版本相关联。它不会将执行与任何指向版本 3 的别名相关联。

如果您的限定 ARN 指的是别名，则 Step Functions 会将执行与该别名以及该别名指向的版本相关联。一次执行只能与一个别名关联。

> **Note**
>
> 如果您使用非限定的 ARN 启动执行，即使版本使用相同的状态机revision，Step Functions 也不会将该执行与版本相关联。例如，如果版本 3 使用最新的修订版，但您使用非限定的 ARN 启动执行，则 Step Functions 不会将该执行与版本 3 相关联。

本主题内容

- 查看使用版本或别名启动的执行

查看使用版本或别名启动的执行

Step Functions 提供了以下方式，您可以通过这些方式查看使用版本或别名启动的执行：

使用 API 操作

您可以通过调用DescribeExecution和 ListExecutionsAPI 操作来查看与版本或别名关联的所有执行。这些 API 操作会返回用于启动执行的版本或别名的 ARN。这些操作还会返回其他详细信息，包括执行的状态和 ARN。

您还可以提供状态机别名 ARN 或版本 ARN，以便列出与特定别名或版本关联的执行。

以下 ListExecutionsAPI 操作的示例响应显示了用于启动名为的状态机执行的别名的 ARN。*myFirstExecution*

以下代码段中的*##*文本表示特定于资源的信息。

```
{
    "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"
```

```
        }
    ]
}
```

使用 Step Functions 控制台

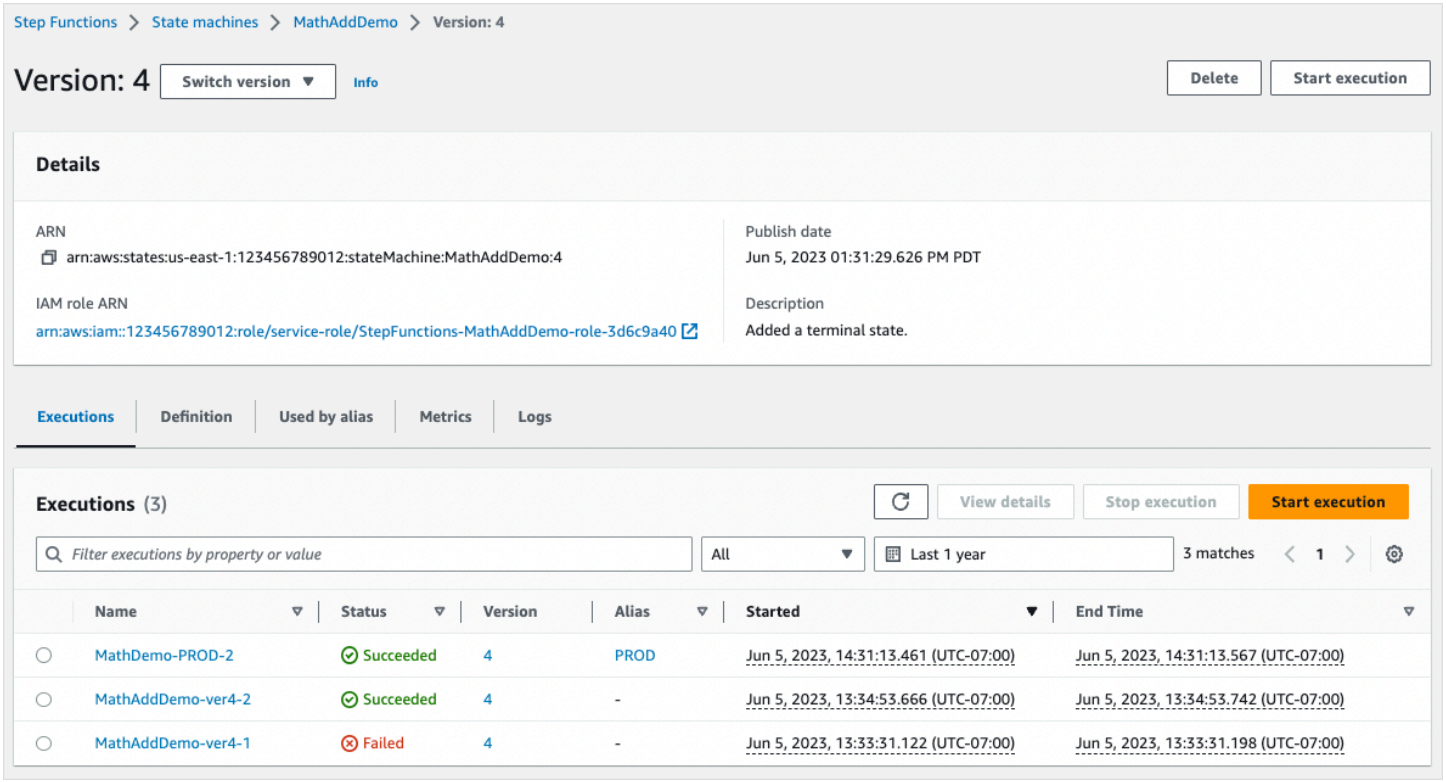
您还可以从 Step Functions 控制台中查看由版本或别名启动的执行。以下步骤显示如何查看使用特定版本启动的执行：

1. 打开 Step Functions 控制台，然后选择一个已发布版本或创建别名的现有状态机。此示例说明如何查看使用特定状态机版本启动的执行。
2. 选择版本选项卡，然后从版本列表中选择一个版本。

> **Tip**
>
> 按属性或值框筛选，搜索特定版本。

3. 在版本详细信息页面上，您可以看到使用所选版本启动的所有正在进行和过去的状态机执行列表。

下图显示了版本详细信息 控制台页面。该页面列出了由名为 *MathAddDemo* 的状态机版本 4 启动的执行。此列表还显示由名为 *PROD* 的别名启动的执行。此别名将执行流量路由到版本 4。



使用 CloudWatch 指标

对于使用Qualified ARN 启动的每个状态机执行，Step Functions 都会发出与当前发出的指标具有相同名称和值的其他指标。这些其他指标包含启动执行时使用的每个版本标识符和别名的维度。利用这些指标，您可以监控版本级别的状态机执行，并确定何时可能需要回滚方案。您也可以根据这些指标创建 Amazon CloudWatch 警报。

Step Functions 会针对您使用别名或版本启动的执行发出以下指标：

- `ExecutionTime`
- `ExecutionsAborted`
- `ExecutionsFailed`
- `ExecutionsStarted`
- `ExecutionsSucceeded`
- `ExecutionsTimedOut`

如果您使用版本 ARN 启动执行，Step Functions 会发布包含 `StateMachineArn` 的指标以及包含 `StateMachineArn` 和 `Version` 维度的第二个指标。

如果您使用别名 ARN 启动执行，Step Functions 会发出以下指标：

- 非限定 ARN 和版本的两个指标。
- 具有 `StateMachineArn` 和 `Alias` 维度的指标。

别名和版本部署示例

下面的金丝雀部署技术示例展示了如何使用 AWS Command Line Interface部署新的状态机版本。在此示例中，您创建的别名将 20% 的执行流量路由到新版本。然后将剩余的 80% 路由到早期版本。要部署新的状态机版本并使用别名转移执行流量，请完成以下步骤：

1. 从当前状态机修订版中发布一个版本。

   使用 AWS CLI 中的 publish-state-machine-version 命令，从名为 *myStateMachine*: 的状态机的当前版本中发布一个版本：

   ```
   aws stepfunctions publish-state-machine-version --state-machine-arn
    arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
   ```

响应会返回发布版本的 stateMachineVersionArn。例如，arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:1。

2. 创建指向状态机版本的别名。

   使用 create-state-machine-alias 命令创建指向 *myStateMachine* 版本 1 的别名，名为 *PROD*：

```
aws stepfunctions create-state-machine-alias --name PROD --routing-
configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
```

3. 验证由别名启动的执行是否使用正确的已发布版本。

   通过在 start-execution 命令中提供别名 **PROD** 的 ARN 来启动 *myStateMachine* 的新执行：

```
aws stepfunctions start-execution
  --state-machine-arn arn:aws:states:us-
east-1:123456789012:stateMachineAlias:myStateMachine:PROD
  --input "{}"
```

   如果您在 StartExecution 请求，则在启动执行时会使用状态机的最新revision，而不是别名中指定的版本。

4. 更新状态机定义并发布新版本。

   更新 *myStateMachine* 并发布其新版本。为此，请使用 update-state-machine 命令的可选 publish 参数：

```
aws stepfunctions update-state-machine
    --state-machine-arn arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine
    --definition $UPDATED_STATE_MACHINE_DEFINITION
    --publish
```

   响应会返回新版本的 stateMachineVersionArn。例如，arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:2。

5. 更新别名以指向两个版本并设置别名的路由配置。

   使用 update-state-machine-alias 命令更新别名 PROD 的路由配置。配置别名，使 80% 的执行流量流向版本 1，其余 20% 流向版本 2：

```
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. 将版本 1 替换为版本 2。

   在验证新的状态机版本能够正常运行后，您可以部署新的状态机版本。为此，请再次更新别名，将 100% 的执行流量分配给新版本。

   使用 update-state-machine-alias 命令设置别名 PROD 的路由配置，为版本 2 分配 100% 的流量：

```
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**
>
> 要回滚版本 2 的部署，请编辑别名的路由配置，将 100% 的流量转移到新部署的版本。
>
> ```
> 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}]"
> ```

您可以使用版本与别名功能来执行其他类型的部署。例如，您可以对状态机的新版本进行滚动部署。为此，请在指向新版本的别名的路由配置中，逐渐增加加权百分比。

您也可以使用版本与别名功能来执行蓝绿部署。为此，请创建一个名为 green 的别名，且该别名运行状态机的当前版本 1。然后，创建另一个名为 blue 的别名，运行新版本，例如 *2*。要测试新版本，请向 blue 别名发送执行流量。当您确信自己的新版本可以正常运行时，更新 green 别名以指向您的新版本。

逐步部署状态机版本

滚动部署是一种部署策略，用新版本的应用程序慢慢地替换旧版本的应用程序。要对状态机版本执行滚动部署，可逐渐增加向新版本发送的执行流量。流量大小和增加速度属于一种参数，由用户进行配置。

您可以使用以下选项之一对版本执行滚动部署：

- Step Functions 控制台 – 创建指向同一状态机的两个版本的别名。对于此别名，您可以对路由配置进行配置，以便在两个版本之间转移流量。有关使用控制台推出版本的更多信息，请参阅版本和别名。
- AWS CLI 和开发工具包的脚本 – 使用 AWS CLI 或 AWS 开发工具包创建 shell 脚本。有关更多信息，请参阅以下有关使用 AWS CLI CLI 和 AWS 开发工具包的部分。
- AWS CloudFormation 模板 – 使用 `AWS::StepFunctions::StateMachineVersion` 和 `AWS::StepFunctions::StateMachineAlias` 资源发布多个状态机版本，并创建指向其中一个或两个版本的别名。

使用 AWS CLI 部署新的状态机版本

本节中的示例脚本说明了如何使用 AWS CLI 将流量从以前的状态机版本逐渐转移到新的状态机版本。您可以使用此示例脚本，也可以根据需要对其进行更新。

此脚本显示了使用别名部署新状态机版本的金丝雀部署。以下步骤概述了脚本执行的任务：

1. 如果 `publish_revision` 参数设置为 true，则将最新的revision作为状态机的下一个版本发布。如果部署成功，则此版本将成为新的实时版本。

   如果将 `publish_revision` 参数设置为 false，则脚本将部署状态机的上一个已发布版本。
2. 如果别名还不存在，则创建一个别名。如果别名不存在，请将该别名的 100% 流量指向新版本，然后退出脚本。
3. 更新别名的路由配置，将一小部分流量从以前的版本转移到新版本中。您可以使用 `canary_percentage` 参数设置此金丝雀百分比。
4. 默认情况下，每 60 秒监控一次可配置的 CloudWatch 警报。如果其中任何一个警报被触发，请立即回滚部署，将 100% 的流量指向先前的版本。

   在 `alarm_polling_interval` 中以秒为单位定义的每个时间间隔后，继续监控警报。持续监视，直到超过 `canary_interval_seconds` 中定义的时间间隔。
5. 如果在 `canary_interval_seconds` 期间未触发任何警报，则将 100% 的流量转移到新版本。
6. 如果新版本成功部署，则删除任何早于 `history_max` 参数中指定数字的版本。

```
#!/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!
```

使用 AWS 开发工具包部署新的状态机版本

aws-stepfunctions-examples 中的示例脚本展示了如何使用适用于 Python 的 AWS 开发工具包将流量从状态机的旧版本逐步转移到新版本。您可以使用此示例脚本，也可以根据需要对其进行更新。

该脚本展示了以下部署策略：

- 金丝雀 – 流量将通过两次递增进行转移。

  在第一次递增中，将一小部分流量（例如，10%）转移到新版本。在第二次递增中，在指定的时间间隔（以秒为单位）结束之前，将剩余的流量转移到新版本。只有在指定的时间间隔内未触发 CloudWatch 警报时，才会将剩余流量切换到新版本。
- 线性或滚动 – 以相等的增量将流量转移到新版本中，每个增量之间的间隔秒数相等。

  例如，如果您将增量百分比指定为 **`20`**，`--interval` 为 **`600`** 秒，则此部署将每 600 秒增加 20% 的流量，直到新版本接收 100% 的流量。

  如果触发了任何 CloudWatch 警报，此部署会立即回滚新版本。
- 一次部署全部或蓝/绿 – 立即将 100% 的流量转移到新版本。此部署会监控新版本，并在触发任何 CloudWatch 警报时自动将其回滚到先前版本。

使用 AWS CloudFormation 部署新的状态机版本

以下 CloudFormation 模板示例发布了名为 *`MyStateMachine`* 的状态机的两个版本。它会创建一个名为 *`PROD`* 的别名，该别名指向这两个版本，然后部署了版本 2。

在此示例中，每五分钟将 10% 的流量转移到版本 2，直到该版本接收 100% 的流量。此示例还介绍了如何设置 CloudWatch 警报。如果您设置的任何警报进入 ALARM 状态，则部署将失败并立即回滚。

```
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 中的执行

当 AWS Step Functions 状态机运行并执行其任务时，就会发生状态机执行。每个 Step Functions 状态机可以有多个同时运行的执行，您可以从 Step Functions 控制台或者使用 AWS Command Line Interface 开发工具包、Step Functions API 操作或 AWS (AWS CLI) 启动这些执行。执行会接收 JSON 输入并生成 JSON 输出。您可以通过以下方式启动 Step Functions 执行。

- 调用 StartExecution API 操作。

- 在 Step Functions 控制台中启动新的执行。
- 使用 Amazon EventBridge 启动执行以响应事件。
- 使用 Amazon EventBridge Scheduler按计划启动状态机执行
- 使用 Amazon API Gateway 启动执行。
- 从 Task 状态启动嵌套工作流执行。

有关 Step Functions 的不同使用方式的更多信息，请参阅开发选项。

从 Task 状态启动工作流执行

AWS Step Functions 可以直接从状态机的 Task 状态启动工作流程执行。这允许您将工作流程分解为较小的状态机，并开始执行这些其他状态机。通过启动这些新的工作流程执行，您可以：

- 将更高级别的工作流程与较低级别的特定于任务的工作流程分开。
- 通过多次调用单独的状态机来避免重复元素。
- 创建模块化可重用工作流程库以加快开发速度。
- 降低复杂性并使编辑状态机和排除其故障变得更容易。

Step Functions 可以通过调用自己的 API 以作为集成服务来启动这些工作流执行。只需从 Task 状态调用 StartExecution API 操作并传递必要的参数即可。您可以使用任何服务集成模式调用 Step Functions API。

> **Tip**
>
> 要将嵌套工作流的示例部署到您的 AWS 账户，请参阅模块 13 - 嵌套快速工作流。

要启动新的状态机执行，请使用类似于以下示例的 Task 状态：

```
{
   "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
}
```

此 Task 状态会启动新的 HelloWorld 状态机执行，并传递 JSON 注释作为输入。

> **Note**
>
> StartExecution API 操作配额可以限制您可以启动的执行次数。对 StepFunctions.ExecutionLimitExceeded 使用 Retry 以确保启动您的执行。请参阅以下内容。
>
> - 与 API 操作限制相关的配额
> - Step Functions 中的错误处理

关联工作流程执行

要将已启动的工作流程执行与启动它的执行相关联，请将执行 ID 从上下文对象传递到执行输入。您可以在正在运行的执行中从您的 Task 状态访问上下文对象中的 ID。通过将 .$ 附加到参数名称并在上下文对象中通过 $$.Execution.Id 引用 ID 来传递执行 ID。

```
"AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
```

在启动执行时，您可以使用名为 AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID 的特殊参数。如果包含此关联，则会在 Step Functions 控制台的步骤详细信息部分提供链接。当提供后，您可以轻松跟踪工作流程的执行情况（从正在启动的执行到已启动的工作流程执行）。使用前面的示例，将执行 ID 与 HelloWorld 状态机的已启动执行相关联，如下所示。

```
{
```

```
   "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
}
```

有关更多信息，请参阅下列内容：

- 使用其他服务
- 将参数传递给服务 API
- 访问上下文对象
- AWS Step Functions

将 Amazon EventBridge 调度器与 AWS Step Functions 结合使用

Amazon EventBridge 调度器是一个无服务器调度器，使您能够从一个中央托管服务创建、运行和管理任务。借助 EventBridge 调度器，您可以使用 cron 和 rate 表达式为定期模式创建计划，也可以配置一次性调用。您可以设置灵活的交付时间窗口、定义重试限制，并为失败的 API 调用设置最大保留时间。

例如，使用 EventBridge 调度器，您可以在发生安全相关事件或自动执行数据处理作业时，按计划启动状态机执行。

本页介绍了如何使用 EventBridge 调度器按计划启动 Step Functions 状态机的执行。

主题

- 设置执行角色
- 创建计划
- 相关的 资源

设置执行角色

创建新计划时，EventBridge 调度器必须有权代表您调用其目标 API 操作。您可以使用执行角色授予 EventBridge 调度器这些权限。您附加到计划执行角色的权限策略定义了所需权限。这些权限取决于您希望 EventBridge 调度器调用的目标 API。

使用 EventBridge 调度器控制台创建计划时，EventBridge 调度器会根据选择的目标自动设置执行角色，如以下步骤所示。如果您想使用 EventBridge 调度器 SDK（AWS CLI 或 AWS CloudFormation ）之一创建计划，您必须拥有现有的执行角色，以授予 EventBridge 调度器调用目标所需的权限。有关为计划手动设置执行角色的更多信息，请参阅 EventBridge Scheduler User Guide 中的 Setting up an execution role。

创建计划

使用控制台创建计划

1. 打开 Amazon EventBridge 调度器控制台，网址为：https://console.aws.amazon.com/scheduler/home。
2. 在计划页面，选择创建计划。
3. 在指定计划详细信息页面，在计划名称和描述部分中，执行以下操作：

   a. 对于计划名称，输入计划的名称。例如，**`MyTestSchedule`**。

   b. （可选）对于描述，输入对计划的描述。例如，**`My first schedule`**。

   c. 对于计划组，从下拉列表中选择一个计划组。如果您没有计划组，选择默认。要创建计划组，选择创建自己的计划。

   您可以使用计划组将标签添加到计划组。

4. • 选择计划选项。

出现	请执行此操作...
一次性计划 一次性计划仅在您指定的日期和时间调用一次目标。	对于日期和时间，请执行以下操作： • 输入 YYYY/MM/DD 格式的有效日期。 • 输入 24 小时 hh:mm 格式的时间戳。

出现	请执行此操作...
	• 对于时区，选择时区。
定期计划 定期计划按照您使用 cron 表达式或 rate 表达式指定的速率调用目标。	a. 对于计划类型，执行以下操作之一： • 要使用 cron 表达式定义计划，请选择基于 cron 的计划并输入 cron 表达式。 • 要使用 rate 表达式定义计划，请选择基于 rate 的计划并输入 rate 表达式。 有关 cron 和 rate 表达式的更多信息，请参阅 Amazon EventBridge Scheduler User Guide 中的 Schedule types on EventBridge Scheduler。 b. 对于灵活的时间窗口，选择关闭以关闭该选项，或者选择一个预定义的时间窗口。例如，如果您选择 15 分钟并且将定期计划设置为每小时调用一次其目标，则该计划将在每小时开始后的 15 分钟内运行。

5. （可选）如果您在上一步中选择定期计划，在时间范围部分，请执行以下操作：

   a. 对于时区，请选择时区。

   b. 对于开始日期和时间，请输入 YYYY/MM/DD 格式的有效日期，然后指定 24 小时 hh:mm 格式的时间戳。

c. 对于结束日期和时间，请输入 YYYY/MM/DD 格式的有效日期，然后指定 24 小时 hh:mm 格式的时间戳。

6. 选择下一步。

7. 在选择目标页面，选择 EventBridge 调度器调用的 AWS API 操作：

   a. 选择 AWS Step Functions StartExecution。

   b. 在 StartExecution 部分，选择一个状态机或选择创建新的状态机。

      目前，您无法按计划运行同步快速工作流。

   c. 输入 JSON 有效负载，用于执行。即使您的状态机不需要任何 JSON 有效负载，您仍必须包含 JSON 格式的输入，如以下示例所示。

```
{
    "Comment": "sampleJSONData"
}
```

8. 选择下一步。

9. 在 Settings (设置) 页面上，执行以下操作：

   a. 要打开计划，在计划状态下，切换启用计划。

   b. 要为计划配置重试策略，在重试策略和死信队列（DLQ）下，请执行以下操作：

      - 切换重试。
      - 对于事件的最长期限，输入 EventBridge 调度器必须保留未处理事件的最长小时和分钟数。
      - 最长时间为 24 小时。
      - 对于最大重试次数，输入在目标返回错误的情况下，EventBridge 调度器重试计划的最大次数。

        最大值为 185 次重试。

      配置重试策略后，如果计划未能调用其目标，EventBridge 调度器将重新运行该计划。如果已配置，则必须为计划设置最长保留时间和最大重试次数。

   c. 选择 EventBridge 调度器存储未送达事件的位置。

死信队列（DLQ）选项	请执行此操作...
请勿存储	选择 None。
将事件存储在创建计划所在的同一 AWS 账户 中	a. 选择在我的 AWS 账户 中选择一个 Amazon SQS 队列作为 DLQ。 b. 选择 Amazon SQS 队列的 Amazon 资源名称（ARN）。
将事件存储在与创建计划所在不同的 AWS 账户 中	a. 选择在另一个 AWS 账户 中指定一个 Amazon SQS 队列作为 DLQ。 b. 输入 Amazon SQS 队列的 Amazon 资源名称（ARN）。

d. 要使用客户托管密钥加密目标输入，在加密下，选择自定义加密设置（高级）。

   如果选择此选项，请输入现有的 KMS 密钥 ARN 或选择创建一个 AWS KMS key 以导航到 AWS KMS 控制台。有关 EventBridge 调度器如何加密静态数据的更多信息，请参阅 Amazon EventBridge Scheduler User Guide 中的 Encryption at rest。

e. 要让 EventBridge 调度器为您创建新的执行角色，请选择为此计划创建新角色。然后，在角色名称中输入名称。如果您选择此选项，EventBridge 调度器会将模板化目标所需的必要权限附加到该角色。

10. 选择下一步。
11. 在查看并创建计划页面上，查看计划的详细信息。在每个部分中，选择编辑返回到该步骤并编辑其详细信息。
12. 选择创建计划。

    您可以在计划页面上查看新的和现有的计划列表。在状态列下，验证新计划是否已启用。

要确认 EventBridge 调度器是否调用了该状态机，查看该状态机的 Amazon CloudWatch Logs。

相关的 资源

有关 EventBridge 调度器的详细信息，请参阅以下内容：

- EventBridge Scheduler User Guide
- EventBridge Scheduler API Reference
- EventBridge Scheduler Pricing

控制台中的标准和快速工作流执行

创建状态机时，您必须选择标准或快速类型。状态机的默认类型为标准。类型为标准的状态机称为标准工作流，类型为快速的状态机被称为快速工作流。

对于标准工作流和快速工作流，您都可以使用 Amazon States Language 来定义状态机。根据您选择的类型，状态机的执行行为会有所不同。

> **Important**
>
> 创建状态机后，无法更改您选择的类型。

有关标准工作流和快速工作流之间差异的更多信息，请参阅标准和快速工作流。

标准工作流执行的历史记录记录在 Step Functions 中，而快速工作流执行的历史记录不记录在 Step Functions 中。要记录 Express 工作流程执行的历史记录，您必须将其配置为向 Amazon 发送日志 CloudWatch。有关更多信息，请参阅使用 CloudWatch Logs 进行日志记录。

在快速工作流上配置日志记录后，您可以在 Step Functions 控制台中查看其执行情况。查看快速工作流执行和标准工作流执行的控制台体验类似，以下区别和限制除外。

> **Note**
>
> 由于 Express 工作流程的执行数据是使用 L CloudWatch ogs Insights 显示的，因此扫描日志将产生费用。默认情况下，您的日志组仅列出最近三个小时内完成的执行。如果您指定较大的时间范围，包括更多的执行事件，则成本将会增加。有关更多信息，请参阅 “CloudWatch 定价” 页面上 “日志” 选项卡下的 “销售日志” 和。使用 CloudWatch Logs 进行日志记录

内容

- 控制台体验差异
- 查看快速工作流执行的注意事项和限制

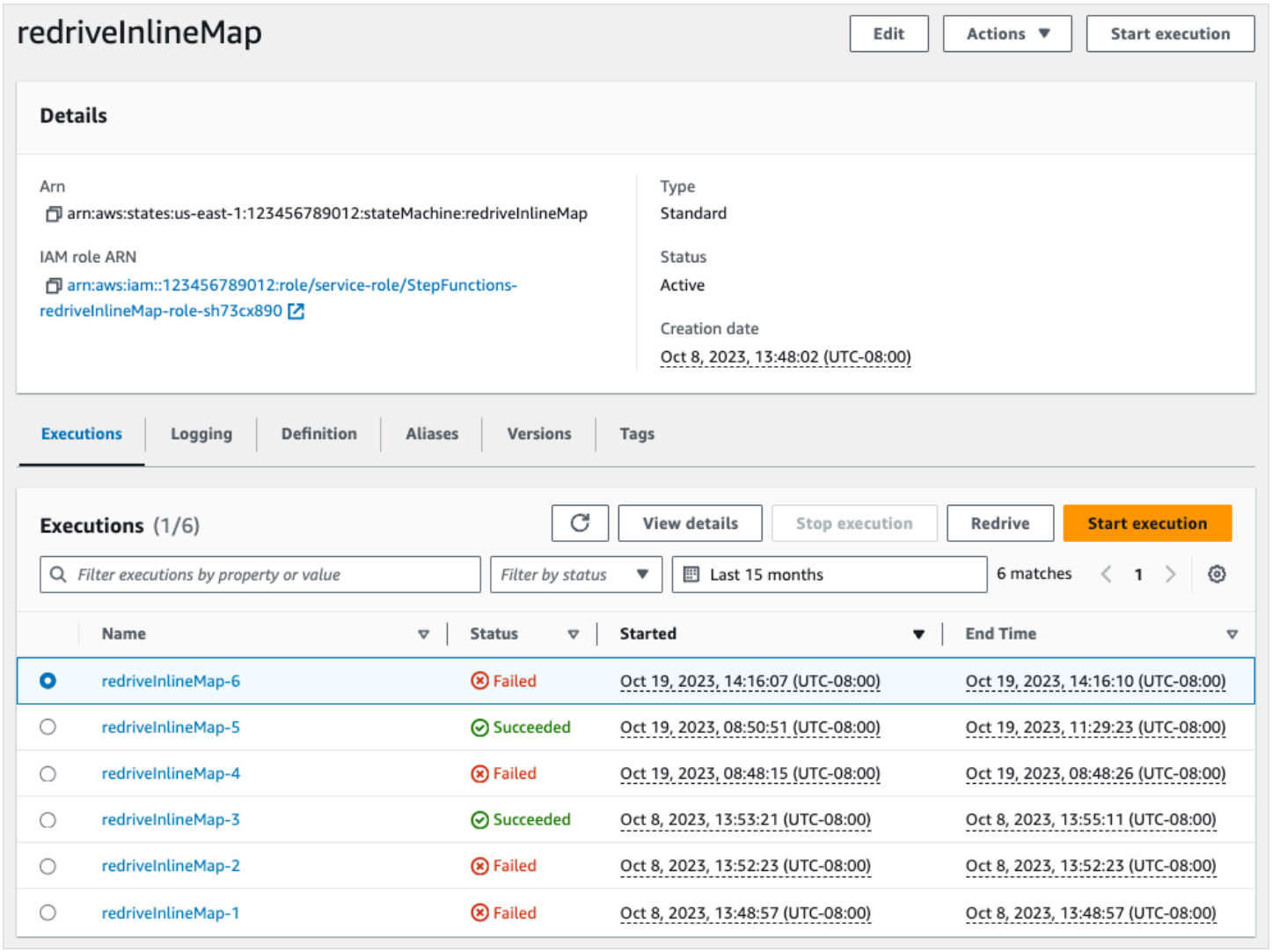
控制台体验差异

对于所有标准和快速工作流，您可以在 Step Functions 控制台的状态机详细信息 页面上查看详细信息，例如状态机及其 IAM 角色 ARN。

在状态机详细信息 页面上，您还可以在执行选项卡下看到状态机的执行历史记录列表。使用“按属性或值筛选执行”框来搜索所选状态机的特定执行、版本或别名。使用“全部”下拉列表按状态筛选执行历史记录。您也可以选择执行历史记录，然后选择查看详细信息按钮，打开其执行详细信息 页面。

标准工作流

标准工作流的执行历史记录始终可查看过去 90 天内完成的执行。



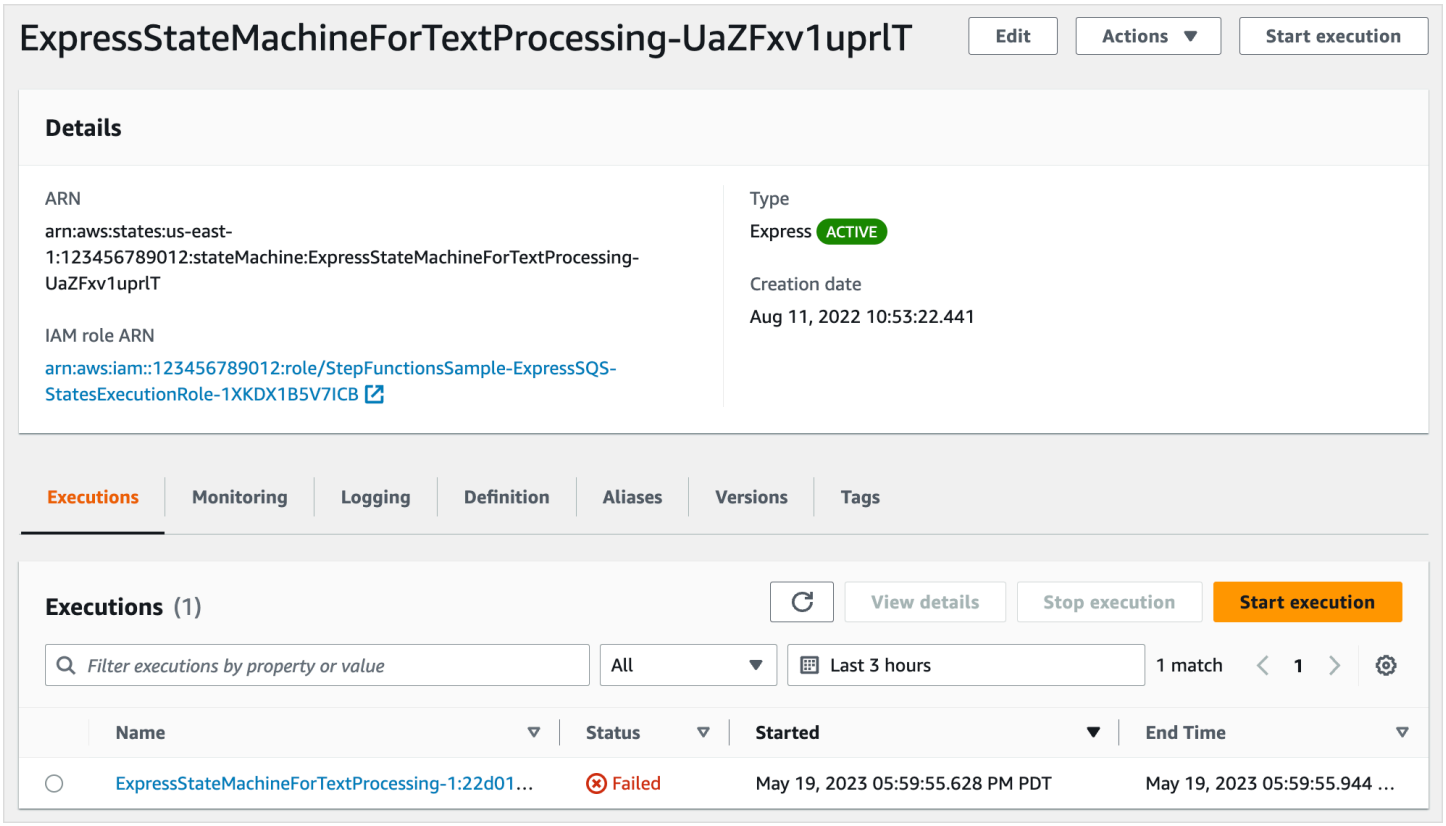
快速工作流

为了显示 Express 工作流程的执行历史记录，Step Functions 控制台会检索通过日志日志组收集的 CloudWatch 日志数据。

您还必须启用新的控制台体验才能查看快速工作流执行情况。为此，请选择执行选项卡上横幅内显示的启用按钮。选择后，该按钮将不会再出现。

> **Tip**
>
> 要在启用或禁用控制台体验之间切换，请使用启用快速执行历史记录切换按钮。

默认情况下，可以查看过去三个小时内完成的执行历史记录。您可以调整此时间范围或指定自定义范围。如果您指定了较大的时间范围，包括更多的执行事件，则扫描日志的成本将增加。有关更多信息，请参阅 “CloudWatch 定价” 页面上 “日志” 选项卡下的 “销售日志” 和。使用 CloudWatch Logs 进行日志记录



查看快速工作流执行的注意事项和限制

在 Step Functions 控制台上查看快速工作流执行时，请谨记以下注意事项和限制。

- Express 工作流程执行详情的可用性依赖于 Amazon CloudWatch 日志
- 如果日志级别为“错误”或“严重”，则可获得部分快速工作流执行详细信息
- 更新后无法查看旧执行的状态机定义

Express 工作流程执行详情的可用性依赖于 Amazon CloudWatch 日志

> **Note**
>
> 如果您不启用新的控制台体验来查看快速工作流执行情况，则 Step Functions 控制台中将无法查看执行历史记录及其相应的执行详细信息。要启用新的控制台体验，请选择执行选项卡横幅内显示的启用按钮。

对于 Express 工作流程，其执行历史和详细的执行信息是通过 CloudWatch Logs Insights 收集的。此信息保存在您在创建状态机时指定的 CloudWatch 日志组中。状态机的执行历史记录显示在 Step Functions 控制台的执行选项卡下。有关状态机每次执行的详细信息显示在所选执行的执行详细信息页面上。

> **Warning**
>
> 如果您删除 Express 工作流程的 CloudWatch 日志，它将不会列在 “执行” 选项卡下。

建议您使用默认日志级别（所有）来记录所有执行事件类型。编辑现有状态机时，可以根据需要更新它们的日志级别。有关更多信息，请参阅 使用 CloudWatch Logs 进行日志记录 和 日志级别。

如果日志级别为“错误”或“严重”，则可获得部分快速工作流执行详细信息

默认情况下，快速工作流执行的日志级别设置为所有。更改日志级别，不会影响已完成执行的执行历史记录和执行详细信息。但是，所有新的执行都将根据更新的日志级别发出日志。有关更多信息，请参阅 使用 CloudWatch Logs 进行日志记录 和 日志级别。

例如，如果您将日志级别从所有更改为错误或严重，则 Step Functions 控制台上的执行选项卡将仅列出失败的执行。在事件视图选项卡中，控制台仅显示失败的状态机步骤的事件详细信息。

建议您使用默认日志级别（所有）来记录所有执行事件类型。编辑状态机时，您可以根据需要更新现有状态机的日志级别。

更新后无法查看旧执行的状态机定义

快速工作流不会存储过去执行的状态机定义。如果更改状态机定义，则只能查看使用最新定义执行的状态机定义。

例如，如果您从状态机定义中移除一个或多个步骤，Step Functions 会检测到定义与先前的执行事件不匹配。由于快速工作流不存储以前的定义，因此 Step Functions 无法显示在状态机定义早期版本上运行的状态机定义。因此，执行输入和输出、定义、图表视图和表格视图选项卡无法用于在状态机定义的早期版本上运行的执行。

在 Step Functions 控制台上查看和调试执行

Step Functions 控制台上的执行详细信息 页面显示有关标准和快速工作流中过去和正在进行的状态机执行的信息。此信息以控制面板格式显示。例如，您可以找到状态机的 Amazon States Language 定义、其执行状态、ARN 和状态转换总数。您还可以查看状态机中任何单个状态的执行详细信息。

内容

- “执行详细信息”页面 – 界面概述
  - 执行摘要
  - 错误消息
  - 视图模式
  - 步骤详细信息
  - 事件
- 教程：使用 Step Functions 控制台检查状态机执行
  - 第 1 步：创建并测试所需的 Lambda 函数
  - 第 2 步：创建并执行状态机
  - 第 3 步：查看状态机执行详细信息
  - 第 4 步：探索不同的视图模式

“执行详细信息”页面 – 界面概述

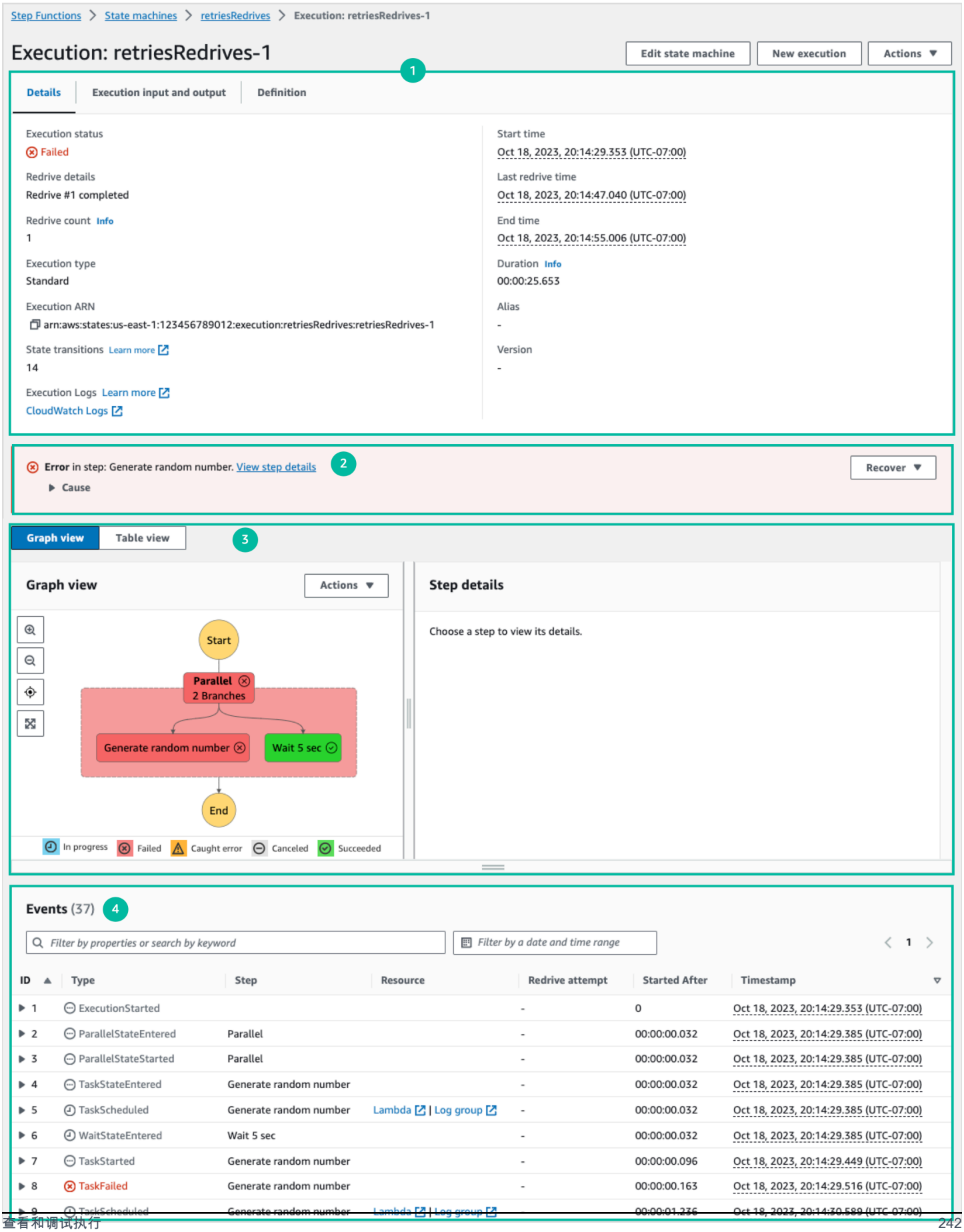
您可以在执行详细信息 页面上找到标准工作流和快速工作流中所有正在进行和过去的状态机执行的详细信息。如果您在启动执行时指定了执行 ID，则此页面的标题将使用该执行 ID。否则，该页面将以 Step Functions 自动生成的唯一执行 ID 为标题。

除了执行指标外，执行详细信息 页面还提供以下选项，用于管理状态机及其执行：

按钮	选择此按钮执行的操作：
编辑状态机	编辑状态机的 Amazon States Language 定义。
新执行	启动一个新的状态机执行。
操作	提供以下选项供选择： • 停止执行 – 停止正在进行的执行。此选项对已完成的执行不可用。 • Redrive – Redrive过去 14 天内未成功完成的标准工作流执行。其中包括执行失败、中止或超时的执行。有关更多信息，请参阅Redriving执行。 • 导出 – 以 JSON 格式导出执行详细信息，以便与他人共享或执行离线分析。 • 发送反馈 – 分享有关界面的反馈。

> **查看以版本或别名启动的执行**
>
> 您还可以在 Step Functions 控制台中查看以版本或别名启动的执行。有关更多信息，请参阅列出版本和别名功能的执行。

执行详细信息 控制台页面包含下列部分：


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. 执行摘要
2. 错误消息
3. 视图模式
4. 步骤详细信息
5. 事件

执行摘要

执行摘要 部分显示在执行详细信息 页面的顶部。本部分为您提供工作流的执行详细信息概览。此信息分为以下三个选项卡：

详细信息

显示信息，例如执行状态、ARN 以及执行启动和结束时间的时间戳。您还可以查看运行状态机执行时发生的状态转换的总数。如果您为状态机启用了跟踪或日志，则还可以查看 X-Ray 跟踪地图和 Amazon CloudWatch 执行日志的链接。

如果状态机执行由另一个状态机启动，则可以在此选项卡上查看父状态机的链接。

如果状态机执行了redriven，则此选项卡会显示redrive相关信息，例如Redrive计数。

执行输入和输出

显示状态机执行输入和输出 side-by-side。

定义

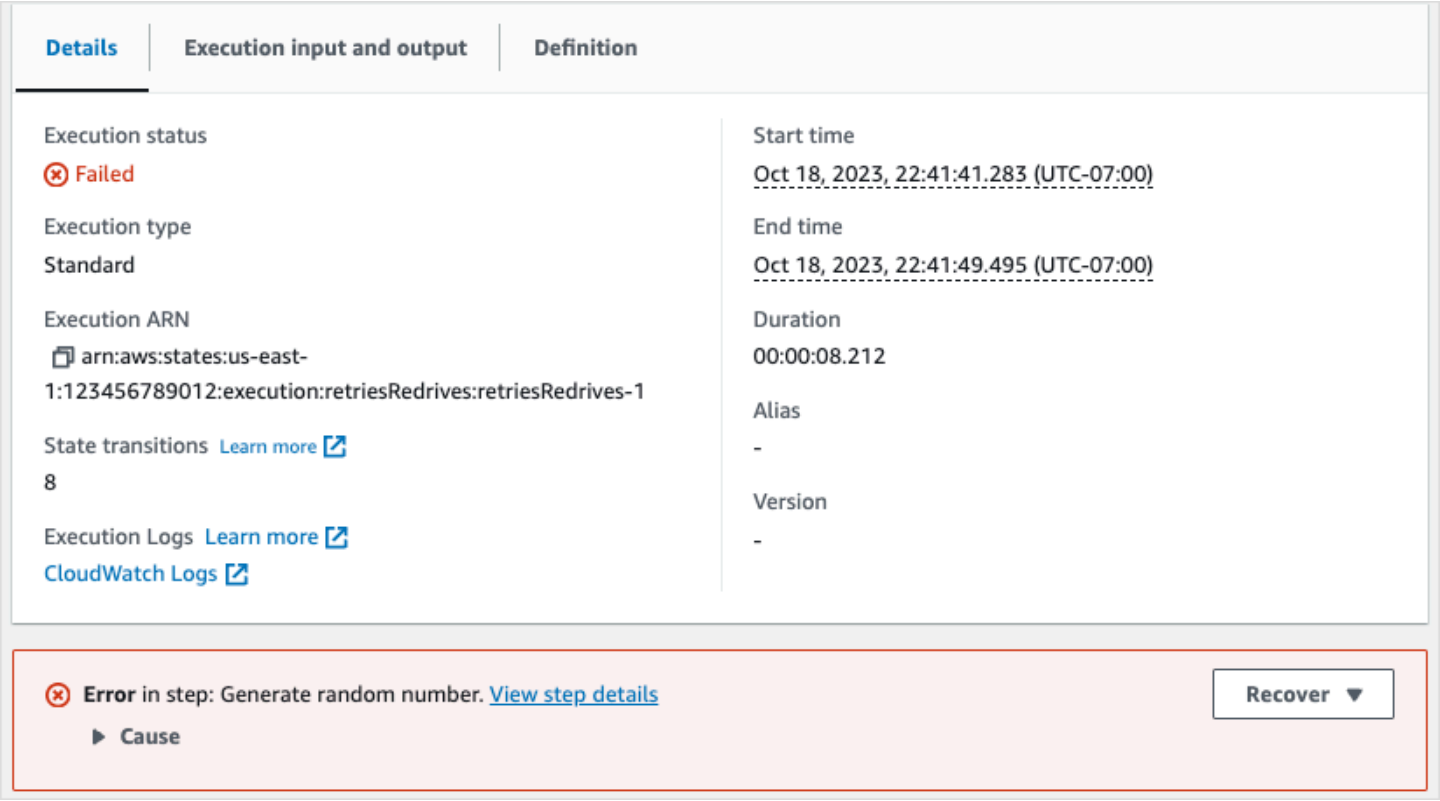
显示状态机的 Amazon States Language 定义。

错误消息

如果状态机执行失败，则执行详细信息 页面会显示一条错误消息。在错误消息中选择原因或查看步骤详细信息，查看执行失败的原因或导致错误的步骤。

如果您选择查看步骤详细信息，Step Functions 将在步骤详细信息、图表视图和表格视图选项卡中突出显示导致错误的步骤。如果步骤是已定义重试的 Task、Map 或 Parallel 状态，则步骤详细信息窗格将显示该步骤的重试选项卡。此外，如果已redriven执行，则可以在步骤详细信息窗格的重试次数和redrives选项卡中查看重试次数和redrive的详细信息。

通过此错误消息上的恢复下拉按钮，您可以redrive失败执行或启动新的执行。有关更多信息，请参阅Redriving执行。



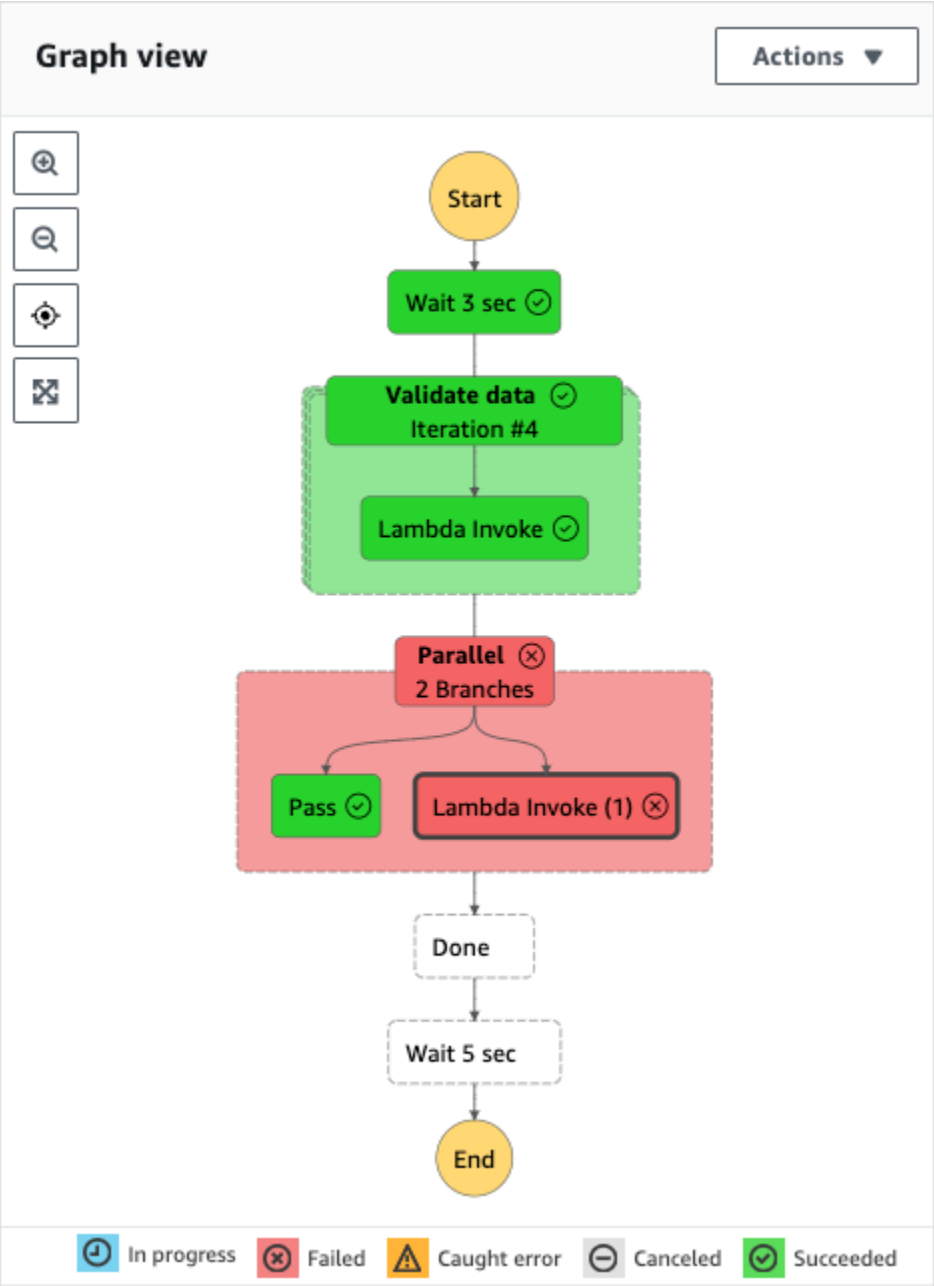
视图模式

视图模式 部分包含状态机两种不同的可视化效果。您可以选择查看工作流的图形化表示、概述工作流状态的表格或与状态机执行相关的事件列表：

> **Note**
>
> 选择一个选项卡查看其内容。

Graph view

图表视图模式显示工作流的图形化表示。底部包含一个图例，用于指示状态机的执行状态。它还包含用于放大、缩小、居中对齐整个工作流或在全屏模式下查看工作流的按钮。



在此视图中，您可以选择工作流中的任意步骤，在步骤详细信息 组件中查看有关其执行的详细信息。在图表视图中选择一个步骤后，表格视图也会显示该步骤。反之亦然。如果从表格视图中选择一个步骤，则图表视图将显示相同的步骤。

如果状态机包含 Map 状态、Parallel 状态或两者兼有，则可以在图表视图中查看它们在工作流中的名称。此外，对于 Map 状态，图表视图允许您在 Map 状态执行数据的不同迭代之间移动。例如，如果您的 Map 状态有五次迭代，并且您想要查看第三个和第四次迭代的执行数据，请执行以下操作：

1. 选择要查看其迭代数据的 Map 状态。
2. 在 Map 迭代查看器中，从下拉列表中选择 #2 来查看第三次迭代。这是因为迭代是从零开始计算的。同样，从下拉列表中选择 #3 表示 Map 状态的第四次迭代。

   或者，还可以使用

   △

   和

   ▽

   控件在 Map 状态的不同迭代之间移动。

   > **Note**
   >
   > 如果您的状态机包含嵌套 Map 状态，则将显示父和子 Map 状态迭代的下拉列表，如以下示例所示：
   >
   > 
   >

3. （可选）如果一次或多次 Map 状态迭代执行失败或停止执行，则可以通过在下拉列表的失败或中止下选择这些迭代编号来查看其数据。

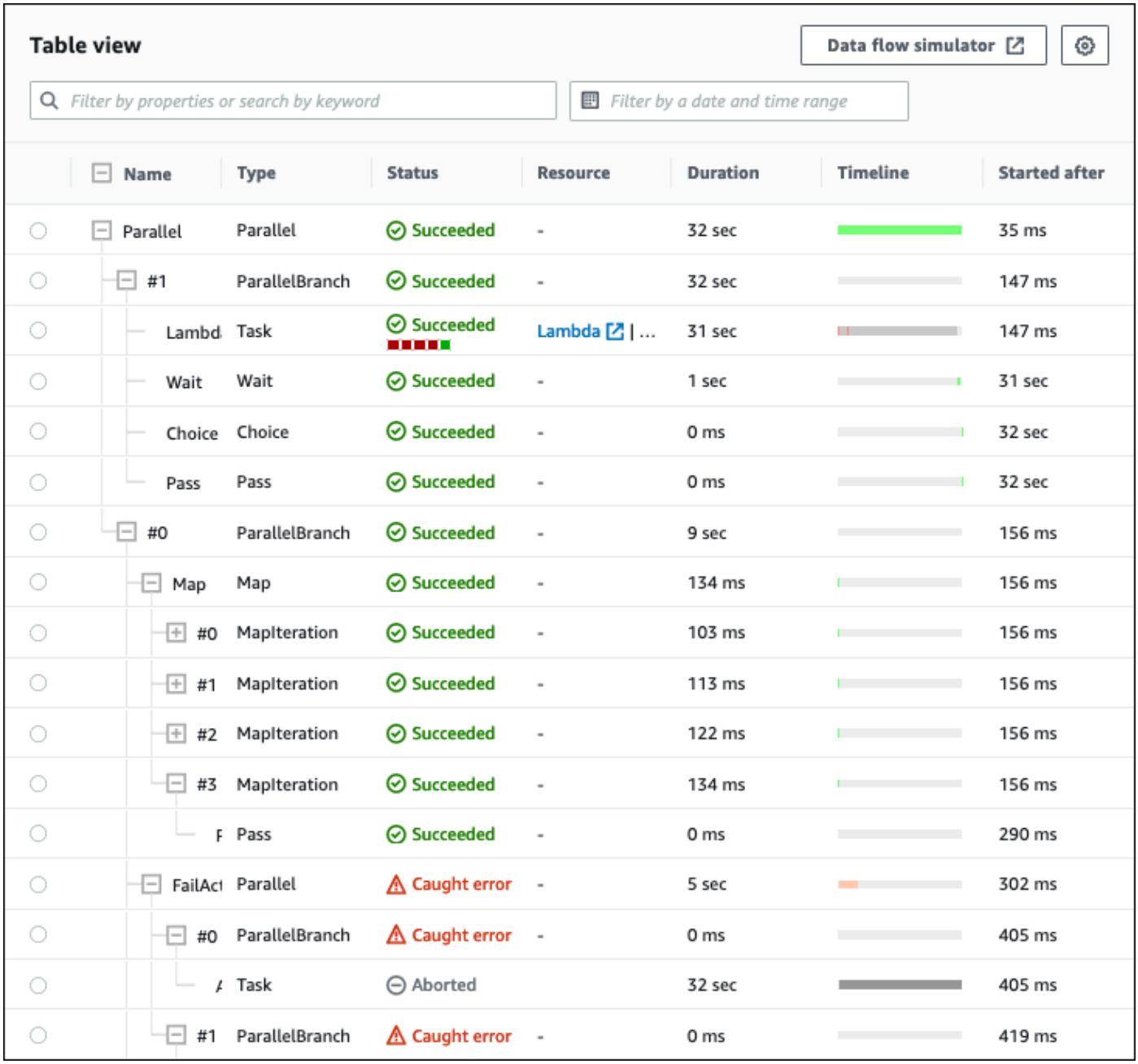
最后，您可以使用导出和布局按钮将工作流图表导出为 SVG 或 PNG 图像。您也可以在工作流的水平和垂直视图之间切换。

Table view

表格视图模式以表格形式显示工作流中的状态。在此视图模式下，您可以查看工作流程中执行的每个状态的详细信息，包括其名称、其使用的任何资源（例如 AWS Lambda 函数）的名称以及该状态是否成功执行。

在此视图中，您可以选择工作流中的任何状态，在步骤详细信息 组件中查看有关其执行的详细信息。当您在表格视图中选择一个步骤时，图表视图也会显示该步骤。反之亦然。如果您从图表视图中选择一个步骤，则表格视图将显示相同的步骤。

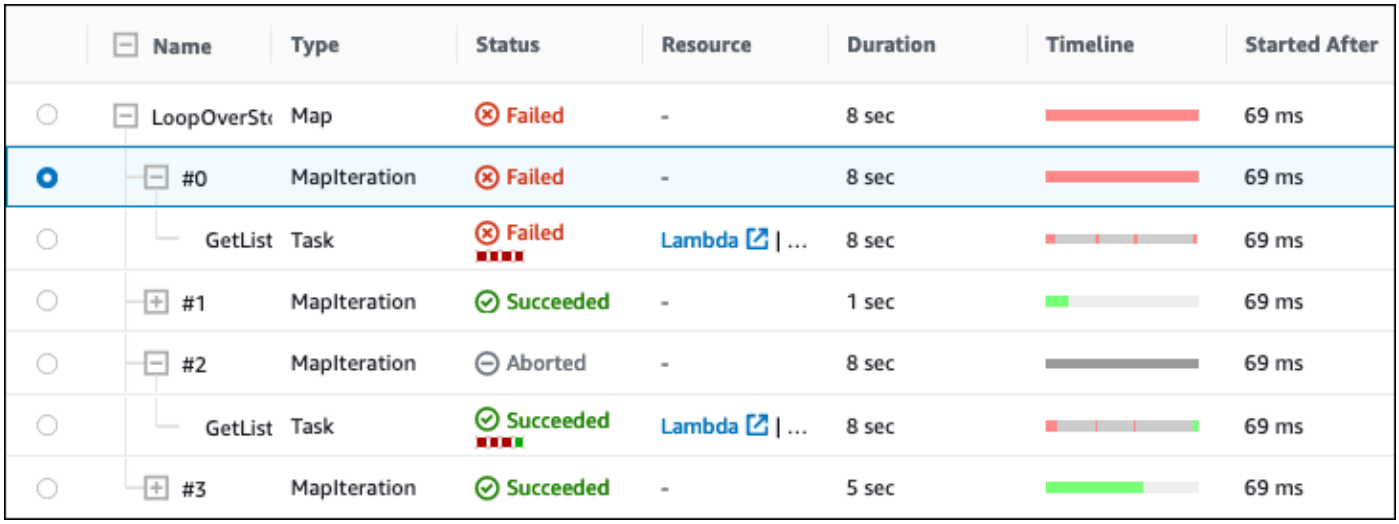
您还可以通过对视图应用筛选条件来限制在表格视图模式下显示的数据数量。您可以为特定属性创建筛选条件，例如状态或Redrive尝试。有关更多信息，请参阅教程：使用 Step Functions 控制台检查状态机执行。



默认情况下，此模式显示名称、类型、状态、资源和之后开始列。您可以使用首选项对话框配置要查看的列。您在此对话框中所做的选择会持续用于将来的状态机执行，直到再次更改。

如果添加时间线列，则显示每个状态的执行持续时间相对于整个执行的运行时间的状况。它将以彩色编码的线性时间线显示。这可以帮助您识别特定状态执行中任何与性能相关的问题。时间线上每个状态的颜色编码段可帮助您识别该状态的执行状态，例如进行中、失败或中止。

例如，如果您在状态机中为某个状态定义了执行重试次数，则这些重试次数将显示在时间线中。红色段代表失败的 `Retry` 尝试，而浅灰色段代表每次 `Retry` 尝试之间的 `BackoffRate`。



如果状态机包含 Map 状态、Parallel 状态或两者兼有，则可以在工作流中以表格视图查看它们的名称。对于 Map 和 Parallel 状态，表格视图模式将其迭代和并行分支的执行数据显示为树形视图中的节点。您可以选择这些状态中的每个节点，在步骤详细信息 部分查看其各自的详细信息。例如，您可以查看导致状态失败的特定 Map 状态迭代的数据。展开 Map 状态的节点，然后在状态列中查看每次迭代的状态。

步骤详细信息

在图表视图或表格视图中选择状态时，步骤详细信息 部分将在右侧打开。该部分包含以下选项卡，可为您提供所选状态的详细信息：

输入

显示所选状态的输入详细信息。如果输入中有错误，则会在选项卡标题上用
⚠
表示。此外，您还可以在此选项卡中查看错误的原因。

您也可以选择高级视图切换按钮，查看数据通过选定状态时的输入数据传输路径。您可以借此识别将一个或多个字段（例如 InputPath、Parameters、ResultSelector、OutputPath 和 ResultPath）应用于数据时，输入是如何被处理的。

输出

显示选定状态的输出。如果输出中有错误，则会在选项卡标题上用
⚠
表示。此外，您还可以在此选项卡中查看错误的原因。

您也可以选择高级视图切换按钮，查看数据通过选定状态时的输出数据传输路径。您可以借此识别将一个或多个字段（例如 InputPath、Parameters、ResultSelector、OutputPath 和 ResultPath）应用于数据时，输入是如何被处理的。

详细信息

显示状态类型、执行状态和执行持续时间等信息。

对于使用资源的Task州（例如） AWS Lambda，此选项卡提供了指向资源定义页面和资源调用的 Amazon CloudWatch 日志页面的链接。它还会显示 Task 状态的 TimeoutSeconds 和 HeartbeatSeconds 字段的值（如果已指定）。

对于 Map 状态，此选项卡显示有关 Map 状态迭代总数的信息。迭代分为失败、中止、成功或。InProgress

定义

显示与所选状态对应的 Amazon States Language 定义。

重试

> **Note**
>
> 仅当您在状态机 Task 或 Parallel 状态下定义了 Retry 字段时，才会显示此选项卡。

显示所选状态在最初执行尝试中的初始和后续重试次数。对于初次尝试和所有后续失败的尝试，选择类型旁边的

▶

，查看下拉框中显示的失败原因。如果重试成功，则可以查看出现在下拉框中的输出。

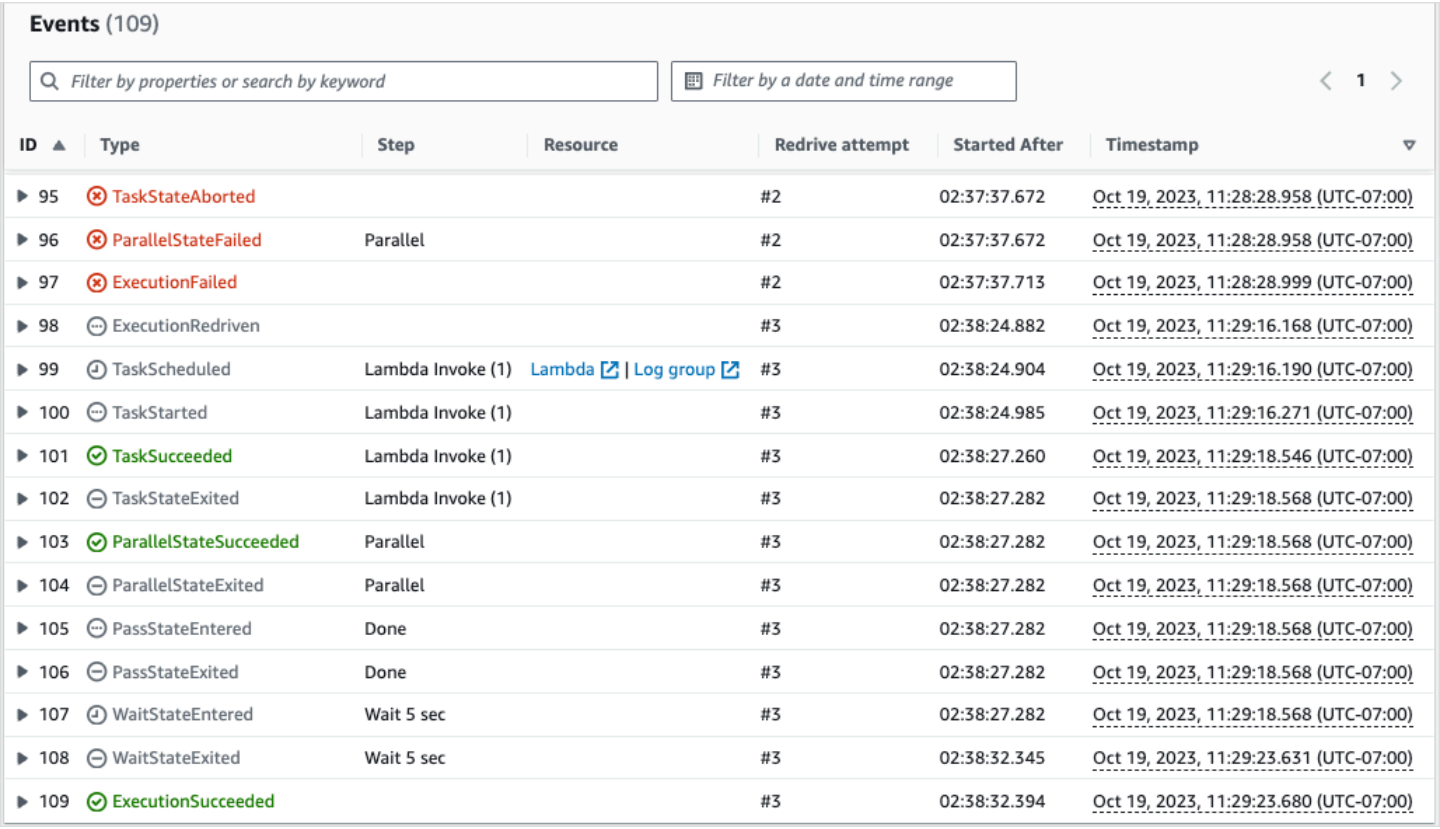
如果redriven了执行，则此选项卡标题将显示名称重试和redrives，并显示每次redrive的重试详细信息。

事件

显示与执行中选定状态关联的事件的筛选列表。此选项卡上的信息是事件 表格中完整执行事件历史记录的子集。

事件

事件表格以事件列表的形式，显示所选执行的完整历史记录，跨越多个页面。每页最多包含 25 个事件。此部分还显示事件总数，这可以帮助您确定是否超过了 2.5 万个事件的最大事件历史记录数。



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)

默认情况下，事件表格中的结果根据事件的时间戳按升序显示。您可以通过单击时间戳列标题将执行事件历史记录的排序更改为降序。

在事件表格中，每个事件都采用颜色编码表示其执行状态。例如，失败的事件以红色显示。要查看有关事件的更多详细信息，请选择事件 ID 旁边的 ▶。打开后，事件详细信息将显示事件的输入、输出和资源调用。

此外，在事件表格中，您可以应用筛选条件来限制显示的执行事件历史记录结果。您可以选择诸如 ID 或Redrive尝试之类的属性。有关更多信息，请参阅 教程：使用 Step Functions 控制台检查状态机执行。

教程：使用 Step Functions 控制台检查状态机执行

在本教程中，您将学习如何检查执行详细信息 页面上显示的执行信息并查看执行失败的原因。然后，您将学习如何访问 Map 状态执行的不同迭代。最后，您将学习如何在表格视图上配置列，以及如何应用合适的筛选条件来仅查看您感兴趣的信息。

在本教程中，您将创建一个标准类型的状态机，用于获取一组水果的价格。为此，状态机使用三个 AWS Lambda 函数，分别返回四个水果的随机列表、每种水果的价格以及水果的平均成本。如果水果的价格小于或等于阈值，Lambda 函数就会出错。

> **Note**
>
> 虽然以下过程包含有关如何检查标准工作流执行详细信息的说明，但您也可以检查快速工作流执行的详细信息。有关标准工作流和快速工作流类型执行详细信息的差异，请参阅控制台中的标准和快速工作流执行。

内容

- 第 1 步：创建并测试所需的 Lambda 函数
- 第 2 步：创建并执行状态机
- 第 3 步：查看状态机执行详细信息
- 第 4 步：探索不同的视图模式

第 1 步：创建并测试所需的 Lambda 函数

1. 打开 Lambda 控制台，然后执行第 1 步：创建 Lambda 函数部分中的第 1 步至第 4 步。确保将 Lambda 函数命名为 **`GetListOfFruits`**。
2. 创建 Lambda 函数后，复制页面右上角显示的函数 Amazon 资源名称 (ARN)。要复制 ARN，请单击

   。

   以下是一个 ARN 示例，其中 *`function-name`* 是 Lambda 函数的名称（在本例中为 `GetListOfFruits`）：

   ```
   arn:aws:lambda:us-east-1:123456789012:function:function-name
   ```
3. 将以下 Lambda 函数的代码复制到页面的GetListOfFruits代码源区域。

   ```
   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. 选择部署，然后选择测试，部署更改并查看 Lambda 函数的输出。
5. 按照以下步骤创建另外两个 Lambda 函数，分别命名为 **GetFruitPrice** 和 **CalculateAverage**：

   a. 将以下代码复制到 GetFruitPriceLambda 函数的代码源区域：

```
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. 将以下代码复制到 CalculateAverageLambda 函数的代码源区域：

```
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. 确保复制这两个 Lambda 函数的 ARN，然后对其进行部署和测试。

第 2 步：创建并执行状态机

使用 Step Functions 控制台创建状态机，用于调用您在第 1 步中创建的 Lambda 函数。在此状态机中，定义三个 Map 状态。每个 Map 状态都包含一个调用 Lambda 函数的 Task 状态。此外，每个 Task 状态都定义了一个 Retry 字段，并为每个状态定义了重试次数。如果一个 Task 状态遇到运行时错误，则会再次执行该状态，但不得超过为此 Task 定义的重试次数。

1. 打开 Step Functions 控制台，然后选择用代码编写工作流。

   > **Important**
   >
   > 确保您的状态机与您之前创建的 Lambda 函数位于相同的 AWS 账户和区域下。

2. 对于类型，保留默认的标准选择。
3. 复制以下 Amazon States Language 定义并将其粘贴到定义下。确保将显示的 ARN 替换为您之前创建的 Lambda 函数的 ARN。

```
{
    "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. 为状态机输入名称。保留本页其他选项的默认选项，然后选择创建状态机。

5. 打开标题为状态机名称的页面。执行第 4 步：运行状态机部分中的第 1 步至第 4 步，但使用以下数据作为执行输入：

```
{
    "stores": [
```

```
      "Store A",
      "Store B",
      "Store C",
      "Store D"
    ]
}
```

第 3 步：查看状态机执行详细信息

在标题为执行 ID 的页面上，您可以查看执行结果并调试任何错误。

1. （可选）从执行详细信息 页面上显示的选项卡中进行选择，查看每个选项卡中显示的信息。例如，要查看状态机输入及其执行输出，请在执行摘要 部分选择执行输入和输出。
2. 如果状态机执行失败，请在错误消息上选择原因或显示步骤详细信息。有关错误的详细信息显示在步骤详细信息 部分。请注意，导致错误的步骤（名为GetListofFruits的Task状态）在图表视图和表格视图中突出显示。

> **Note**
>
> 由于GetListofFruits步骤是在Map状态内定义的，并且该步骤未能成功执行，因此状态步骤的**Map**状态显示为 “失败”。

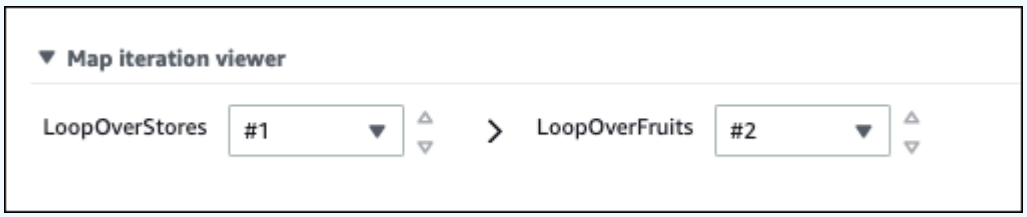
第 4 步：探索不同的视图模式

您可以选择首选模式来查看状态机工作流或执行事件历史记录。在这些视图模式 下可以执行的部分任务如下：

图表视图 – 在不同的 **Map** 状态迭代之间切换

如果您的 Map 状态有五次迭代，并且您想要查看第三次和第四次迭代的执行详细信息，请执行以下操作：

1. 选择要查看其迭代数据的 Map 状态。
2. 在 Map 迭代查看器中，选择要查看的迭代。迭代从零开始计数。要从五次迭代中选择第三次迭代，请从 Map 状态名称旁边的下拉列表中选择 #2。

> **Note**
>
> 如果状态机包含嵌套 Map 状态，Step Functions 会将父和子 Map 状态迭代显示为两个单独的下拉列表：
>
> 
>

3. （可选）如果一次或多次 Map 状态迭代未能执行或停止在中止状态，则可以查看有关失败迭代的详细信息。要查看这些详细信息，请在下拉列表的失败或已中止下选择受影响的迭代编号。

表格视图 – 在不同的 **Map** 状态迭代之间切换

如果您的 Map 状态有五次迭代，并且您想要查看第三次和第四次迭代的执行详细信息，请执行以下操作：

1. 选择要查看其不同迭代数据的 Map 状态。
2. 在 Map 状态迭代的树视图显示中，选择名为 #2 的迭代行，打开第三次迭代。同样，选择名为 #3 的行，打开第四次迭代。

表格视图 – 配置要显示的列

选择  。然后，在首选项对话框中，在选择可见列下选择要显示的列。

默认情况下，此模式显示名称、类型、状态、资源和之后开始列。

表格视图 – 筛选结果

通过应用一个或多个基于属性（如状态）或日期和时间范围的筛选条件，来限制显示的信息量。例如，要查看执行失败的步骤，请应用以下筛选条件：

1. 选择按属性筛选或按关键词搜索，然后在属性下选择状态。
2. 在运算符下，选择状态 =。

3. 选择状态 = Failed。
4. （可选）选择清除筛选条件，移除已应用的筛选条件。

事件视图 – 筛选结果

通过应用一个或多个基于属性（如类型）或日期和时间范围的筛选条件，来限制显示的信息量。例如，要查看执行失败的 Task 状态步骤，请应用以下筛选条件：

1. 选择按属性筛选或按关键词搜索，然后在属性下选择类型。
2. 在运算符下，选择类型 =。
3. 选择类型 = TaskFailed。
4. （可选）选择清除筛选条件，移除已应用的筛选条件。

活动视图-检查TaskFailed活动详情

选择TaskFailed事件 ID

旁边的，查看其详细信息，包括出现在下拉框中的输入、输出和资源调用。

Redriving执行

您可以使用redrive重启过去 14 天内未成功完成的标准工作流执行。其中包括执行失败、中止或超时的执行。

redrive执行时，将继续执行未成功步骤的失败执行，并使用相同的输入。 Step Functions 会保留成功步骤的结果和执行历史记录，并且在redrive执行时不会重新运行这些步骤。例如，假设您的工作流包含两个状态：一个Pass 状态，后跟一个 任务状态 状态。如果您的工作流执行在 Task 状态下失败，而您redrive执行，那么执行会重新安排时间，然后重新运行 Task 状态。

Redriven执行使用与原始执行尝试相同的状态机定义和执行 ARN。如果您原始执行尝试与版本、别名或两者相关联，则redriven执行将与相同的版本、别名或两者相关联。即使更新别名指向其他版本，redriven执行也会继续使用与原始执行尝试相关联的版本。由于redriven执行使用相同的状态机定义，因此如果您更新状态机定义，则必须启动新的执行。

redrive执行时，状态机级别的超时（如果已定义）将重置为 0。有关状态机级别超时的更多信息，请参阅 `TimeoutSeconds`。

执行redrives被视为状态转换。有关状态转换如何影响计费的信息，请参阅 Step Functions 定价。

主题

- 未成功执行的Redrive资格
- 单个状态的Redrive行为
- redrive执行的 IAM 权限
- 在控制台中Redriving执行
- 使用 API Redriving执行
- 检查redriven执行
- redriven执行的重试行为

未成功执行的Redrive资格

如果您原始执行尝试满足以下条件，则可以redrive执行：

- 在 2023 年 11 月 15 日或之后启动执行。在此日期之前开始的执行不符合redrive资格。
- 执行状态不是 SUCCEEDED。
- 工作流执行未超过 14 天的redrivable期限。Redrivable期限是指可以redrive给定执行的时间。这段时间从状态机完成执行的当天开始计算。
- 工作流执行未超过一年的最长开放时间。有关状态机执行配额的信息，请参见与状态机执行相关的配额。
- 执行事件历史记录计数小于 24999。Redriven执行会将其事件历史记录附加到现有的事件历史记录中。确保您的工作流执行包含少于 24,999 个事件，从而可以容纳 ExecutionRedriven 历史事件和至少一个其他历史事件。

单个状态的Redrive行为

根据工作流中失败的状态，所有失败状态的redrive行为会有所不同。下表描述了所有状态的redrive行为。

状态名称	Redrive执行行为
Pass	如果前面的步骤失败或状态机超时，Pass 状态将退出且不会在redrive上面执行。
任务状态	重新安排和启动 Task 状态。

状态名称	Redrive执行行为
	redrive重新运行 Task 状态的执行时，该状态的 `TimeoutSeconds` （如果已定义）将重置为 0。有关超时的更多信息，请参阅 Task 状态。
Choice	重新评估 Choice 状态规则。
Wait	如果状态指定的 `Timestamp` 或 `TimestampPath` 指向过去的时间戳，redrive会退出 Wait 状态并进入 `Next` 字段中指定的状态。
Succeed	进入 Succeed 状态的状态机执行不会redrive。
Fail	重新进入 Fail 状态并再次失败。
Parallel	仅重新安排和redrives失败或中止的分支。 如果状态由于 `States.DataLimitExceeded` 错误而失败，则会重新运行 Parallel 状态，包括在原始执行尝试中成功的分支。
内联 Map 状态	仅重新安排和redrives失败或中止的迭代。 如果状态由于 `States.DataLimitExceeded` 错误而失败，则会重新运行内联 Map 状态，包括在原始执行尝试中成功的迭代。
分布式 Map 状态	redrives Map Run 中未成功执行的子工作流。有关更多信息，请参阅Redriving Map Run。 如果状态因 `States.DataLimitExceeded` 错误而失败，则重新运行分布式 Map 状态。这包括在原始执行尝试中成功的子工作流。

redrive执行的 IAM 权限

Step Functions 需要适当的权限才能redrive执行。以下 IAM 策略示例授予状态机所需的最低权限，以便redriving执行。切记用特定资源信息替换## 文本。

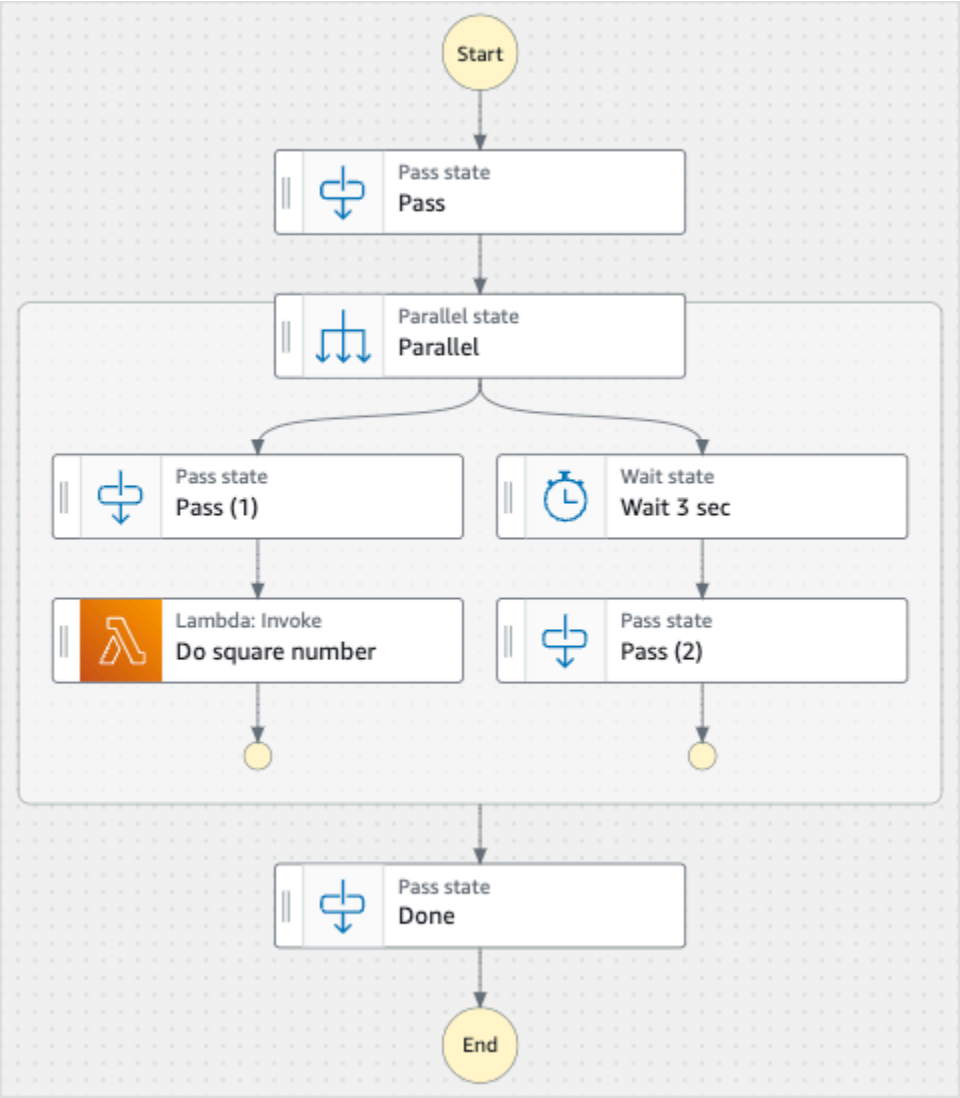
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:RedriveExecution"
            ],
            "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:*"
        }
    ]
}
```

有关redrive Map Run 所需权限的示例，请参阅redriving分布式 Map 的 IAM 策略示例。

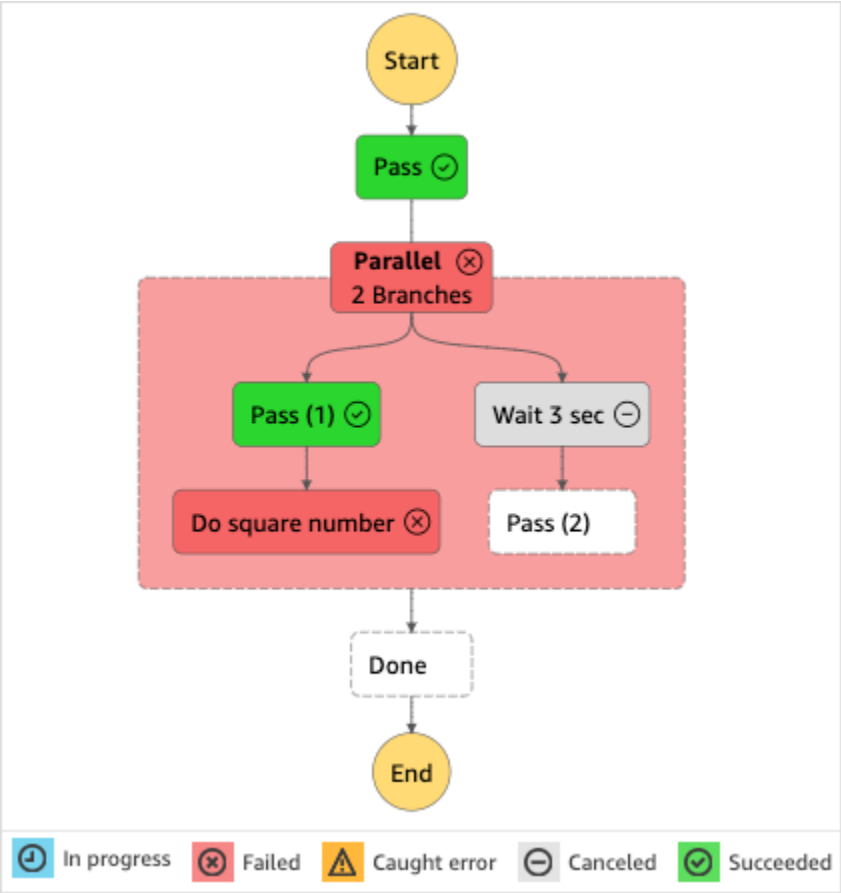
在控制台中Redriving执行

您可以从 Step Functions 控制台中redrive符合条件的执行。

例如，假设以下图像代表状态机的工作流图。



想象一下，您运行了这个状态机。下图显示了状态机执行图。



如下图所示，在 Parallel 状态下名为 Do square number 的 Lambda 调用 步骤返回了一个错误。这导致了 Parallel 状态失败。正在执行或未启动的分支将停止，状态机执行失败。

从控制台redrive执行

1. 打开 Step Functions 控制台，然后选择执行失败的现有状态机。
2. 在状态机详细信息页面的执行下，选择一个失败的执行实例。
3. 选择Redrive。
4. 在Redrive对话框中，选择Redrive执行。

> **Tip**
>
> 如果您在执行失败的执行详细信息 页面，请执行以下操作之一来redrive执行：
>
> - 选择恢复，然后从故障中Redrive。
> - 选择操作，然后选择Redrive。

请注意，redrive将使用相同的状态机定义和 ARN。它从原始执行失败的步骤开始继续执行。在这个例子中，这里的步骤是 Parallel 状态下的 Do square number 步骤和 Wait 3 sec 分支。在 Parallel 状态下重新开始执行这些未成功的步骤后，redrive将继续执行到 Done 步骤。

5. 选择该执行，打开执行详细信息 页面。

   在此页面上，您可以查看redriven执行的结果。例如，在执行摘要部分中，您可以看到Redrive计数，它表示执行redriven的次数。在事件部分，可以看到与redrive相关的执行事件附加到原始执行尝试的事件中。有关示例，请参阅 `ExecutionRedriven` 事件。

使用 API Redriving执行

您可以使用 RedriveExecutionAPI 进行redrive符合条件的执行。此 API 会从失败、中止或超时的步骤中重新启动标准工作流的未成功执行。

在 AWS Command Line Interface (AWS CLI) 中，运行以下命令以使redrive状态机执行失败。切记用特定资源信息替换*##* 文本。

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

检查redriven执行

您可以在控制台中或使用 API 检查redriven执行情况：GetExecutionHistory和DescribeExecution。

在控制台中检查redriven执行

1. 打开 Step Functions 控制台，然后选择已redriven执行的现有状态机。
2. 打开执行详细信息 页面。

   在此页面上，您可以查看redriven执行的结果。例如，在执行摘要部分中，您可以看到Redrive计数，它表示执行redriven的次数。在事件部分，可以看到与redrive相关的执行事件附加到原始执行尝试的事件中。有关示例，请参阅 `ExecutionRedriven` 事件。

使用 API 检查redriven执行

如果您已经redriven了状态机执行，则可以使用以下 API 之一来查看有关redriven执行的详细信息。切记用特定资源信息替换*##* 文本。

- GetExecutionHistory — 以事件列表的形式返回指定执行的历史记录。此 API 还会返回有关执行redrive尝试的详细信息（如果有）。

  在中 AWS CLI，运行以下命令。

  ```
  aws stepfunctions get-execution-history --execution-arn arn:aws:states:us-east-2:123456789012:execution:myStateMachine:foo
  ```

- DescribeExecution — 提供有关状态机执行的信息。这可以是与执行关联的状态机、执行输入和输出、执行redrive详细信息（如果有）以及相关的执行元数据。

  在中 AWS CLI，运行以下命令。

  ```
  aws stepfunctions describe-execution --execution-arn arn:aws:states:us-east-2:123456789012:execution:myStateMachine:foo
  ```

redriven执行的重试行为

如果您的redriven执行重新运行了已定义重试的 任务状态、Parallel 或内联 Map状态，这些状态的重试尝试计数将重置为 0。这允许redrive上的最大尝试次数。对于redriven执行，您可以使用控制台跟踪这些状态的单个重试尝试。

在控制台中检查单个重试尝试

1. 在 Step Functions 控制台的执行详细信息 页面上，选择redrive上已重试过的状态。
2. 选择重试次数和redrives选项卡。
3. 选择每次重试尝试旁边的

   ，查看其详细信息。如果重试尝试成功，则可以在下拉框中显示的 输出 中查看结果。

下图举例说明了在原始执行尝试和redrives执行尝试中对某个状态执行的重试次数。在此图像中，原始执行尝试和redrive执行尝试中进行了三次重试。第四次redrive尝试执行成功并返回输出 16。

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   }
```

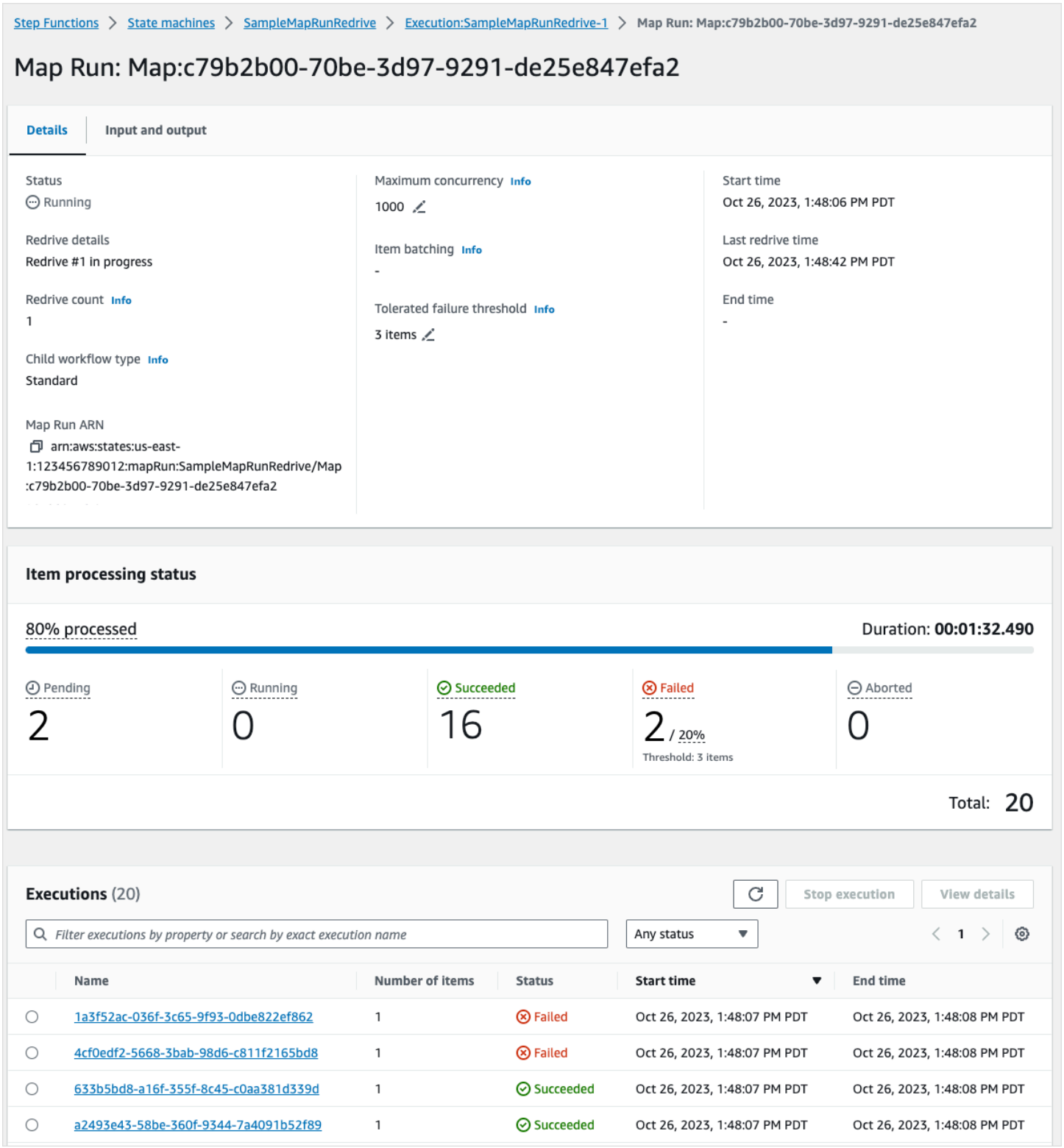
Formatted

检查分布式 Map 状态的 Map Run

当您运行分布式模式下的 Map 状态时，Step Functions 会创建一个 Map Run 资源。Map Run 是指分布式 Map 状态 启动的一组子工作流执行，以及控制这些执行的运行时设置。Step Functions 会为 Map Run 分配一个 Amazon 资源名称 (ARN)。您可以在 Step Functions 控制台中查看 Map Run。您也可以调用 DescribeMapRun API 操作。Map Run 还会向发送指标。 CloudWatch

Step Functions 控制台提供 Map Run 详细信息 页面，其中显示了与分布式 Map 状态 执行相关的所有信息。例如，您可以查看分布式 Map 状态 的执行、Map Run 的 ARN 以及由分布式 Map 状态启动的子工作流执行中处理项目的状态。您还可以查看所有子工作流执行的列表并访问其详细信息。此外，如果您 Map Run 经过了redriven，则可以在 Map Run 执行摘要部分中查看 Map Run 的redrive详细信息。例如，上次redrive时间。控制台采用控制面板格式显示此信息。

Map Run 详细信息 页面包含以下部分：



内容

- Map Run 执行摘要
- 错误消息

- 项目处理状态
- 执行清单
- Redriving Map Run
  - Map Run 中子工作流的Redrive资格
  - 子工作流执行redrive行为
  - Map Run redrive时使用的输入场景
  - redrive Map Run 的 IAM 权限
  - 在控制台中Redriving Map Run
  - 使用 API Redriving Map Run

Map Run 执行摘要

Map Run 执行摘要 部分显示在 Map Run详细信息 页面的顶部。本节为您提供分布式 Map 状态 的执行详细信息概览。此信息分为以下几个选项卡：

详细信息

显示分布式 Map 状态 的执行状态、Map Run ARN 和分布式 Map 状态 启动的子工作流执行类型等信息。您可以查看其他配置，例如 Map Run 容许的故障阈值和为子工作流执行指定的最大并发数。您也可以编辑这些配置。

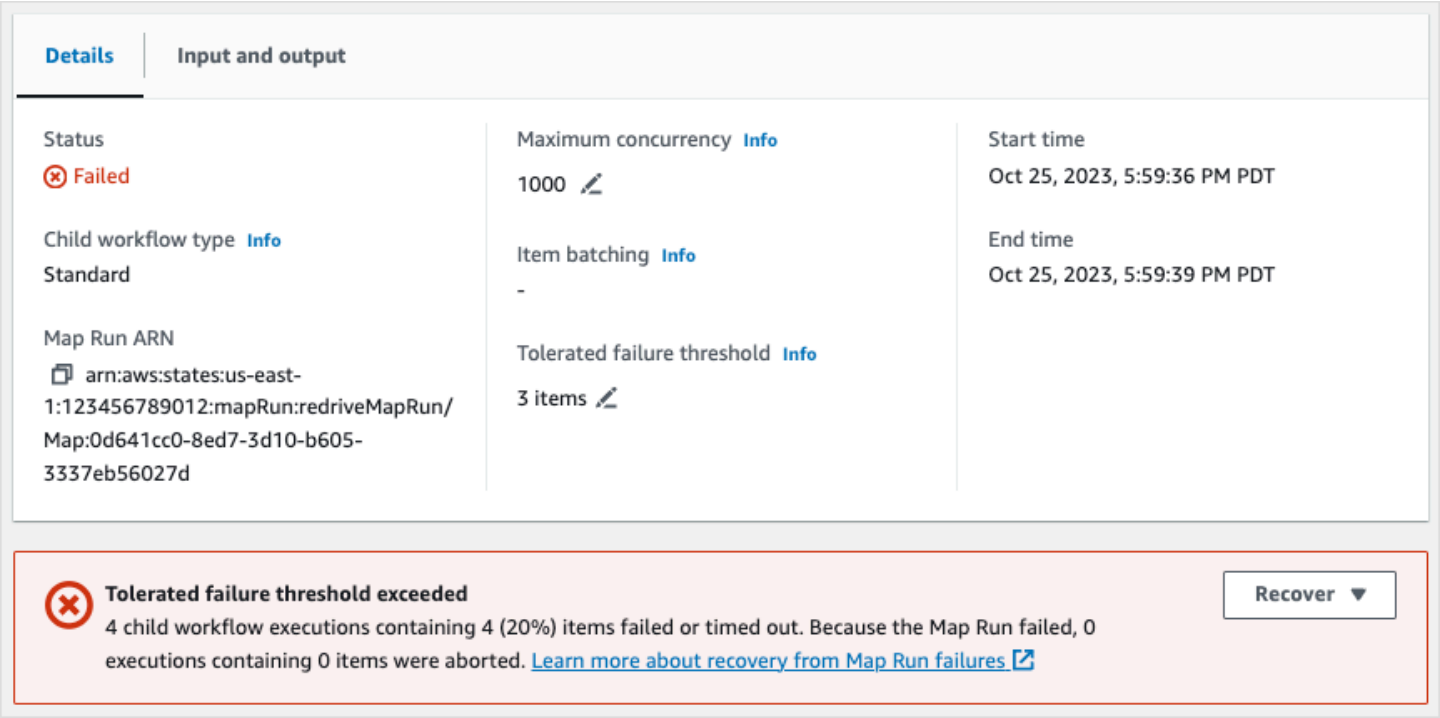
输入和输出

显示分布式 Map 状态 接收的输入及其生成的相应输出。例如，您可以查看输入数据集及其位置，以及应用于该数据集中各个数据项的输入筛选条件。如果您导出分布式 Map 状态 执行的输出，则此选项卡会显示包含执行结果的 Amazon S3 存储桶路径。如果未导出输出，则会将您指向父工作流的执行详细信息 页面，以便查看执行输出。

错误消息

如果您的 Map Run 失败，Map Run 详细信息 页面会显示一条错误消息，说明失败的原因。

通过此错误消息上的恢复下拉按钮，您可以redrive由该地图运行启动的未成功的子工作流执行，或重新启动父工作流的执行。有关更多信息，请参阅Redriving Map Run。



项目处理状态

项目处理状态部分显示在 Map Run 中处理的项目状态。例如，待处理表示子工作流执行尚未开始处理该项目。

项目状态取决于处理项目的子工作流执行的状态。如果子工作流执行失败、超时或用户取消执行，Step Functions 将不会收到有关该子工作流执行中项目处理结果的任何信息。该执行处理的所有项目都共享子工作流执行的状态。

例如，假设您要在两个子工作流执行中处理 100 个项目，其中每个执行都要处理一批 50 个项目。如果其中一个执行失败而另一个执行成功，则您将有 50 个成功的项目和 50 个失败的项目。

下表说明了所有项目可用的处理状态类型：

Status	描述
待处理	表示子工作流执行尚未开始处理的项目。如果 Map Run 在项目处理开始之前停止、失败或用户取消了执行，则该项目将保持待处理状态。 例如，如果 Map Run 失败，有 10 个项目待处理，则这 10 个项目将保持待处理状态。

Status	描述
正在运行	表示子工作流执行当前正在处理的项目。
已成功	表示子工作流执行成功处理了该项目。 成功的子工作流执行不能有任何失败的项目。如果数据集中的一个项目在执行过程中失败，则整个子工作流的执行将失败。
已失败	表示子工作流执行未能成功处理项目或超时。如果子工作流执行中处理的任何一个项目失败，则整个子工作流执行都将失败。 例如，假设一个处理 1000 个项目的子工作流执行。如果该数据集中的任何一个项目在执行期间失败，则 Step Functions 会将整个子工作流的执行视为失败。 redrive Map Run 时，处于此状态的项目计数会重置为 0。
已中止	表示子工作流程执行已开始处理该项目，但要么用户取消了执行，要么是 Step Functions 因 Map Run 失败而停止了执行。 例如，假设一个正在运行的子工作流执行正在处理 50 个项目。如果 Map Run 由于失败或用户取消执行而停止，则子工作流执行和所有 50 个项目的状态都将更改为已中止。 如果您使用快速类型的子工作流执行，则无法停止执行。 当您redrive启动快速类型的子工作流执行的 Map Run 时，具有此状态的项目计数将重置为 0。这是因为 Express 子工作流程是使用 StartExecutionAPI 操作而不是重新启动的。redriven

执行清单

执行部分列出了特定 Map Run 的所有子工作流执行。使用按准确的执行名称搜索字段可搜索特定的子工作流执行。您也可以使用任何状态下拉列表按状态筛选子工作流执行历史记录。要查看有关特定执行的详细信息，请从列表中选择子工作流执行，然后选择查看详细信息按钮，打开其执行详细信息 页面。

> **Important**
>
> 子工作流执行的保留政策为 90 天。超过此保留期的已完成子工作流执行不会显示在执行表格中。即使分布式 Map 状态或父工作流的运行时间持续超过保留期，也是如此。如果使用 `ResultWriter` 将分布式 Map 状态输出导出到 Amazon S3 存储桶，则可以查看这些子工作流执行的详细信息，包括结果。

> **Tip**
>
> 选择刷新按钮
>
> 
>
> 可查看所有子工作流执行的最新列表。

Redriving Map Run

您可以通过redriving父工作流，重新启动未成功的子工作流执行。redriven父工作流会redrives所有未成功的状态，包括分布式 Map。如果父工作流完成执行时没有与状态的 `<stateType>Exited` 事件相对应的 `<stateType>Entered` 事件，则父工作流会重新驱动未成功的状态。例如，如果事件历史记录不包含与 `MapStateEntered` 事件对应的 `MapStateExited` 事件，则可以redrive父工作流，以便 redrive Map Run 中所有未成功的子工作流执行。

当状态机没有访问 ItemReader、ResultWriter 或两者所需的权限时，Map Run 要么未启动，要么在原始执行尝试中失败。如果 Map Run 没有在父工作流的原始执行尝试中启动，则redriving父工作流将首次启动 Map Run。要解决这个问题，请在状态机角色中添加所需的权限，然后redrive父工作流。如果在未添加所需权限的情况下redrive父工作流，则会尝试启动新的 Map Run 运行，但会再次失败。有关可能需要的权限信息，请参阅使用分布式 Map 状态的 IAM 策略。

主题

- Map Run 中子工作流的Redrive资格
- 子工作流执行redrive行为
- Map Run redrive时使用的输入场景
- redrive Map Run 的 IAM 权限
- 在控制台中Redriving Map Run
- 使用 API Redriving Map Run

Map Run 中子工作流的Redrive资格

如果满足以下条件，您可以在 Map Run 中redrive未成功的子工作流执行：

- 在 2023 年 11 月 15 日或之后启动父工作流执行。在此日期之前开始的执行不符合redrive资格。
- 您没有超过给定 Map Run 的 1000 次redrives硬限制。如果已超过此限制，则会收到 `States.Runtime` 错误消息。
- 父工作流是redrivable。如果父工作流无法redrivable，则无法在 Map Run 中redrive子工作流。有关工作流redrive资格的更多信息，请参阅未成功执行的Redrive资格。
- Map Run 中标准类型的子工作流执行未超过 2.5 万个执行事件历史记录上限。超过事件历史记录上限的子工作流执行将计入容许的故障阈值并被视为失败。有关执行redrive资格的更多信息，请参阅未成功执行的Redrive资格。

在以下情况下，即使 Map Run 在原始执行尝试中失败，也会启动新的 Map Run，而不会redriven现有的地图运行：

- 由于 `States.DataLimitExceeded` 错误，Map Run 失败。
- 由于 JSON 数据插值错误 (`States.Runtime`)，Map Run 失败。例如，您在 OutputPath 中选择了一个不存在的 JSON 节点。

即使在父工作流停止或超时后，Map Run 仍可以继续运行。在这些情况下，redrive不会立即发生：

- Map Run 可能仍在取消正在进行的标准类型子工作流执行，或者等待快速类型子工作流执行完成执行。
- 如果您将 Map Run 配置为导出结果，它可能仍在向 ResultWriter 中写入结果。

在这些情况下，正在运行的 Map Run 会在尝试redrive之前完成其操作。

子工作流执行redrive行为

Map Run 中的redriven子工作流执行表现出下表所述的行为。

快速子工作流	标准子工作流
所有在原始执行尝试中失败或超时的子工作流程执行均使用 StartExecutionAPI 操作启动。首先运行 ItemProcessor 中的第一个状态。	所有在原始执行尝试中失败、超时或取消的子工作流执行均使用 RedriveExecutionAPI 操作redriven。这些子工作流程redriven来自导致其执行失败的最后一个状态。 ItemProcessor
未成功的执行可始终redriven。这是因为 Express 子工作流程执行总是使用 StartExec ution API 操作作为新执行启动。	未成功的标准子工作流执行并不总是可以redriven。如果执行无法redrivable，就不会再尝试执行。执行的最后一个错误或输出是永久性的。当执行的历史事件超过 2.5 万个，或可redr ivable的 14 天期限已过，就有可能出现这种情况。 redrivable如果父工作流执行在 14 天内关闭，但子工作流执行在 14 天前关闭，则标准子工作流执行可能无法。
快速子工作流执行使用与原始执行尝试相同的执行 ARN，但无法明确识别其单独的redrives。	标准子工作流执行使用与原始执行尝试相同的执行 ARN。您可以在控制台redrives中使用 API（例如GetExecutionHistory和 DescribeE xecution）清楚地识别个人身份。有关更多信息，请参阅 检查redriven执行。

如果您redriven一个 Map Run，并且已达到其并发数上限，则该 Map Run 中的子工作流执行会过渡到待处理状态。Map Run 的执行状态也会转换为待redrive状态。在指定的并发数上限允许运行更多子工作流执行之前，该执行将一直处于待redrive状态。

例如，工作流中分布式 Map 的并发数上限为 3000，而需要重新运行的子工作流数量为 6000。这会导致 3000 个子工作流并行运行，而其余 3000 个工作流程仍处于待重启状态。在第一批 3000 个子工作流完成执行后，剩余的 3000 个子工作流才会开始运行。

当 Map Run 完成执行或中止时，处于待redrive状态的子工作流执行计数将重置为 0。

Map Run redrive时使用的输入场景

根据您在原始执行尝试中向分布式 Map 提供输入的方式，redriven Map Run 将使用下表中描述的输入。

原始执行尝试中的输入	Map Run redrive时使用的输入
从上一状态传递的输入或执行输入。	redriven Map Run 使用相同的输入。
由于以下条件之一为 true，使用 ItemReader 传递的输入和 Map Run 未启动子工作流执行： • Map Run 因 States.ItemReaderFailed 错误而失败。 • Map Run 因 States.ResultWriterFailed 错误而失败。 • 在 Map Run 启动之前，父工作流执行超时或取消。	redriven Map Run 使用 Amazon S3 存储桶中的输入。
使用传递的输入 ItemReader。启动或尝试启动子工作流执行后，Map Run 失败。	redriven Map Run 使用的输入与原始执行尝试中提供的输入相同。

redrive Map Run 的 IAM 权限

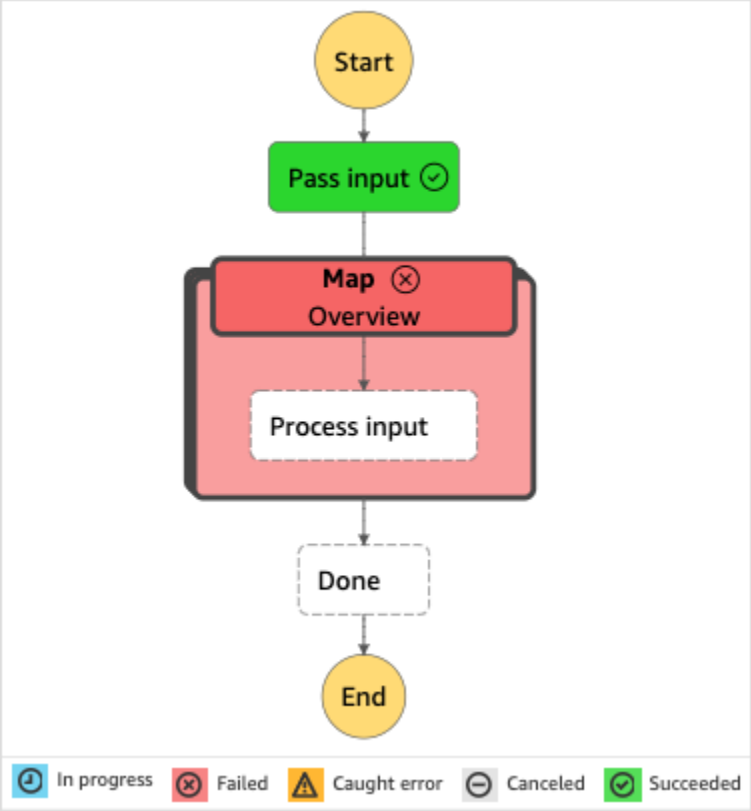
Step Functions 需要适当的权限才能redrive Map Run。以下 IAM 策略示例向状态机授予了redriving Map Run 所需的最低权限。切记用特定资源信息替换*## 文本。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
```

```
}
```

在控制台中Redriving Map Run

下图显示了包含分布式 Map 的状态机执行图。Map Run 失败，导致此执行失败。要redrive Map Run，必须redrive父工作流。



从控制台中redrive Map Run 的操作步骤

1. 打开 Step Functions 控制台，然后选择一个包含执行失败的分布式 Map 的现有状态机。
2. 在状态机详细信息页面的执行下，选择此状态机的失败执行实例。
3. 选择Redrive。
4. 在Redrive对话框中，选择Redrive执行。

> **Tip**
>
> 您也可以从执行详细信息 或 Map Run 详细信息 页面redrive Map Run。
> 如果您在执行详细信息 页面上，请进行以下操作之一来redrive执行：
>
> - 选择恢复，然后从故障中Redrive。

- 选择操作，然后选择Redrive。

如果您在 Map Run 详细信息 页面上，请选择恢复，然后选择从故障中Redrive。

请注意，redrive将使用相同的状态机定义和 ARN。它从原始执行失败的步骤开始继续执行。在此示例中，失败的步骤是名为 Map 的分布式 Map 步骤和其中的处理输入步骤。重新启动 Map Run 未成功的子工作流执行后，redrive将继续执行完成步骤。

5. 在执行详细信息 页面中，选择 Map Run，查看redriven Map Run 的详细信息。

   在此页面上，您可以查看redriven执行的结果。例如，在该 Map Run 执行摘要部分中，您可以看到Redrive计数，它表示redriven Map Run 的次数。在事件部分，可以看到与redrive相关的执行事件附加到原始执行尝试的事件中。有关示例，请参阅 `MapRunRedriven` 事件。

完成 Map Run 后，您可以在控制台中或使用GetExecutionHistory和 DescribeExecutionAPI 操作查看其redrive详细信息。redriven有关检查redriven执行的更多信息，请参阅检查redriven执行。

使用 API Redriving Map Run

您可以使用父工作流上的 RedriveExecution API redrive符合条件的 Map Run。此 API 可重启 Map Run 中未成功的子工作流执行。

在 AWS Command Line Interface (AWS CLI) 中，运行以下命令来redrive未成功的状态机执行。切记用特定资源信息替换*##* 文本。

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

完成 Map Run 后，您可以在控制台中或使用 DescribeMapRunAPI 操作查看其redrive详细信息。redriven要在 Map Run 中查看标准工作流程执行的redrive详细信息，您可以使用GetExecutionHistory或 DescribeExecutionAPI 操作。有关检查redriven执行的更多信息，请参阅the section called “检查redriven执行”。

如果在父工作流上启用了日志记录功能，则可以在 Step Functions 控制台中检查 Map Run 中快速工作流执行的redrive详细信息。有关更多信息，请参阅 使用 CloudWatch Logs 进行日志记录。

Step Functions 中的错误处理

除 `Pass` 和 `Wait` 状态之外的所有状态都可能遇到运行时错误。有多种原因可导致错误发生，例如以下示例：

- 状态机定义问题（例如，`Choice` 状态中没有匹配规则）
- 任务失败（例如，AWS Lambda 函数中的异常）
- 临时性问题（例如，网络分区事件）

默认情况下，当某个状态报告错误时，AWS Step Functions 会导致执行完全失败。

> **Tip**
>
> 要在 AWS 账户中部署包含错误处理的工作流示例，请参阅《AWS Step Functions 研讨会》模块 8 - 错误处理。

主题

- 错误名称
- 出错后重试
- 回退状态
- 使用 Retry 和使用 Catch 的状态机示例

错误名称

Step Functions 使用区分大小写的字符串（称为错误名称）来识别 Amazon States Language 中的错误。Amazon States Language 定义了一组内置字符串用于命名广为人知的错误，均以 `States.` 前缀开头。

`States.ALL`

一个与任何已知错误名称匹配的通配符。

> **Note**
>
> 此错误类型无法捕获 States.DataLimitExceeded 终端错误类型和运行时错误类型。有关这些错误类型的更多信息，请参见 States.DataLimitExceeded 和 States.Runtime。

States.DataLimitExceeded

Step Functions 会在以下条件下报告 States.DataLimitExceeded 异常：

- 当连接器的输出大于有效负载大小配额时。
- 当状态的输出大于有效负载大小配额时。
- Parameters 处理后，状态的输入大于有效负载大小配额时。

有关配额的更多信息，请参阅配额。

> **Note**
>
> 这是一个无法被 States.ALL 错误类型捕获的终端错误。

States.ExceedToleratedFailureThreshold

Map状 态失败是因为失败项目的数量超过了状态机定义中指定的阈值。有关更多信息，请参阅分布式 Map 状态容许的故障阈值。

States.HeartbeatTimeout

Task 状态未能发送检测信号的时间超过 HeartbeatSeconds 值。

> **Note**
>
> 此错误仅出现在 Catch 和 Retry 字段中。

States.IntrinsicFailure

尝试在有效负载模板中调用内置函数失败。

States.ItemReaderFailed

Map 状态因无法读取 ItemReader 字段中指定的项目来源而失败。有关更多信息，请参阅ItemReader。

States.NoChoiceMatched

Choice 状态未能将输入与选项规则中定义的条件相匹配，并且未指定默认过渡。

States.ParameterPathFailure

尝试替换状态 Parameters 字段中名称以路径结尾的 .$ 字段失败。

States.Permissions

Task 状态因没有足够的权限运行指定代码而失败。

States.ResultPathMatchFailure

Step Functions 未能将状态的 ResultPath 字段应用于该状态接收到的输入。

States.ResultWriterFailed

Map 状态因其无法将结果写入 ResultWriter 字段中指定的目的地而失败。有关更多信息，请参阅ResultWriter。

States.Runtime

执行因某些无法处理的异常而失败。这些通常是由运行时的错误引起的，例如尝试在 null JSON 有效负载上应用 InputPath 或 OutputPath。States.Runtime 错误不可检索，并且始终会导致执行失败。在 States.ALL 上重试或捕获不会捕获 States.Runtime 错误。

States.TaskFailed

一个在执行期间失败的 Task 状态。在 retry 或 catch 中使用时，States.TaskFailed 充当通配符，可匹配除 States.Timeout 之外的任何已知错误名称。

States.Timeout

一个 Task 状态，该状态要么运行时间长度超过 TimeoutSeconds 值，要么在超过 HeartbeatSeconds 值的时段长度中未能发送检测信号。

此外，如果状态机的运行时间超过指定 TimeoutSeconds 值，则执行失败并显示 States.Timeout 错误。

状态可以报告具有其他名称的错误。但是，这些错误名称不能以 States. 前缀开头。

作为最佳实操，确保生产代码可以处理 AWS Lambda 服务异常（`Lambda.ServiceException` 和 `Lambda.SdkClientException`）。有关更多信息，请参阅处理 Lambda 服务异常。

> **Note**
>
> Lambda 中未处理的错误在错误输出中报告为 `Lambda.Unknown`。这些错误包括内存不足错误和函数超时。您可以匹配 `Lambda.Unknown`、`States.ALL` 或 `States.TaskFailed` 来处理这些错误。当 Lambda 达到最大调用次数时，会出现 `Lambda.TooManyRequestsException` 错误。有关 Lambda 函数错误的更多信息，请参阅《AWS Lambda 开发者指南》中的错误处理和自动重试。

出错后重试

`Task`、`Parallel` 和 `Map` 状态可以有名为 `Retry` 的字段，其值必须是称为重试器 的对象数组。单个重试器表示特定的重试次数，通常是以递增的时间间隔。

当其中一个状态报告错误并且有一个 `Retry` 字段时，Step Functions 会按照数组中列出的顺序扫描重试器。当错误名称出现在某个重试器的 `ErrorEquals` 字段中时，状态机就会按照 `Retry` 字段中的定义进行重试。

如果redriven执行重新运行了已定义重试的 任务状态、Parallel 或内联 Map 状态，这些状态的重试尝试计数将重置为 0，以便允许redrive上的最大尝试次数。对于redriven执行，您可以使用控制台跟踪这些状态的单个重试尝试。有关更多信息，请参阅Redriving执行中的redriven执行的重试行为。

重试器包含以下字段：

> **Note**
>
> 重试被视为一种状态转换。有关状态转换如何影响计费的信息，请参阅 Step Functions 定价。

`ErrorEquals`（必填）

一个匹配错误名称的非空字符串数组。当状态报告错误时，Step Functions 会全面扫描重试器。当错误名称出现在此数组中时，它实施该重试器描述的重试策略。

`IntervalSeconds`（可选）

一个正整数，表示第一次重试尝试之前等待的秒数（默认值为 1）。`IntervalSeconds` 的最大值为 99999999。

MaxAttempts（可选）

一个正整数，表示重试的最大次数 (默认值为 3)。如果错误重复发生超过指定次数，则停止重试并恢复正常错误处理。值为 0 表示永不重试错误。 MaxAttempts 的最大值为 99999999。

BackoffRate（可选）

每次重试后，IntervalSeconds 所表示的重试间隔增加的倍数。默认情况下，BackoffRate 值会增加 2.0。

例如，假设 IntervalSeconds 为 3，MaxAttempts 为 3，BackoffRate 为 2。第一次重试尝试在错误发生三秒后进行。第二次重试尝试在第一次重试尝试六秒后进行。而第三次重试尝试在第二次重试尝试 12 秒后进行。

MaxDelaySeconds（可选）

一个正整数，用于设置重试间隔可增加的最大值（以秒为单位）。该字段与 BackoffRate 字段配合使用会很有帮助。在此字段中指定的值将限制应用于每次连续重试尝试的回退率倍数所产生的指数等待时间。您必须为 MaxDelaySeconds 指定一个大于 0 小于 31622401 的值。

如果您未指定此值，则 Step Functions 不会限制两次重试尝试之间的等待时间。

JitterStrategy（可选）

一个字符串，用于确定是否在连续重试尝试之间的等待时间中包含抖动。抖动会将同时重试尝试的次数分散到一个随机的延迟时间间隔内，从而减少重试次数。此字符串可接受 FULL 或 NONE 作为其值。默认值为 NONE。

例如，您将 MaxAttempts 设置为 3，IntervalSeconds 设置为 2，BackoffRate 设置为 2。第一次重试尝试在错误发生两秒后进行。第二次重试在第一次重试尝试后四秒进行，第三次重试在第二次重试尝试后八秒进行。如果将 JitterStrategy 设置为 FULL，则第一次重试间隔在 0 至 2 秒之间随机进行，第二次重试间隔在 0 至 4 秒之间随机进行，第三次重试间隔在 0 至 8 秒之间随机进行。

重试字段示例

本节包含以下 Retry 字段示例。

- Retry with BackoffRate
- Retry with MaxDelaySeconds
- Retry all errors except States.Timeout

- Complex retry scenario

> **Tip**
>
> 要在 AWS 账户中部署错误处理工作流示例，请参阅《AWS Step Functions 研讨会》中的错误处理模块。

示例 1 – 使用 BackoffRate 重试

下面的 `Retry` 示例进行了两次重试，第一次重试在等待 3 秒后进行。根据指定的 `BackoffRate`，Step Functions 会增加每次重试的间隔时间，直到达到重试尝试的最大次数。在下面的示例中，第二次重试在第一次重试 3 秒后开始。

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "IntervalSeconds": 3,
   "MaxAttempts": 2,
   "BackoffRate": 1
} ]
```

示例 2 – 使用 MaxDelaySeconds 重试

以下示例进行了三次重试尝试，并将 `BackoffRate` 产生的等待时间限制为 5 秒。第一次重试在等待 3 秒钟后进行。由于 `MaxDelaySeconds` 设置了最长等待时间限制，第二次和第三次重试将在前一次重试后等待 5 秒后进行。

```
"Retry": [ {
    "ErrorEquals": [ "States.Timeout" ],
    "IntervalSeconds": 3,
    "MaxAttempts": 3,
    "BackoffRate":2,
    "MaxDelaySeconds": 5,
    "JitterStrategy": "FULL"
} ]
```

如果未设置 `MaxDelaySeconds`，第二次重试将在第一次重试后 6 秒进行，第三次重试将在第二次重试后 12 秒进行。

示例 3 – 重试除 States.Timeout 之外的所有错误

重试器的 ErrorEquals 字段中显示的保留名称 States.ALL 是一个通配符，与所有错误名称匹配。它必须单独显示在 ErrorEquals 数组中，并且必须显示在 Retry 数组的最后一个重试器中。名称 States.TaskFailed 也是一个通配符，可匹配除 States.Timeout 以外的任何错误。

下面的 Retry 字段示例重试了除 States.Timeout 以外的任何错误。

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "MaxAttempts": 0
}, {
   "ErrorEquals": [ "States.ALL" ]
} ]
```

示例 4 – 复杂的重试场景

在单状态执行的上下文中，重试器的参数应用到对该重试器的所有访问。

考虑以下 Task 状态。

```
"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"
   } ]
}
```

此任务连续失败四次，输出以下错误名称：ErrorA、ErrorB、ErrorC 和 ErrorB。出现以下结果：

- 前两个错误与第一个重试器匹配，导致等待 1 秒和 2 秒。
- 第三个错误与第二个重试器匹配，导致等待 5 秒。

- 第四个错误也与第一个重试器匹配。但是，针对该特定错误的两次重试 (MaxAttempts) 已达到最大值。因此，该重试器失败，执行会通过 Catch 字段将工作流重定向到 Z 状态。

回退状态

Task、Map 和 Parallel 状态可以具有名为 Catch 的字段。此字段的值必须是称为捕获器 的对象的数组。

捕获器包含以下字段。

ErrorEquals（必填）

一个与错误名称匹配的非空字符串数组，由相同名称的重试器字段完全按其原样指定。

Next（必填）

一个字符串，必须与状态机的状态名称之一完全匹配。

ResultPath（可选）

一个路径，确定捕获器将什么输入发送到在 Next 字段中指定的状态。

当某个状态报告错误并且没有 Retry 字段或者重试无法解决错误时，Step Functions 按照数组中列出的顺序全面地扫描捕获器。当错误名称显示在捕获器的 ErrorEquals 字段中时，状态机转换为在 Next 字段中指定的状态。

捕获器的 ErrorEquals 字段中显示的保留名称 States.ALL 是一个通配符，与所有错误名称匹配。它必须单独显示在 ErrorEquals 数组中，并且必须显示在 Catch 数组的最后一个捕获器中。名称 States.TaskFailed 也是一个通配符，可匹配除 States.Timeout 以外的任何错误。

以下示例说明的是一个 Catch 字段，该字段在 Lambda 函数输出未处理的 Java 异常时转换为名为 RecoveryState 的状态。否则，该字段转换为 EndState 状态。

```
"Catch": [ {
   "ErrorEquals": [ "java.lang.Exception" ],
   "ResultPath": "$.error-info",
   "Next": "RecoveryState"
}, {
   "ErrorEquals": [ "States.ALL" ],
   "Next": "EndState"
} ]
```

> **Note**
>
> 每个捕获器可以指定多个要处理的错误。

错误输出

Step Functions 转换为 Catch 名称中指定的状态时，对象通常包含字段 Cause。此字段的值是人类可读格式的错误说明。该对象称为错误输出。

在本示例中，第一个捕获器包含一个 ResultPath 字段。其工作方式类似于状态顶级中的 ResultPath 字段，有两种可能的结果：

- 它获取状态执行的结果，并覆盖状态的全部或部分输入。
- 它获取结果并将其添加到输入中。在由捕获器处理错误时，状态执行的结果是错误输出。

因此，在本示例中，对于第一个捕获器，如果输入中还没有名为 error-info 的字段，捕获器就会将错误输出添加到输入中。然后，捕获器会将整个输入发送到 RecoveryState。对于第二个捕获器，错误输出会覆盖输入，捕获器只会将错误输出发送到 EndState。

> **Note**
>
> 如果您不指定 ResultPath 字段，则它会默认为 $，这会选择和覆盖整个输入。

当状态同时包含 Retry 和 Catch 字段时，Step Functions 会首先使用任何合适的重试器。如果重试策略无法解决错误，Step Functions 会应用匹配的捕获器转换。

原因有效负载和服务集成

捕获器会返回一个字符串有效负载作为输出。在使用诸如 Amazon Athena 或 AWS CodeBuild 之类的服务集成时，您可能需要将 Cause 字符串转换为 JSON。下面这个带有内置函数的 Pass 状态示例展示了如何将 Cause 字符串转换为 JSON。

```
"Handle escaped JSON with JSONtoString": {
  "Type": "Pass",
  "Parameters": {
    "Cause.$": "States.StringToJson($.Cause)"
```

```
  },
  "Next": "Pass State with Pass Processing"
},
```

使用 Retry 和使用 Catch 的状态机示例

以下示例中定义的状态机假定存在两个 Lambda 函数：一个始终失败，另一个等待足够长的时间，以允许发生状态机中定义的超时。

这是始终失败的 Lambda 函数的定义，返回消息 `error`。在后面的状态机示例中，此 Lambda 函数名为 `FailFunction`。有关创建 Lambda 函数的信息，请参阅第 1 步：创建 Lambda 函数部分。

```
exports.handler = (event, context, callback) => {
    callback("error");
};
```

这是一个休眠 10 秒的 Lambda 函数的定义。在后面的状态机示例中，此 Lambda 函数名为 `sleep10`。

> **Note**
>
> 当您在 Lambda 控制台中创建此 Lambda 函数时，请记住在 Advanced settings 部分中将 Timeout 值从 3 秒 (默认值) 更改为 11 秒。

```
exports.handler = (event, context, callback) => {
    setTimeout(function(){
    }, 11000);
};
```

使用 Retry 处理失败

此状态机使用 `Retry` 字段重试失败的函数，并输出错误名称 `HandledError`。该函数重试两次，在两次重试之间使用指数回退。

```
{
   "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
      }
   }
}
```

此变体使用预定义的错误代码 States.TaskFailed，这与 Lambda 函数输出的任意错误匹配。

```
{
   "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**
>
> 作为最佳实操，引用 Lambda 函数的任务应处理 Lambda 服务异常。有关更多信息，请参阅处理 Lambda 服务异常。

使用 Catch 处理失败

此示例使用 Catch 字段。当 Lambda 函数输出错误时，将会捕获错误，并且状态机转换为 fallback 状态。

```
{
   "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
      }
   }
}
```

此变体使用预定义的错误代码 States.TaskFailed，这与 Lambda 函数输出的任意错误匹配。

```
{
   "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
      }
   }
}
```

使用 Retry 处理超时

此状态机根据 TimeoutSeconds 中指定的超时值，使用 Retry 字段重试超时的 Task 状态。Step Functions 在此 Task 状态下重试两次 Lambda 函数调用，两次重试之间会出现指数回退。

```
{
   "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
      }
   }
}
```

使用 Catch 处理超时

此示例使用 Catch 字段。在出现超时的情况下，状态机会转换为 fallback 状态。

```
{
   "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**
>
> 可以通过使用 `ResultPath` 来保留状态输入以及错误。请参阅 ResultPath 用于在 a 中同时包含错误和输入 `Catch`。

从其他服务调用 AWS Step Functions

您可以配置其他几个服务来调用状态机。根据状态机的工作流类型，您可以异步或同步调用状态机。要同步调用状态机，请使用 `StartSyncExecution` API 调用或 Amazon API Gateway 与快速工作流集成。使用异步调用时，Step Functions 会暂停工作流执行，直到返回任务令牌。但是，等待任务令牌会使工作流同步。

您可以配置为调用 Step Functions 的服务包括：

- AWS Lambda，使用 `StartExecution` 调用。
- Amazon API Gateway
- Amazon EventBridge
- AWS CodePipeline
- AWS IoT 规则引擎
- AWS Step Functions

Step Functions 调用受 StartExecution 配额的约束。有关更多信息，请参阅：

- 配额

Step Functions 中的读取一致性

在 AWS Step Functions 中，状态机更新具有最终一致性。几秒钟内的所有 StartExecution 调用都将使用更新后的定义和 roleArn（IAM 角色的 Amazon 资源名称）。但在调用 UpdateStateMachine 后立即启动的执行可能会使用以前的状态机定义和 roleArn。

有关更多信息，请参阅下列内容：

- 《AWS Step Functions API 参考》中的 UpdateStateMachine
- 入门 AWS Step Functions中的更新工作流。

Step Functions 中的标记

AWS Step Functions 支持标记状态机（标准和快速）和活动。这可帮助您跟踪和管理与资源关联的成本，并增强 AWS Identity and Access Management (IAM) 策略中的安全性。标记 Step Functions 资源后，可以使用 AWS Resource Groups 对其进行管理。有关资源组更多信息，请参阅 AWS Resource Groups 用户指南。

对于基于标签的授权，状态机执行资源（如下例所示）会继承与状态机关联的标签。

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

当调用 DescribeExecution 或其他指定执行资源 ARN 的 API 时，Step Functions 会使用与状态机相关的标记来接受或拒绝请求，同时执行基于标记的授权。这有助于在状态机级别允许或拒绝对状态机执行的访问。

要查看与资源标记相关的限制，请参阅与标记相关的限制。

主题

- 成本分配的标记
- 标记以提高安全性
- 在 Step Functions 控制台中查看和管理标签

- 使用 Step Functions API 操作管理标签

成本分配的标记

要组织并标识您的 Step Functions 资源以进行成本分配，您可以添加用于确定状态机或活动的用途的元数据标签。这在您拥有许多资源时尤其有用。您可以使用成本分配标签组织 AWS 账单，以反映您自己的成本结构。要执行此操作，请注册以获取 AWS 账户账单来包含标签键和值。有关更多信息，请参阅《AWS Billing用户指南》中的设置月度成本分配报告。

例如，您可以添加表示 Step Functions 资源的成本中心和用途的标签，如下所示。

资源	键	Value
StateMachine1	Cost Center	34567
	Application	Image processing
StateMachine2	Cost Center	34567
	Application	Rekognition processing
Activity1	Cost Center	12345
	Application	Legacy database

此标记方案可让您将执行相关任务的两个状态机分组到同一成本中心，并使用不同的成本分配标签来标记不相关的活动。

标记以提高安全性

IAM 支持基于标签控制对资源的访问。要基于标签控制访问，请在 IAM 策略的条件元素中提供有关您的资源标签的信息。

例如，您可能会限制对下面这样的所有 Step Functions 资源的访问：在这些资源包含的标签中，具有键 `environment` 和值 `production`。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "states:TagResource",
                "states:DeleteActivity",
                "states:DeleteStateMachine",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

有关更多信息，请参阅《IAM 用户指南》中的使用标签控制访问。

在 Step Functions 控制台中查看和管理标签

Step Functions 可让您在 Step Functions 控制台中查看和管理状态机的标签。在状态机的 Details (详细信息) 页面中，选择 Tags (标签)。在这里，您可以查看与状态机关联的现有标签。

> **Note**
>
> 要管理活动的标签，请参阅使用 Step Functions API 操作管理标签。

要添加或删除与状态机关联的标签，请选择 Manage Tags (管理标签) 按钮。

1. 浏览到状态机的详细信息页面。
2. 选择执行和定义旁边的标签。
3. 选择 Manage tags（管理标签）。
   - 要修改现有标签，请编辑 Key (键) 和 Value (值)。
   - 要删除现有标签，请选择 Remove tag (删除标签)。
   - 要添加新标签，请选择 Add tag (添加标签)，然后输入 Key (键) 和 Value (值)。
4. 选择保存。

使用 Step Functions API 操作管理标签

要使用 Step Functions API 管理标签，请使用以下 API 操作：

- ListTagsForResource
- TagResource
- UntagResource

AWS Step Functions 工作流程工作室

Workflow Studio for AWS Step Functions 是一款低代码的可视化工作流设计器，可让您通过编排 AWS 服务来创建无服务器工作流程。使用其 drag-and-drop 功能或内置的代码编辑器，您可以创建和编辑工作流程，控制如何筛选或转换每种状态的输入和输出，以及配置错误处理。拖放状态来构建工作流时，Workflow Studio 会验证您的操作并自动生成代码。您可以在代码编辑器中查看生成的代码或更新状态机定义。完成后，您可以保存工作流并运行，然后在 Step Functions 控制台中检查结果。您可以直观地添加和修改工作流，以编排应用程序中的多项服务。

要使用 Step Functions Workflow Studio，你需要一个 AWS 账户和证书，以便为你要使用的任何资源提供正确权限。有关更多信息，请参阅 入门的先决条件 AWS Step Functions。

> **Note**
>
> Workflow Studio 不支持 Internet Explorer 11。如果您使用的是 Internet Explorer 11，并在使用 Workflow Studio 时遇到问题，请尝试使用其他浏览器。

在 Step Functions 中创建或编辑工作流时，可以通过 Step Functions 控制台访问 Workflow Studio。您也可以在 AWS 应用程序编辑器 中访问 Workflow Studio。Application Composer 中的 Workflow Studio 提供一个视觉 IaC 环境，可让您轻松地将工作流整合到使用 AWS CloudFormation 模板等 IaC 工具构建的无服务器应用程序中。使用 Application Composer 中的 Workflow Studio，您可以利用 AWS CloudFormation 模板构建工作流。在 Application Composer 中，您可以添加新工作流、修改现有工作流，以及将各个工作流步骤连接到其他应用程序资源。Application Composer 会自动创建和更新所需的 CloudFormation 资源和配置。这可以帮助您在一个位置集中创建和管理工作流中使用的所有资源。这还可以帮助您加快从工作流原型设计到生产部署的过程。

此外，在 Application Composer 中使用 Workflow Studio 时，您还可以直接连接到本地项目。这有助于您在集成式开发环境 (IDE) 中工作，同时使用可视画布。有关更多信息，请参阅 使用 Application Composer 中的 Workflow Studio。

主题

- 界面概述
- 使用 Workflow Studio
- 为状态配置输入和输出
- Workflow Studio 中的执行角色
- 错误处理

- 教程：学习使用 AWS Step Functions Workflow Studio

界面概述

Workflow Studio for AWS Step Functions 是一款低代码的可视化工作流设计器，可让您通过编排来创建无服务器工作流程。 AWS 服务借助拖放特征，Workflow Studio 可让您轻松构建、编辑和可视化工作流原型。Workflow Studio 还提供了一个内置的代码编辑器，用于在 Step Functions 控制台中使用 Amazon States Language (ASL) 编写和编辑工作流定义。

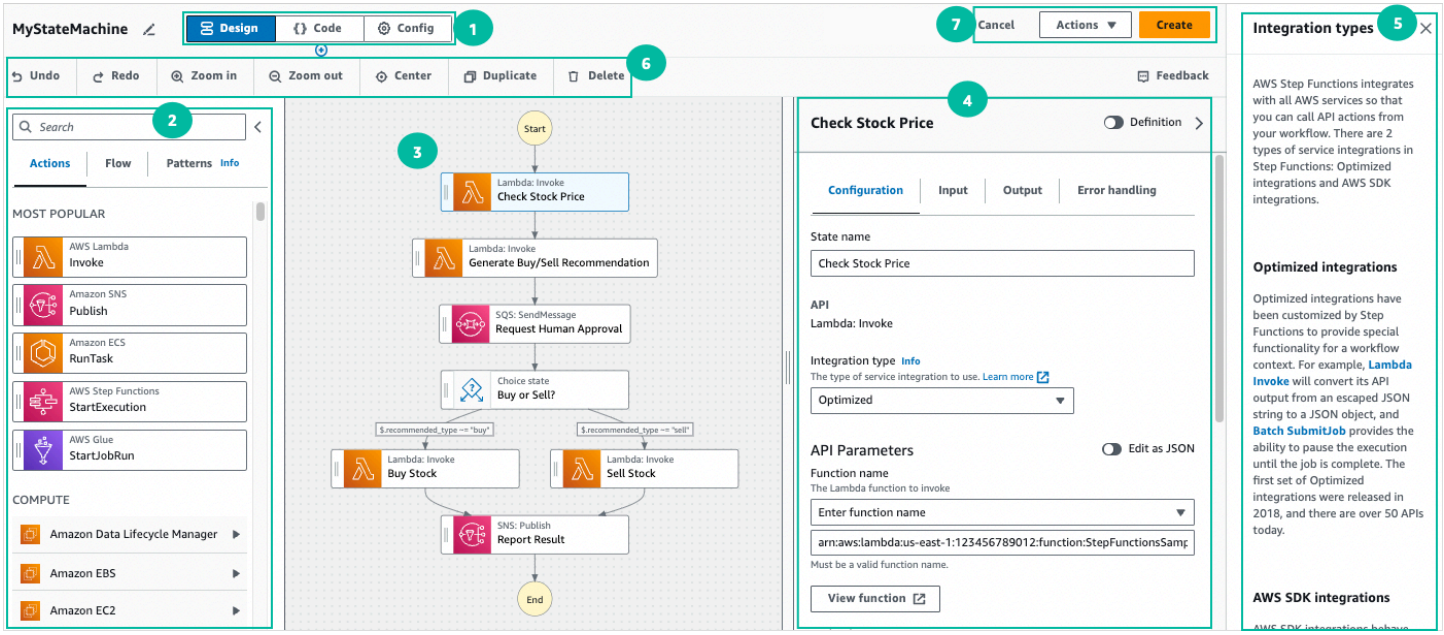
为了帮助您构建和可视化工作流、编辑其定义和管理其配置，Workflow Studio 提供了三种模式：设计、代码和配置。以下各部分详细描述了这些模式。

本主题内容

- 设计模式
- 代码模式
- 配置模式
- 键盘快捷键

设计模式

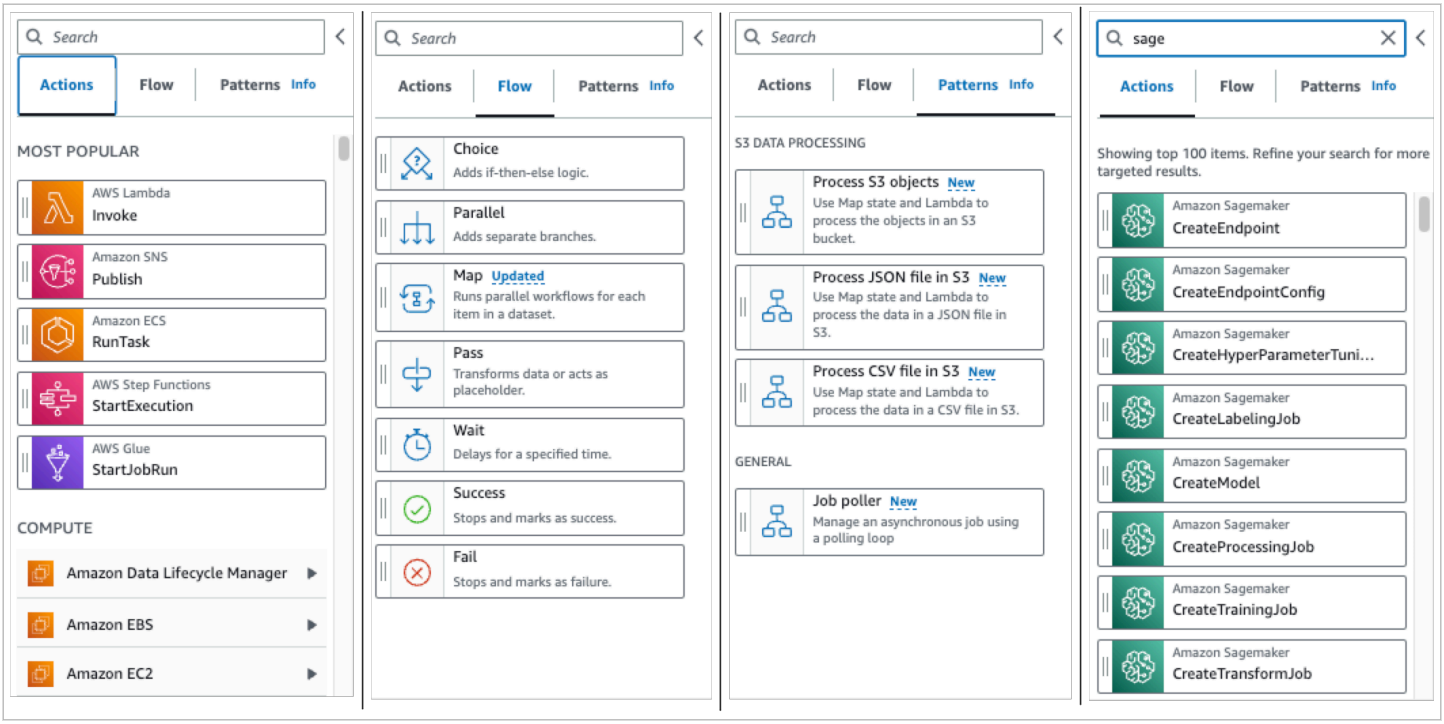
Workflow Studio 的设计模式提供了一个图形界面，可在您构建工作流原型时对其进行可视化。下图显示了设计模式中可用的不同组件。



1. 模式按钮 – 使用模式按钮在 Workflow Studio 的设计、代码和配置模式之间进行切换。如果工作流 ASL 定义中的 JSON 无效，则无法切换模式。
2. 状态浏览器包含以下三个选项卡：
   - “操作” 选项卡提供了 AWS API 列表，您可以将这些API拖放到画布中的工作流程图中。每个操作都代表一个 任务状态 状态。
   - 流选项卡提供了流状态列表，您可以将这些状态拖放到画布中的工作流图中。
   - P at terns 选项卡提供了几个 ready-to-use可重复使用的构造块，您可以将其用于各种用例。例如，您可以使用这些模式以迭代方式处理 Amazon S3 存储桶中的数据。
3. 您可以在画布上将状态拖放到工作流图中，更改状态的顺序，并选择要配置或查看的状态。
4. 在Inspector面板中，您可以查看和编辑画布上所选状态的属性。打开定义切换开关，可查看工作流的 Amazon States Language 代码，并突出显示当前选定的状态。
5. 需要帮助时，可以使用信息链接打开一个包含上下文信息的面板。这些面板还包括指向 Step Functions 文档中相关主题的链接。
6. 设计工具栏 – 包含一组用于执行常见操作的按钮，例如撤消、删除和放大。
7. 实用工具按钮 – 一组用于执行任务的按钮，例如保存工作流或将其 ASL 定义导出为 JSON 或 YAML 文件。

状态浏览器

在状态浏览器中，您可以选择要拖放到工作流图中的状态。“操作” 选项卡提供 AWS API 列表，“流程” 选项卡提供流程状态列表。而 “模式” 选项卡提供了几个 ready-to-use可重复使用的构建块，您可以将其用于各种用例。您可以使用顶部的搜索框在状态浏览器中搜索所有状态。



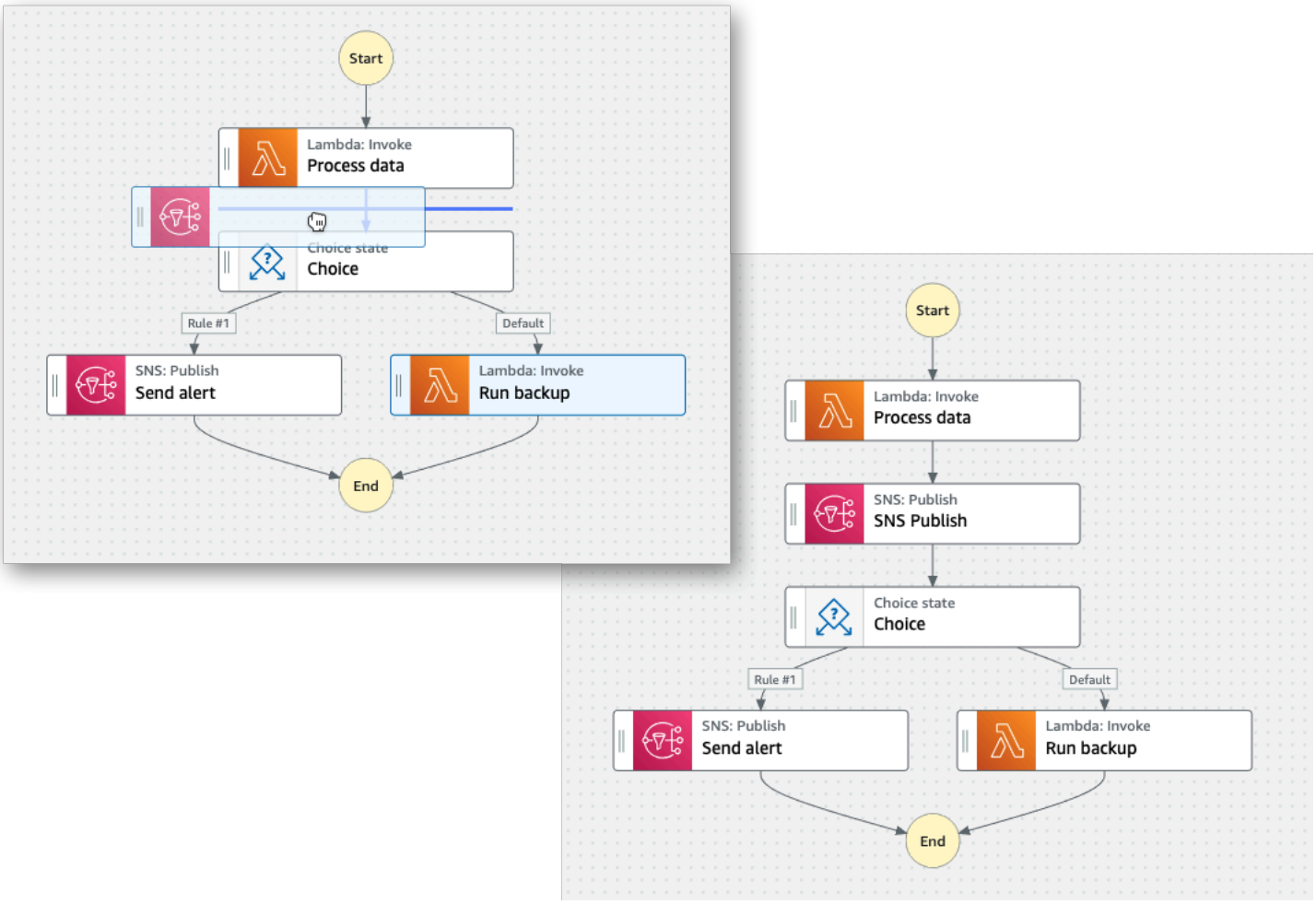
您可以使用七种流状态来指导和控制您的工作流。所有状态都会接收前一个状态的输入，其中多数还可以筛选前一个状态的输入，并输出到后面的状态。流状态包括：

- Choice：在工作流的执行分支之间添加选择。在 Inspector 的配置选项卡中，您可以配置规则，用于确定工作流将过渡到哪个状态。
- Parallel：为工作流添加并行执行分支。
- Map：针对输入数组每个元素的动态迭代步骤。与 `Parallel` 流状态不同，`Map` 状态将对状态输入中数组的多个条目执行相同的步骤。
- Pass: 允许将输入传递到输出。(可选) 您可以将固定数据添加到输出。
- Wait：让工作流暂停一段时间或直到指定的时间或日期。
- Succeed：成功后停止工作流。
- Fail：因故障而停止工作流。

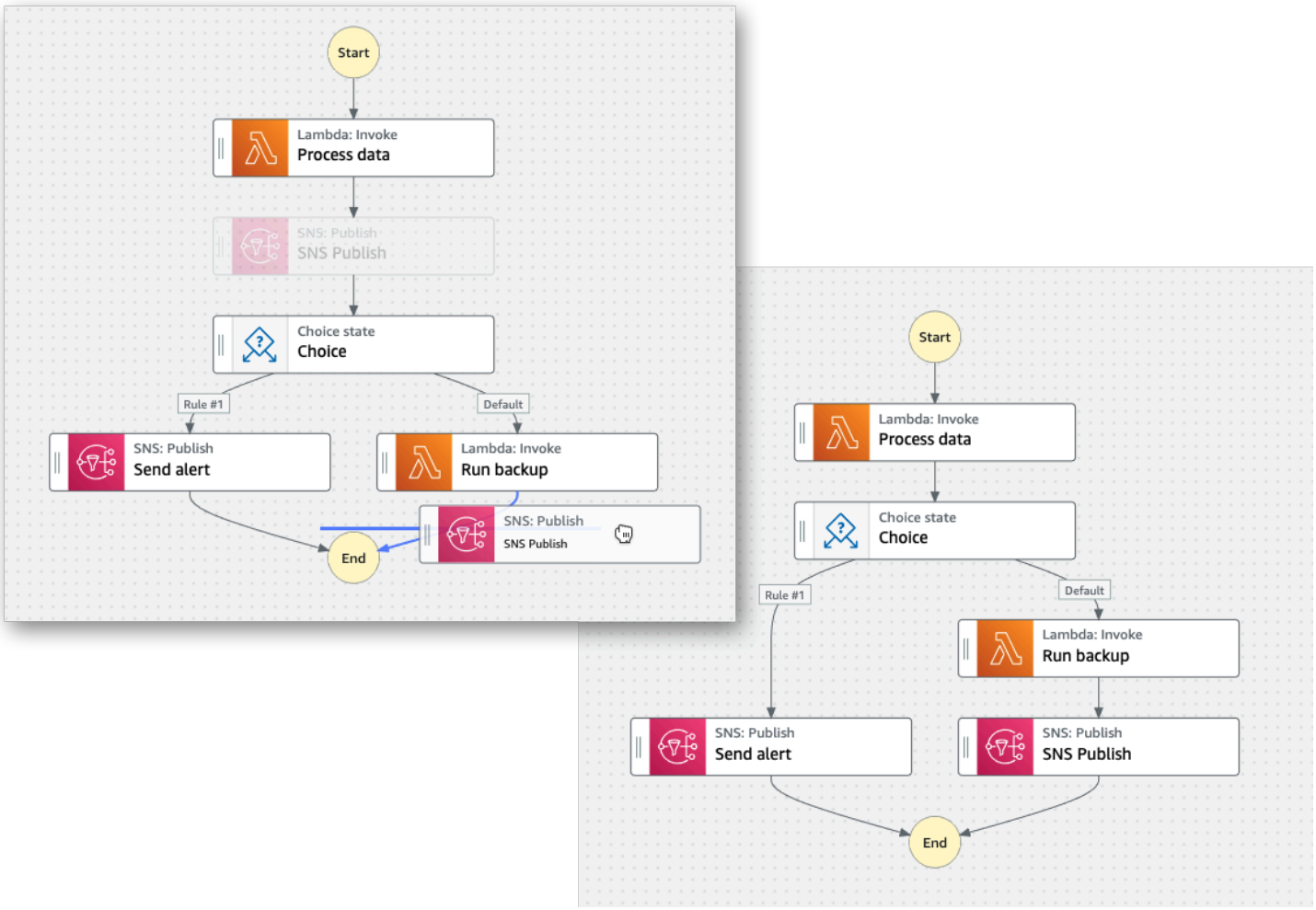
画布

选择要添加到工作流的状态后，将其拖到画布上，然后放到工作流图中。您也可以拖放状态，将其移动到工作流中的不同位置。如果您的工作流很复杂，您可能无法在画布面板上查看其所有内容。使用画布顶部的控件可以放大或缩小。要查看工作流图的不同部分，您可以在画布中拖动工作流图。

从操作或流选项卡中将状态拖放到工作流中。有一条线将显示其在您工作流中的放置位置。新的工作流状态已添加到工作流中，其代码会自动生成。

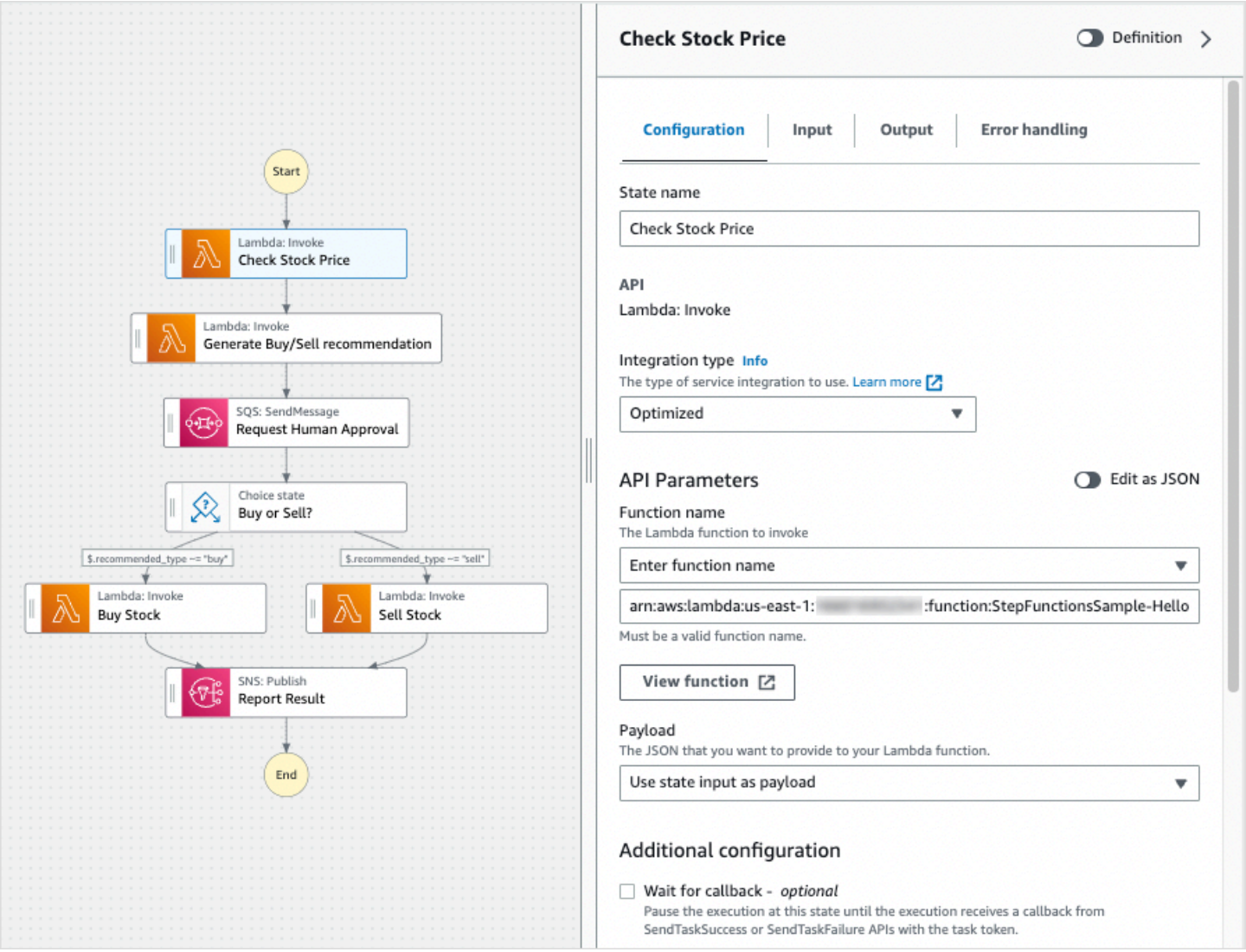


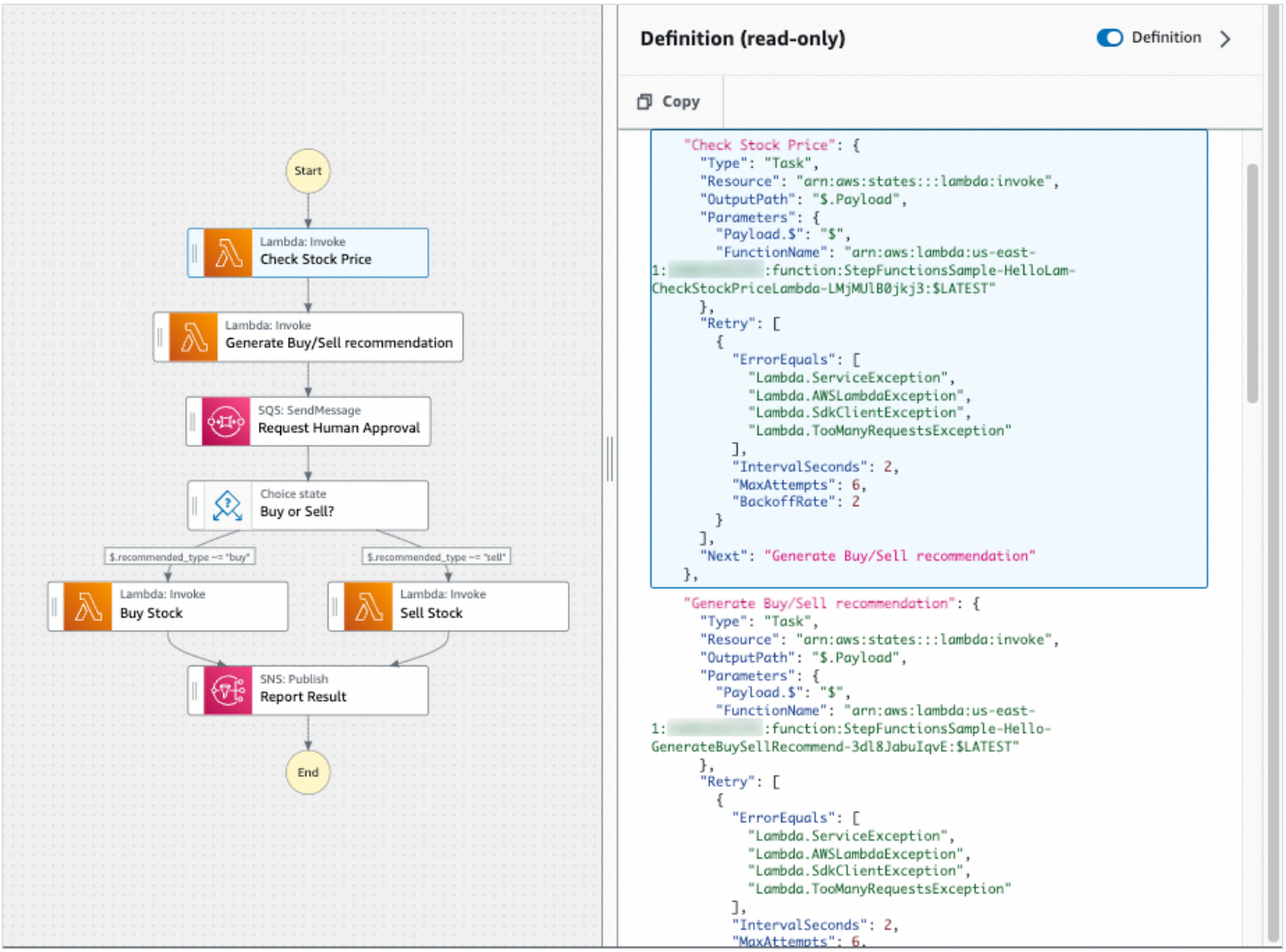
要更改状态的顺序，您可以将其拖动到工作流中的其他位置。



Inspector

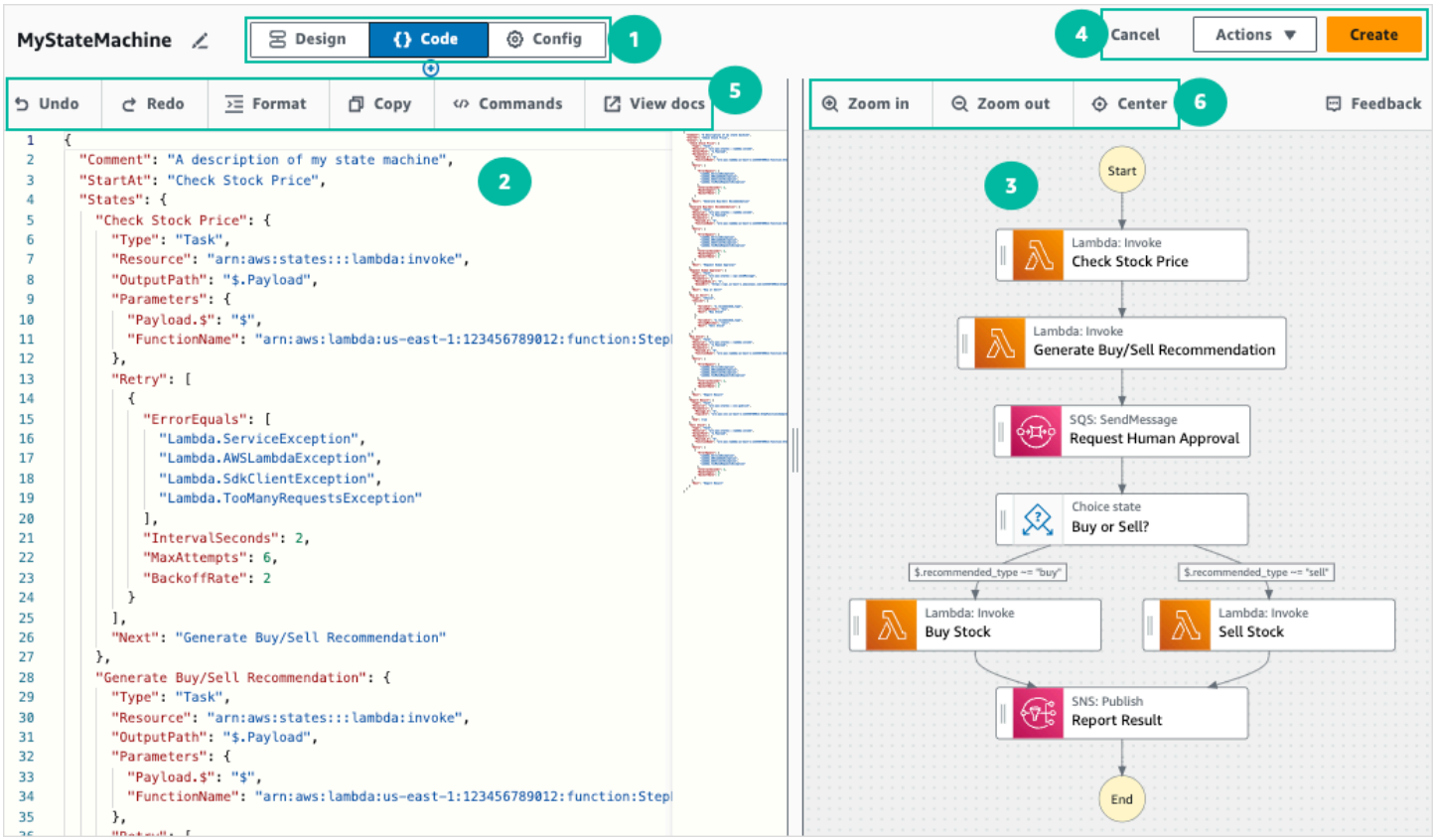
您可以配置添加到工作流中的任何状态。选择要配置的状态，即可在 Inspector 面板中看到其配置选项。要查看工作流代码自动生成的 ASL 定义，请打开定义切换开关。与您选择的状态关联的 ASL 定义将突出显示。


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.



代码模式

Workflow Studio 的代码模式提供了一个集成的代码编辑器，用于在 Step Functions 控制台中查看、编写和编辑工作流的 Amazon States Language (ASL) 定义。下图显示了代码模式下可用的不同组件。



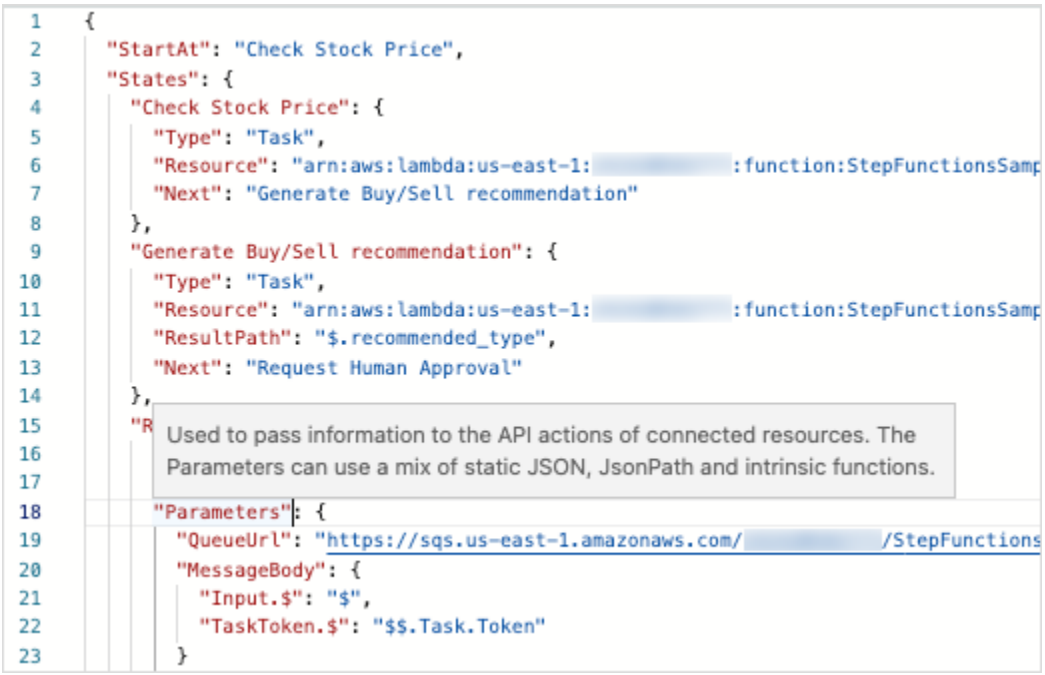
1. 模式按钮 – 使用模式按钮在 Workflow Studio 的设计、代码和配置模式之间进行切换。如果工作流 ASL 定义中的 JSON 无效，则无法切换模式。
2. 代码编辑器是在 Workflow Studio 中编写和编辑工作流的 ASL 定义的地方。代码编辑器还提供语法突出显示和自动完成等特征。
3. 图表可视化窗格 – 显示工作流的实时图形可视化。
4. 实用工具按钮 – 一组用于执行任务的按钮，例如保存工作流或将其 ASL 定义导出为 JSON 或 YAML 文件。
5. 代码工具栏 – 包含一组用于执行常见操作的按钮，例如撤消操作或格式化代码。
6. 图表工具栏 – 包含一组用于执行常见操作的按钮，例如放大和缩小工作流图表。

代码编辑器

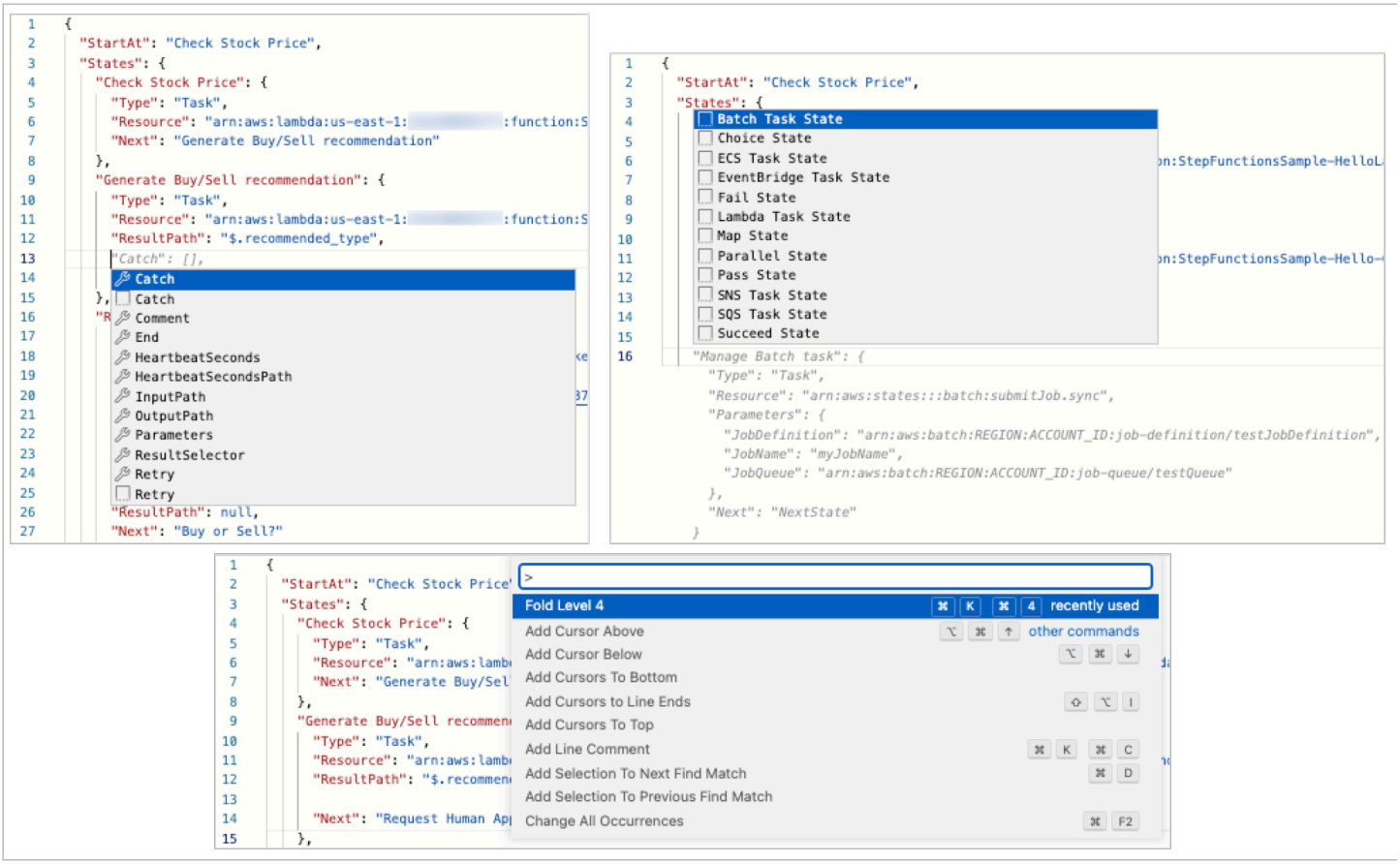
代码编辑器提供了类似 IDE 的体验，让您可以在 Workflow Studio 中使用 JSON 编写和编辑工作流定义。代码编辑器包括多项特征，例如语法突出显示、自动完成建议、ASL 定义验证和上下文相关帮助显示。更新工作流定义时，图表可视化窗格会呈现工作流的实时图表。您还可以在设计模式中查看更新的工作流图。

如果在设计模式或图表可视化窗格中选择一个状态，则该状态的 ASL 定义将在代码编辑器中突出显示。如果在设计模式或图表可视化窗格中重新排序、删除或添加状态，则工作流的 ASL 定义会自动更新。

将光标放在工作流定义中的任何字段上，即可以工具提示的形式查看其上下文相关帮助。



自动完成建议会显示可包含在工作流中的字段或状态的代码段。要查看可在特定状态下包含的字段列表，请按 **Ctrl+Space**。要为工作流中的新状态生成代码段，请在当前状态定义后按 **Ctrl+Space**。您也可以按 **F1** 键显示可用命令列表。



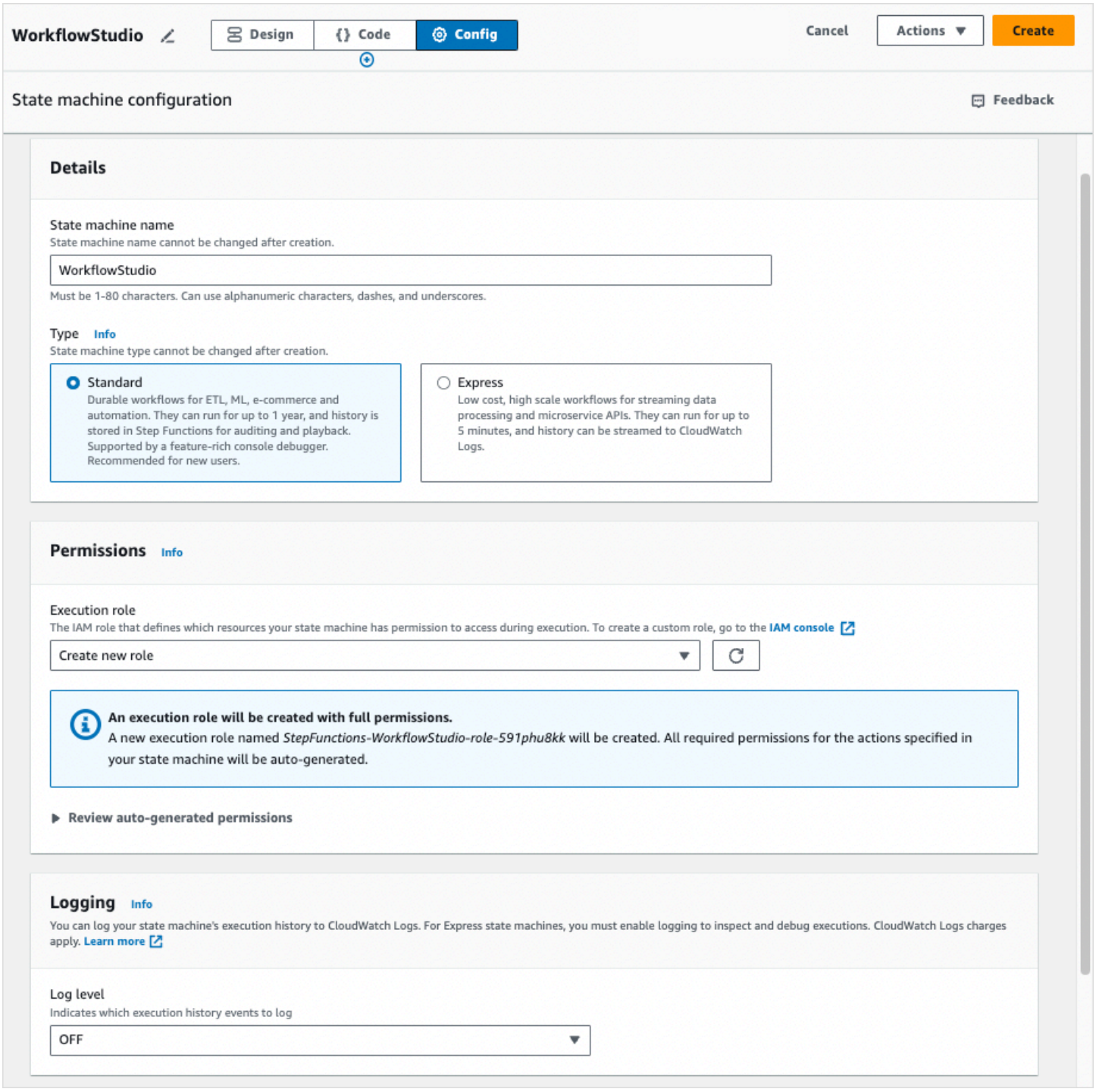
图表可视化窗格

图形可视化可让您以图形格式查看工作流的外观。当在 Workflow Studio 的代码编辑器中编写工作流定义时，图表可视化窗格会呈现工作流的实时图表。当在图表可视化窗格中重新排序、删除或复制状态时，代码编辑器中的工作流定义会自动更新。同样，当您在代码编辑器中更新工作流定义、重新排序、删除或添加状态时，可视化窗格也会自动更新。

如果工作流 ASL 定义中的 JSON 无效，则图形可视化窗格会暂停渲染并在窗格底部显示一条状态消息。

配置模式

Workflow Studio 的配置模式可用于管理状态机的配置。在此模式下，您可以指定详细信息，例如状态机名称及其类型、IAM 权限以及状态机的日志配置。您可以在此模式下指定的其他配置包括在创建状态机时启用 AWS X-Ray 跟踪和发布版本。创建状态机后，您可以编辑所有状态机配置选项，但无法编辑状态机名称和类型。下图展示了您可以在配置模式下可以指定的一些配置。



管理状态机配置

要管理状态机配置，请执行以下操作：

1. 在状态机名称框中输入状态机的名称。

> **Tip**
>
> 或者，选择默认状态机名称旁边的编辑图标MyStateMachine。然后，在状态机配置下指定一个名称。

> **Important**
>
> 创建状态机后，将无法编辑状态机名称。

2. 在类型中，选择标准或快速状态机类型。有关状态机的更多信息，请参阅标准和快速工作流。

> **Important**
>
> 创建状态机后，将无法编辑状态机类型。

3. 在权限中，选择将用作状态机执行角色的 IAM 角色。

- 创建新角色（推荐）：如果选择此选项，Step Functions 会在创建状态机时，自动为状态机创建一个执行角色，且具有所需的最低权限。这些自动生成的 IAM 角色对 AWS 区域 您在其中创建状态机的角色有效。

  > **Tip**
  >
  > 要查看 Step Functions 为状态机自动生成的权限，请选择查看自动生成的权限。

  > **Note**
  >
  > 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

- 选择现有角色：为状态机创建自己的 IAM 角色，然后从选择现有角色下列出的选项中选择该角色。确保该角色的策略包含您希望状态机承担的权限。

  有关如何创建 IAM 策略 的信息，请参阅 IAM 用户指南中的创建 IAM 策略。

- 输入角色 ARN：指定将用于此状态机的现有 IAM 角色的 Amazon 资源名称 (ARN)。例如，`arn:aws:iam::123456789012:role/service-role/StepFunctions-WorkflowStudio-role-777f4027`。

4. 在日志记录中，设置状态机的日志级别。Step Functions 会根据您的选择记录执行历史事件。您可以选择下列选项之一：
   - 全部：记录所有事件类型。
   - 错误：记录所有错误事件类型，例如 TaskFailed 和 ExecutionFailed。
   - 致命：记录所有致命错误事件类型，例如 ExecutionAborted 和 ExecutionFailed。
   - 关：不记录任何事件类型。

   有关日志级别的更多信息，请参阅日志级别。
5. 在其他配置中，设置以下一项或多项可选配置：
   - 启用 X-Ray 跟踪：为 Step Functions 选择此复选框，即使上游服务未传递跟踪 ID，也可将状态机执行的跟踪发送到 X-Ray。有关更多信息，请参阅 AWS X-Ray 和 Step Functions。
   - 创建时发布版本：版本是您可以运行的状态机快照，带编号且不可变。选中此复选框可在创建状态机时发布状态机的版本。Step Functions 将版本 1 作为状态机的第一个修订版发布。

     有关版本的更多信息，请参阅状态机版本。
   - 添加新标签：选择此框可为状态机添加标签。添加标签可帮助您跟踪和管理与资源关联的成本，并增强 IAM 策略中的安全性。有关标签的更多信息，请参阅 Step Functions 中的标记。
6. 选择创建。
7. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色配置，返回配置 模式。

键盘快捷键

Workflow Studio 支持以下键盘快捷键：

键盘快捷键	功能
Shortcuts for the Code mode	
Ctrl+space	Auto-complete suggestions

键盘快捷键	功能
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

使用 Workflow Studio

学习使用 Step Functions Workflow Studio 创建、编辑和运行工作流。工作流准备就绪后，您可以将其导出。您也可以使用 Workflow Studio 进行快速设计原型。

本主题内容

- 创建工作流
- 设计工作流
- 运行工作流
- 编辑工作流
- 导出工作流
- 创建工作流原型

创建工作流

在 Workflow Studio 中，您可以选择初学者模板，也可以选择空白模板从头开始创建工作流。对于空白模板，您可以使用设计或代码模式来创建工作流。

入门模板是一个 ready-to-run 示例项目，它会自动创建工作流程 proptotype 和定义，并将项目所需的所有相关 AWS 资源部署到您的。 AWS 账户您可以使用这些初学者模板按原样部署和运行，也可以使用工作流原型在其基础上进行构建。有关初学者模板的更多信息，请参阅 Step Functions 的示例项目。

使用初学者模板创建工作流

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在选择模板对话框中，执行以下任一操作来选择示例项目，例如任务计时器示例项目：
   - 在“按关键字搜索”框中键入 **`Task Timer`**，然后从返回的搜索结果中选择任务计时器。
   - 浏览右侧窗格中全部下列出的示例项目，然后选择任务计时器。
3. 选择下一步继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。
5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：
   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。在设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，更新工作流的 Amazon States Language (ASL) 定义。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了 “运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

     > **Tip**
     >
     > 要查看示例项目的状态机定义，请选择代码。

   准备就绪后，选择部署并运行以部署示例项目并创建资源。

> **Note**
>
> 创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

> **Important**
>
> CloudFormation 模板中使用的每项服务均按标准收费。

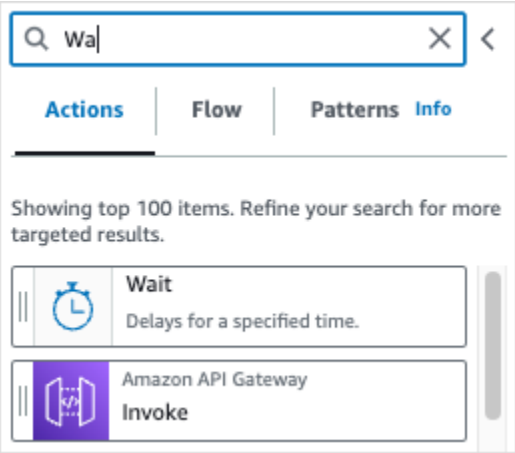
使用空白模板创建工作流

1. 打开 Step Functions 控制台。
2. 选择创建状态机。
3. 在 选择模板对话框中，选择空白。
4. 选择选择。这将在设计模式中打开 Workflow Studio。

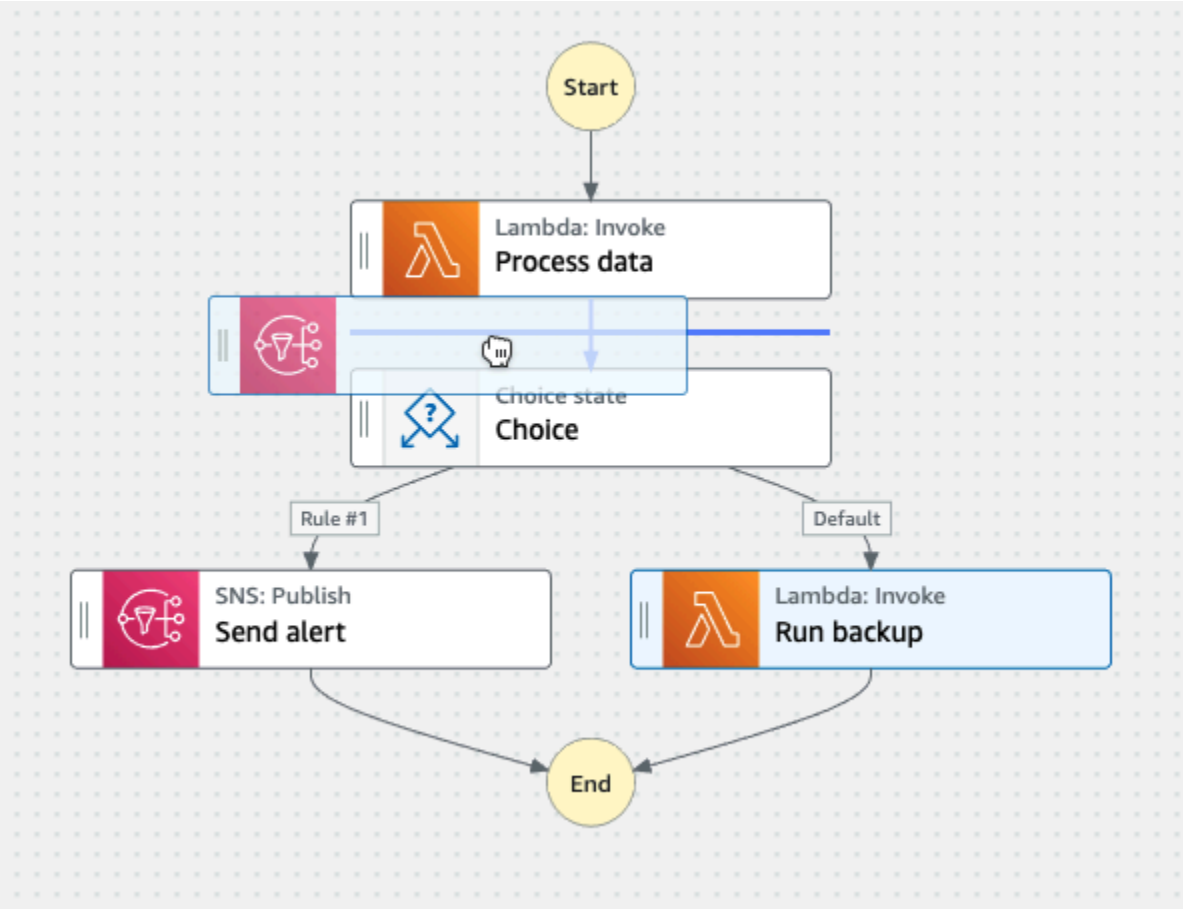
   现在，您可以开始在设计模式下设计工作流，或在代码模式下编写工作流定义。
5. 选择配置可在配置模式下管理工作流程的配置。例如，为工作流提供名称并选择其类型。

设计工作流

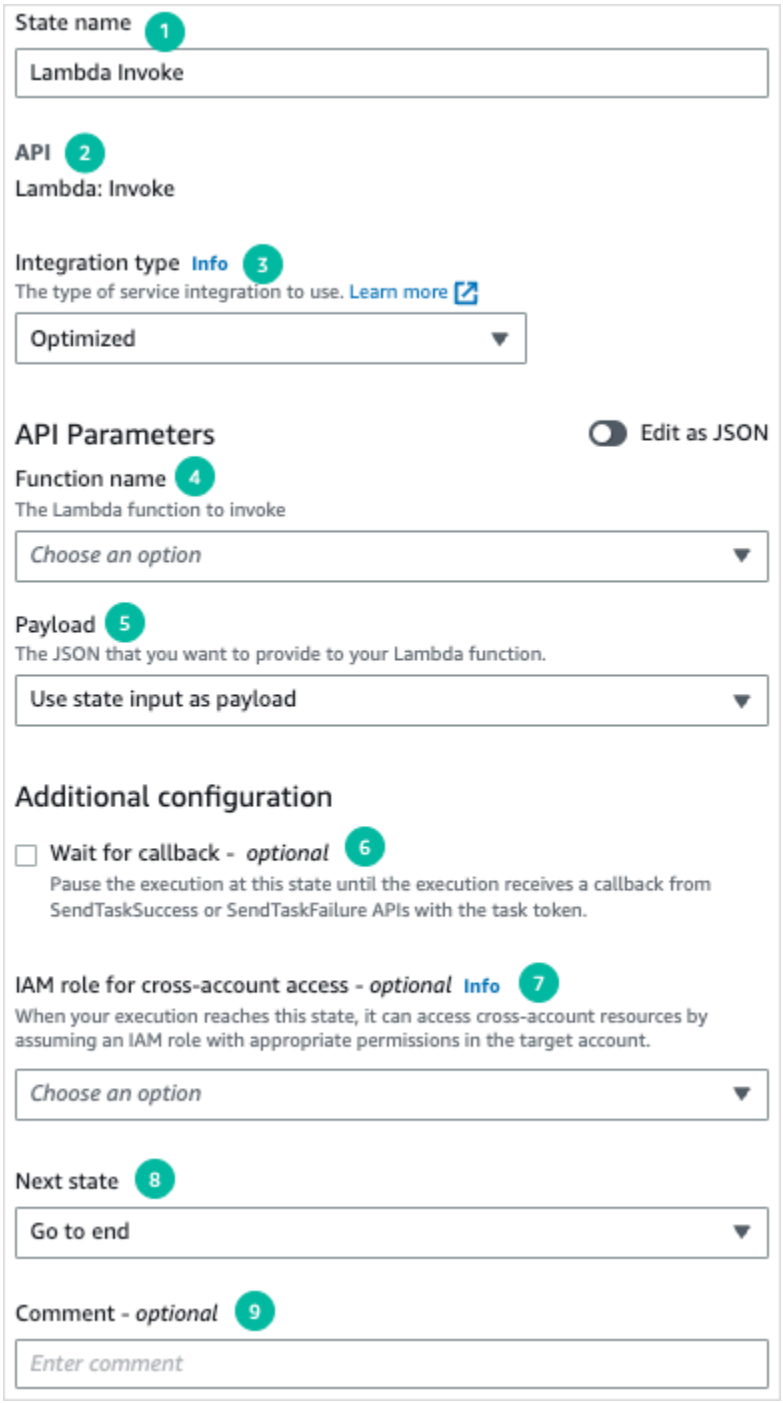
如果您知道要添加的状态的名称，请使用顶部的搜索框，在设计模式的操作和状态浏览器流选项卡中查找该状态。



如果不知道名称，请从状态浏览器中选择一个状态，然后将其拖放到画布上，放置在工作流中所需的位置。您还可以通过将状态拖到工作流中的其他位置，从而对工作流中的状态进行重新排序。将状态拖到画布上时，工作流中可以放置状态的位置会出现一条线。将状态拖放到画布上后，其代码会自动生成并添加到您的工作流定义中。要查看定义，请打开 Inspector 面板上的定义切换开关。要编辑工作流定义，请选择提供集成代码编辑器的代码模式。



将状态拖放到画布上后，可以在右侧的 Inspector 面板中对其进行配置。此面板包含您在画布上放置的每个状态或 API 操作的配置、输入、输出和错误处理选项卡。您可以在配置选项卡中配置工作流中包含的状态。例如，Lambda Invoke API 操作的配置选项卡包含以下选项：

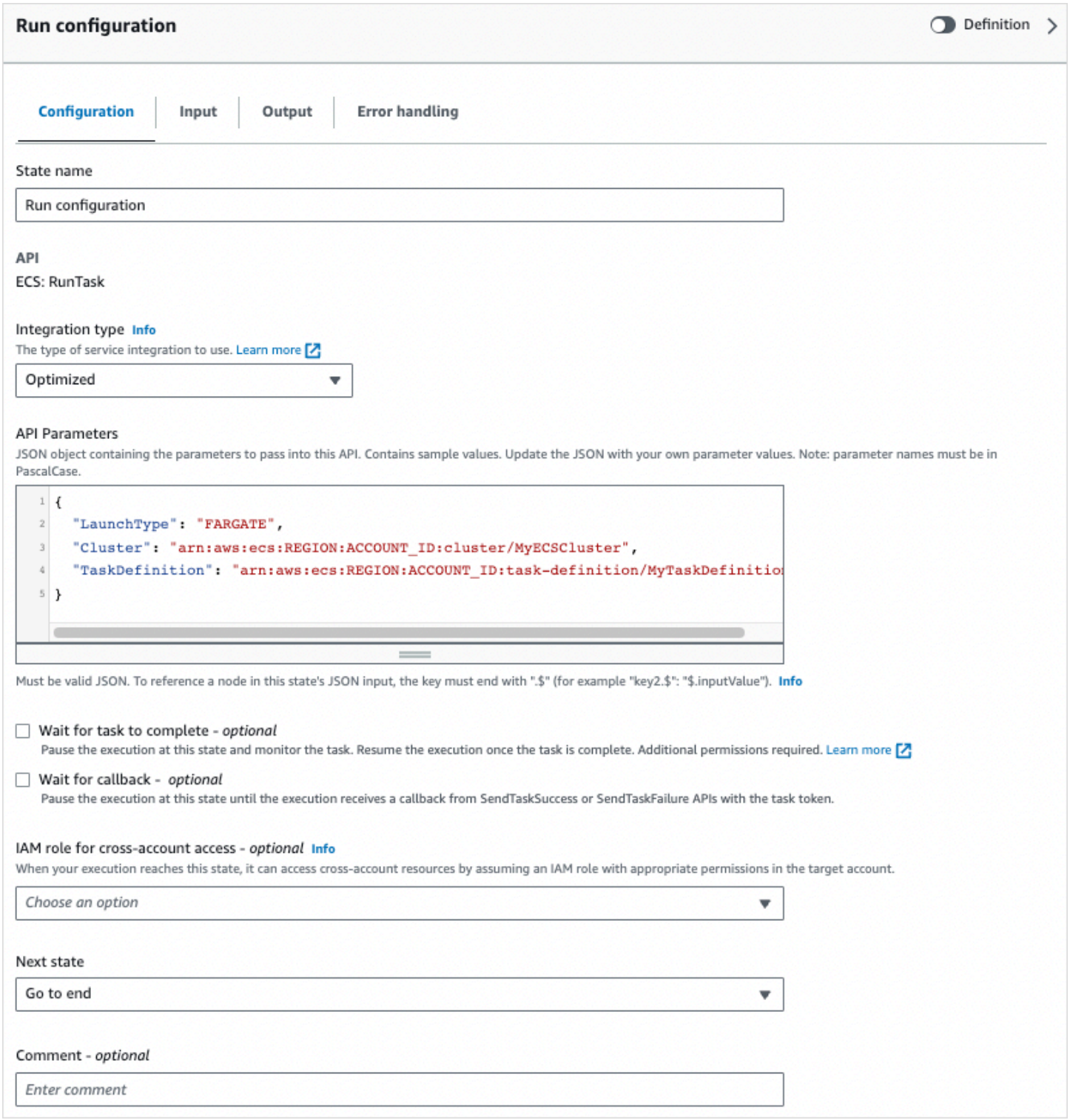


1. 状态名称用于标识状态。您可以自定义名称，也可以接受默认生成的名称。
2. API 显示状态使用的 API 操作。
3. 集成类型下拉列表提供用于在 Step Functions 中选择可用服务集成类型的选项。您选择的集成类型用于调用工作流程 AWS 服务 中特定的 API 操作。
4. 函数名称提供了以下选项：

- 输入函数名称：您可以输入函数名称或其 ARN。
- 运行时从状态输入中获取函数名称：您可以使用此选项根据指定的路径从状态输入中动态获取函数名称。
- 选择函数名称：您可以直接从您的账户和区域的可用函数中进行选择。

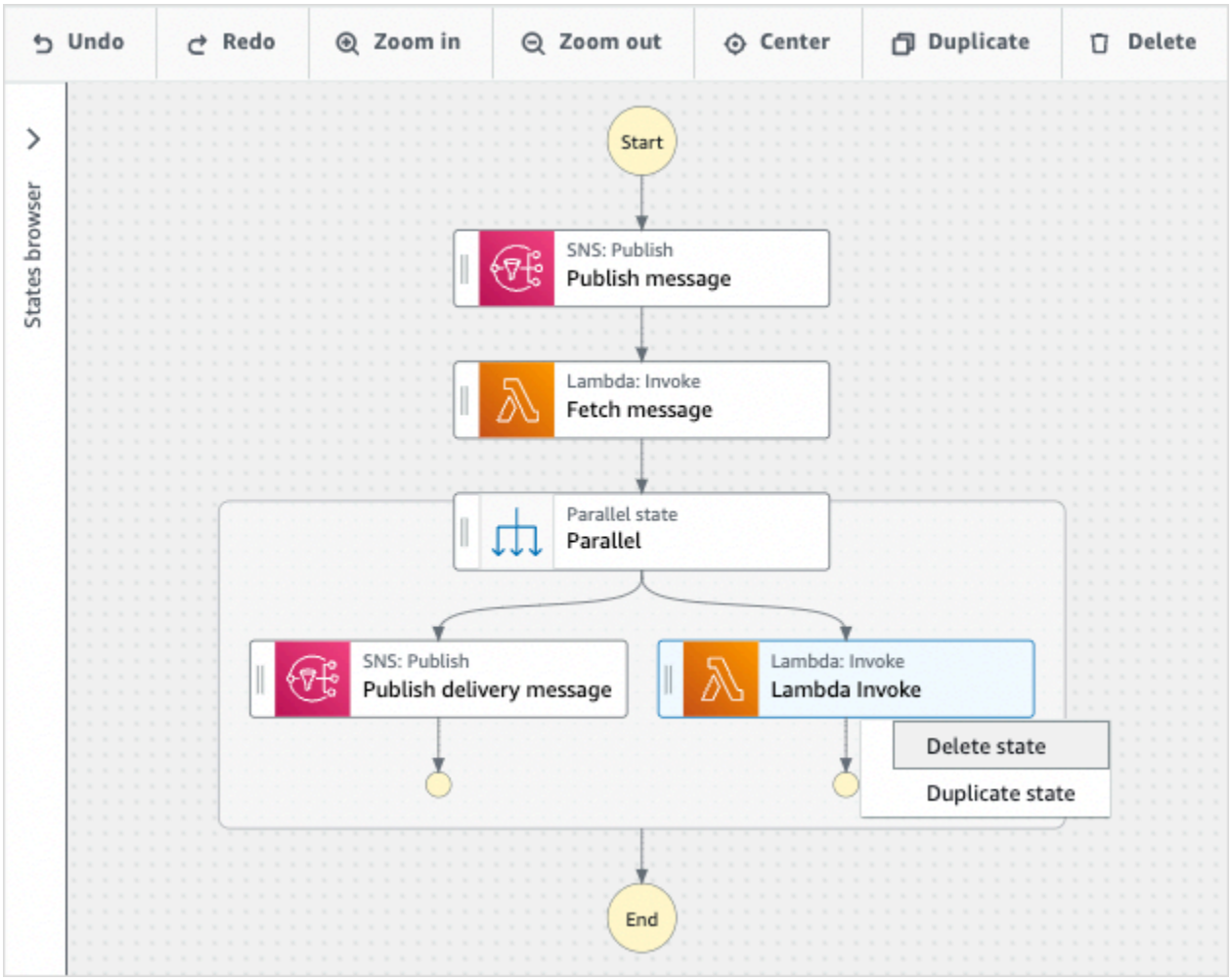
5. 有效负载允许您从以下选项中进行选择：
   - 使用状态输入作为有效负载：您可以使用此选项将状态输入作为有效负载传递给 Lambda 函数。
   - 输入自己的有效负载：您可以使用此选项构造一个 JSON 对象，作为有效负载传递给 Lambda 函数。此 JSON 可以包含静态值和从状态输入中选择的值。
   - 无有效负载：如果不想向 Lambda 函数传递任何有效负载，则可以使用此选项。
6. （可选）某些状态可以选择等待任务完成或等待回调。如果可用，这些选项可选择以下服务集成模式之一：
   - 未选择任何选项：Step Functions 将使用请求响应集成模式。Step Functions 将等待 HTTP 响应，然后进入下一个状态。Step Functions 不会等待作业完成。当没有可用选项时，状态将使用此模式。
   - 等待任务完成：Step Functions 将使用运行作业 (.sync) 集成模式。
   - 等待回调：Step Functions 将使用等待具有任务令牌的回调集成模式。
7. （可选）为了访问工作流程 AWS 账户 中不同配置的资源，Step Functions 提供跨账户访问权限。用于跨账户存取的 IAM 角色提供以下选项：
   - 提供 IAM 角色 ARN：指定包含相应资源访问权限的 IAM 角色。这些资源可在目标账户中使用，您可以 AWS 账户 向该账户进行跨账户调用。
   - 在运行时从状态输入中获取 IAM 角色 ARN：在包含 IAM 角色的状态 JSON 输入中指定一个指向现有键值对的引用路径。
8. 下一个状态用于选择下一个要过渡的状态。
9. （可选）注释字段可用于添加自己的注释。它不会影响工作流，但可以用来为您的工作流添加注释。

有些状态会有更通用的配置选项。例如，Amazon ECS `RunTask` 状态配置包含一个填充了占位符值的 `API Parameters` 字段。



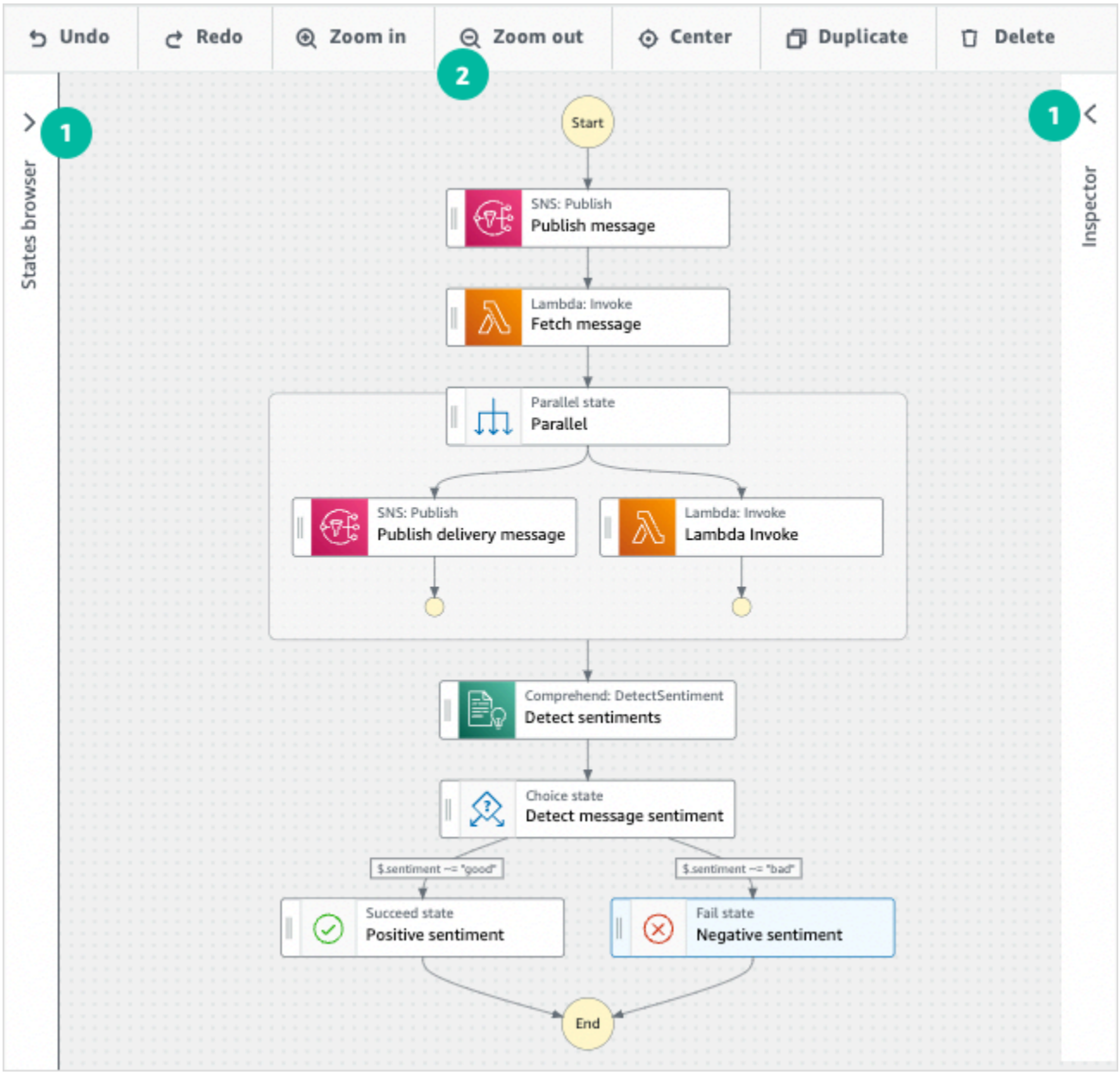
对于此类状态，您可以将占位符值替换为适合您需求的配置。

要删除状态，您可以使用退格键，右键单击并选择删除状态，或者在设计工具栏上选择删除。



随着工作流的发展，其可能不适合画布。您可以：

1. 使用侧面板上的控件调整面板大小或关闭面板。
2. 使用画布顶部设计工具栏控件放大或缩小工作流图表。



运行工作流

使用 Workflow Studio 创建或编辑工作流后，您可以在 Step Functions 控制台中运行该工作流并查看其执行情况。

在 Workflow Studio 中运行工作流

1. 在设计、代码或配置模式下，选择执行。

   开始执行对话框将在新选项卡中打开。
2. 在启动执行对话框中，执行以下操作：

1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。
3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

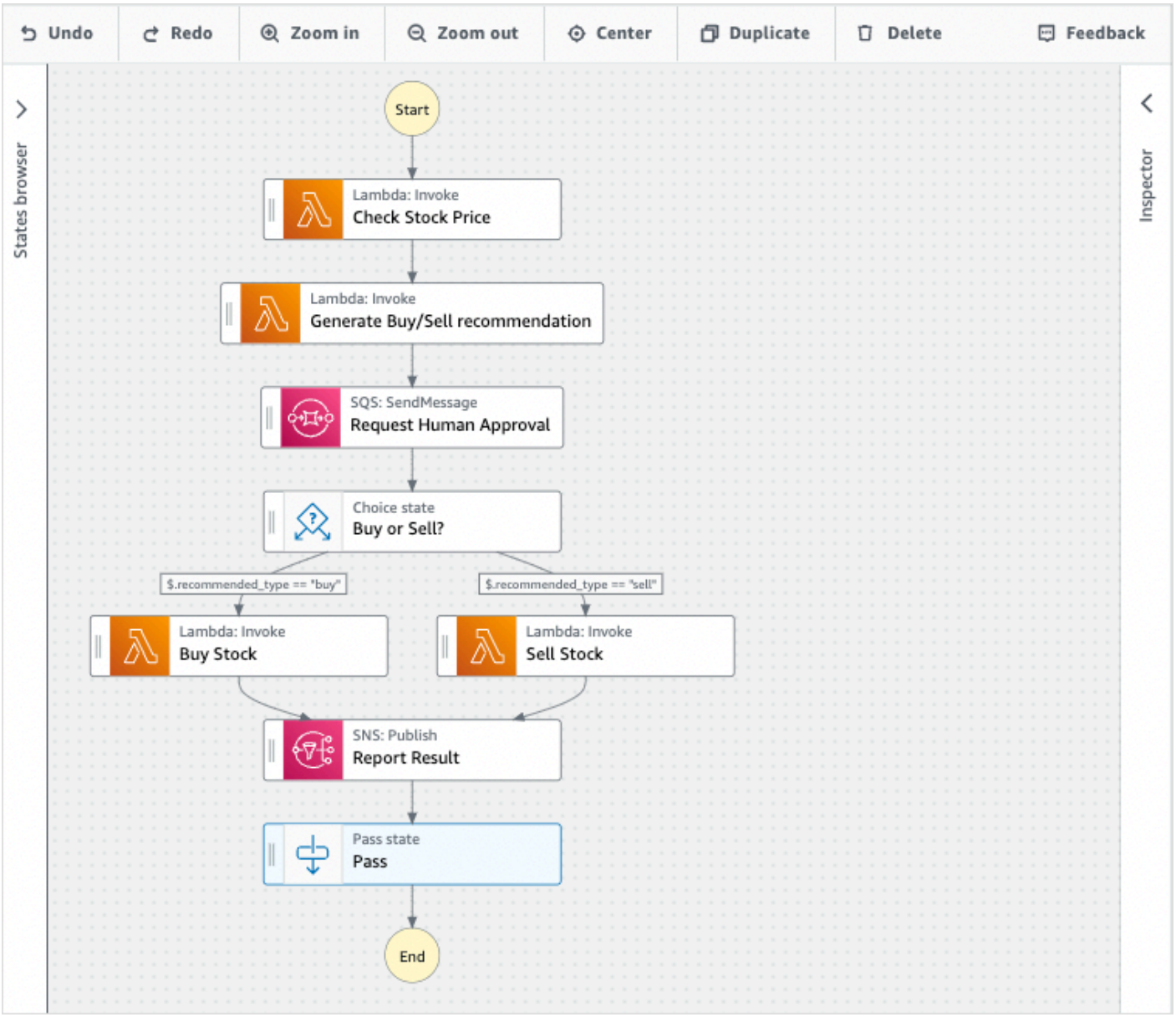
   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

编辑工作流

您可以在 Workflow Studio 的设计模式中直观地编辑现有工作流。您也可以在 Workflow Studio 的代码模式中编辑工作流定义。

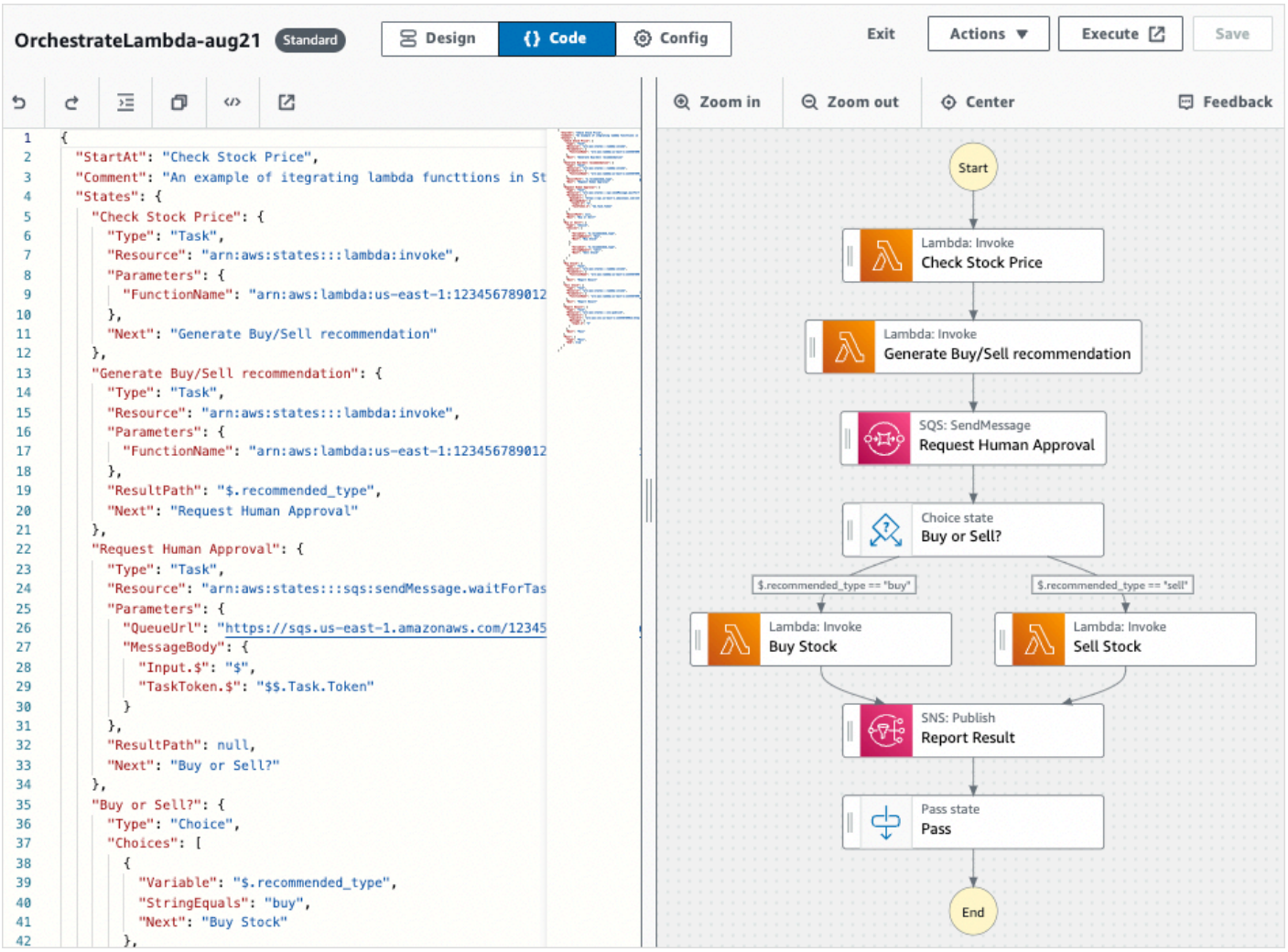
要编辑现有工作流，请执行以下操作：

1. 打开 Step Functions 控制台。
2. 在状态机页面上，选择要编辑的工作流。
3. 在状态机详细信息页面上，选择编辑。
4. 工作流程在 Workflow Studio 的设计模式下打开。根据需要编辑工作流。



> **Note**
>
> 如果您在工作流中发现错误，则必须在设计模式下进行修复。如果工作流中存在任何错误，则无法切换到代码或配置模式。

5. （可选）选择代码按钮，在 Workflow Studio 中查看或编辑工作流定义。



6. 操作完成后，选择保存，保存更新后的工作流。
7. （可选）要运行更新后的工作流，请选择执行。开始执行对话框将在新选项卡中打开。

导出工作流

您可以导出工作流程 Amazon States Language (ASL) 的定义和工作流图：

1. 在 Step Functions 控制台中选择您的工作流。
2. 在状态机详细信息页面上，选择编辑。
3. （可选）您的工作流在 Workflow Studio 的设计模式下打开。在设计模式下编辑工作流或切换到代码模式。
4. 选择操作下拉按钮，然后执行以下一项或两项操作：
   - 要将工作流图表导出到 SVG 或 PNG 文件，请在导出图表下选择所需的格式。

- 要将工作流定义导出为 JSON 或 YAML 文件，请在导出定义下选择所需的格式。

创建工作流原型

您可以使用 Workflow Studio 创建包含占位符资源的新工作流原型。您也可以使用 Application Composer 中的 Workflow Studio 来构建工作流。要创建原型，请执行以下操作：

1. 登录 Step Functions 控制台。
2. 选择创建状态机。
3. 在 选择模板对话框中，选择空白。
4. 选择选择。这将在设计模式中打开 Workflow Studio。
5. 打开 Workflow Studio 的设计模式。在 Workflow Studio 中设计您的工作流。要包含占位符资源，请执行以下操作：

   a. 选择要为其添加占位符资源的状态，然后在配置中：

      - 对于 Lambda 调用状态，选择函数名称，然后选择输入函数名称。您也可以输入函数的自定义名称。
      - 对于“Amazon SQS 发送消息”状态，选择队列 URL，然后选择输入队列 URL。输入占位符队列 URL。
      - 对于“Amazon SNS 发布”状态，从主题中选择一个主题 ARN。
      - 对于操作下列出的所有其他状态，您可以使用默认配置。

      > **Note**
      >
      > 如果您在工作流中发现错误，则必须在设计模式下进行修复。如果工作流中存在任何错误，则无法切换到代码或配置模式。

   b. （可选）要查看自动生成的工作流 ASL 定义，请选择定义。

   c. （可选）要在 Workflow Studio 中更新工作流定义，请选择代码按钮。

      > **Note**
      >
      > 如果您在工作流定义中看到错误，则必须在代码模式下对其进行修复。如果工作流定义中存在任何错误，则无法切换到设计或配置模式。

6. （可选）要编辑状态机名称，请选择默认状态机名称旁边的编辑图标，MyStateMachine然后在状态机名称框中指定名称。

   您也可以切换到配置模式编辑默认状态机名称。

7. 指定工作流设置，例如状态机类型及其执行角色。

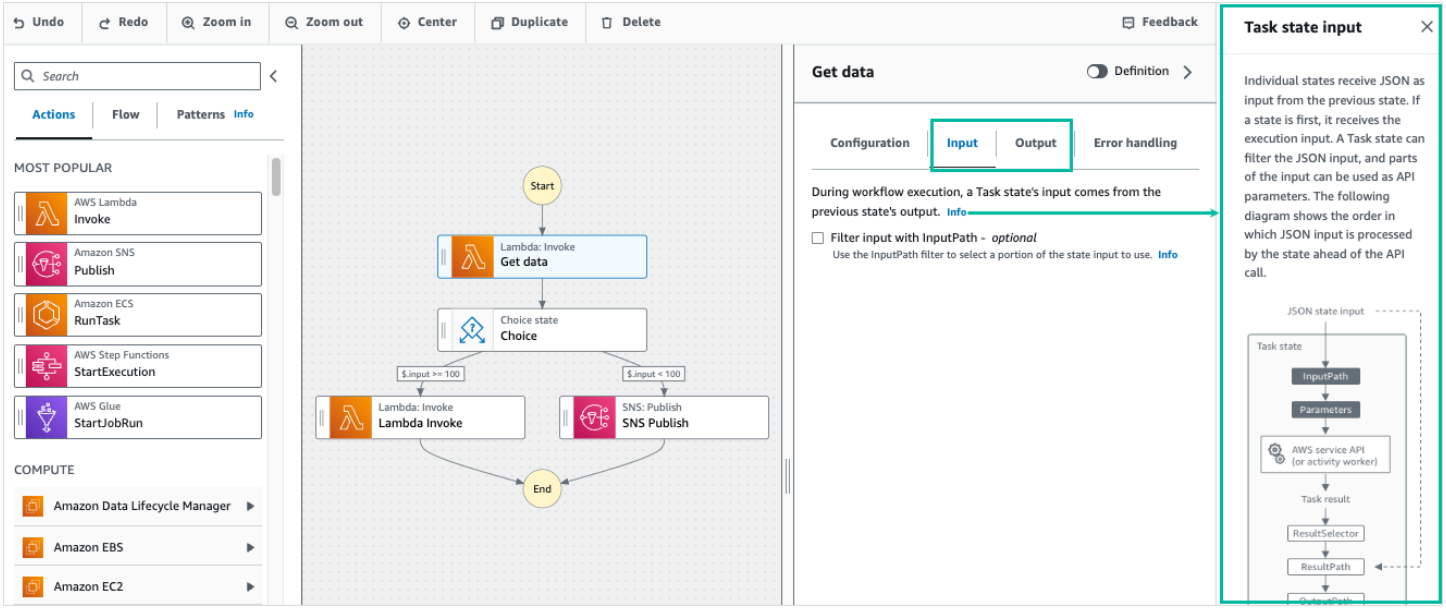
8. 选择 创建。

现在，您已经创建了一个新的工作流，其中包含可用于原型的占位符资源。您可以导出工作流定义和工作流图表。

- 要将工作流定义导出为 JSON 或 YAML 文件，请在设计或代码模式下，选择操作下拉按钮。然后，在导出定义下，选择要导出的格式。您可以使用此导出的定义作为使用 AWS Toolkit for Visual Studio Code 进行本地开发的起点。
- 要将工作流图表导出为 SVG 或 PNG 文件，请在设计或代码模式下，选择操作下拉按钮。然后，在导出定义下，选择所需的格式。

为状态配置输入和输出

每个状态都根据接收的输入做出决定或执行操作。在大多数情况下，该状态会将输出传递到其他状态。在 Workflow Studio 中，可以在 Inspector 面板的输入和输出选项卡中配置状态如何筛选和操作其输入和输出数据。配置输入和输出时，使用信息链接访问上下文帮助。



有关 Step Functions 如何处理输入和输入的详细信息，请参阅 Step Functions 中的输入和输出处理。

配置状态的输入

每个状态都以 JSON 形式接收来自前一个状态的输入。如果要筛选输入，可以使用 Inspector 面板中输入选项卡下的 `InputPath` 筛选条件。`InputPath` 是一个以 `$` 开头的字符串，用于标识特定 JSON 节点。它们被称为引用路径，它们遵循 JsonPath 语法。

要筛选输入，请进行以下操作：

- 选择 “筛选输入” InputPath。
- JsonPath为`InputPath`筛选器输入有效的值。例如，**`$.data`**。

`InputPath` 筛选条件将添加到您的工作流中。

Example 示例 1：在工作流工作室中使用 InputPath 过滤器

假设状态的输入包含以下 JSON 数据。

```
{
  "comment": "Example for InputPath",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

要应用InputPath过滤器，请选择 “筛选输入” InputPath，然后输入相应的参考路径。如果您输入 **`$.dataset2.val1`**，则以下 JSON 将作为输入传递给状态。

```
{"a"}
```

参考路径也可以选择值。如果您引用的数据是 `{ "a": [1, 2, 3, 4] }`，并且您应用引用路径 `$.a[0:2]` 作为 `InputPath` 筛选条件，则结果如下。

```
[ 1, 2 ]
```

Parallel、Map 和 Pass 流状态在其输入选项卡下还有一个名为 `Parameters` 的输入筛选选项。此过滤器在过滤 InputPath 器之后生效，可用于构造由一个或多个键值对组成的自定义 JSON 对象。每对的值可以是静态值，可从输入中选择，也可通过路径从Context 对象中选择。

> **Note**
>
> 要指定参数使用引用路径指向输入中的 JSON 节点，请使用 `.$` 作为参数名称的结尾。

Example 示例 2：为 Parallel 状态创建自定义 JSON 输入

假设以下 JSON 数据是一个 Parallel 状态的输入。

```
{
  "comment": "Example for Parameters",
  "product": {
    "details": {
      "color": "blue",
      "size": "small",
      "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

要选择该输入的一部分并传递带有静态值的附加键值对，可以在 Parallel 状态的输入选项卡下的参数字段中指定以下内容。

```
{
 "comment": "Selecting what I care about.",
 "MyDetails": {
    "size.$": "$.product.details.size",
    "exists.$": "$.product.availability",
    "StaticValue": "foo"
    }
 }
```

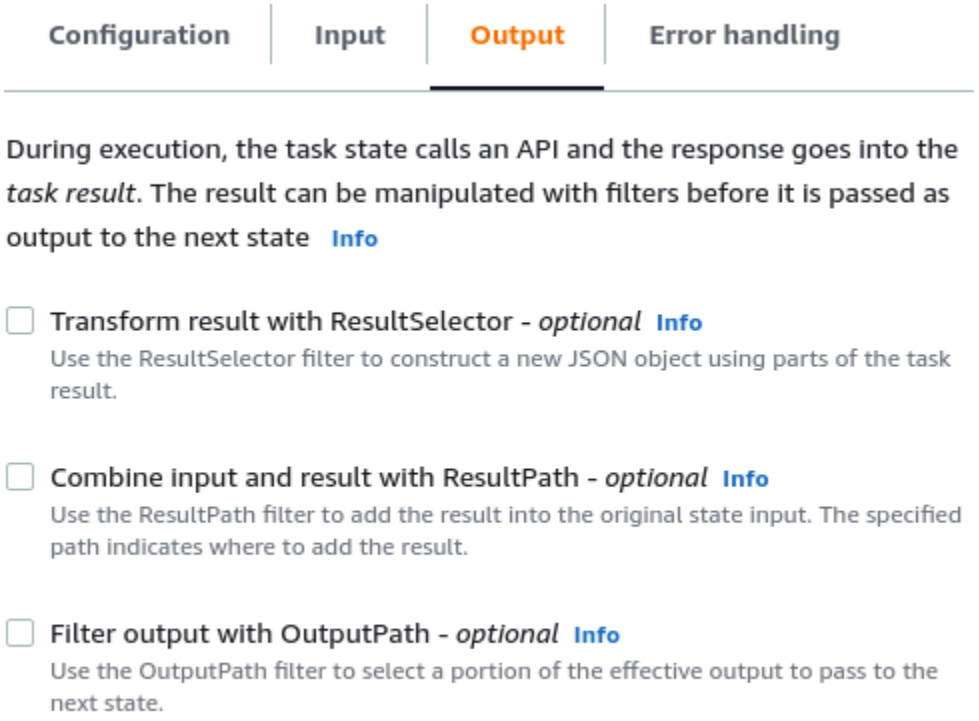
结果将是以下 JSON 数据。

```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
    "size": "small",
    "exists": "in stock",
    "StaticValue": "foo"
  }
}
```

配置状态的输出

每个状态都会生成 JSON 输出，可以在将其传递到下一个状态之前对其进行筛选。有几个筛选条件可用，每个筛选条件对输出的影响都不一样。每种状态可用的输出筛选条件列在 Inspector 面板的输出选项卡下。对于 任务状态 状态，您选择的任何输出筛选条件都将按以下顺序处理：

1. ResultSelector：使用此筛选条件来操纵状态的结果。您可以使用部分结果构造一个新的 JSON 对象。
2. ResultPath：使用此筛选条件选择要传递到输出的状态输入和任务结果的组合。
3. OutputPath：使用此筛选条件筛选 JSON 输出，用于选择将结果中的哪些信息传递到下一个状态。



使用 ResultSelector

`ResultSelector` 是用于以下状态的可选输出筛选条件：

- 任务状态 状态，即状态浏览器的操作选项卡中列出的所有状态。
- Map 状态，在状态浏览器的流选项卡中。
- Parallel 状态，在状态浏览器的流选项卡中。

`ResultSelector` 可用于构造由一个或多个键值对组成的自定义 JSON 对象。每对值可以是静态值，也可以通过路径从状态的结果中选择。

> **Note**
>
> 要指定参数使用路径引用结果中的 JSON 节点，请使用 `.$` 作为参数名称的结尾。

Example 使用 ResultSelector 过滤器的示例

在此示例中，您可以使用ResultSelector操作亚马逊 EMR CreateCluster API 调用对亚马逊 EMR 状态的响应。CreateCluster以下是 Amazon EMR CreateCluster API 调用的结果。

```
{
  "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"
  }
}
```

要选择部分信息并传递带有静态值的附加键值对，请在状态的 “输出” 选项卡下的ResultSelector字段中指定以下内容。

```
{
 "result": "found",
 "ClusterId.$": "$.output.ClusterId",
 "ResourceType.$": "$.resourceType"
 }
```

使用 ResultSelector 会产生以下结果。

```
{
 "result": "found",
 "ClusterId": "AKIAIOSFODNN7EXAMPLE",
 "ResourceType": "elasticmapreduce"
}
```

使用 ResultPath

状态的输出可以是其输入的副本、其生成的结果或其输入和结果的组合。使用 ResultPath 可控制传递到状态输出的上述两种内容的组合。有关更多 ResultPath 的用例，请参阅 ResultPath。

ResultPath 是用于以下状态的可选输出筛选条件：

- 任务状态 状态，即状态浏览器的操作选项卡中列出的所有状态。
- Map 状态，在状态浏览器的流选项卡中。
- Parallel 状态，在状态浏览器的流选项卡中。
- Pass 状态，在状态浏览器的流选项卡中。

ResultPath 可用于将结果添加到原始状态输入中。指定的路径表示添加结果的位置。

Example 使用 ResultPath 过滤器的示例

假设以下是一个 Task 状态的输入。

```
{
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

该 Task 状态的结果为以下内容。

```
Hello, AWS Step Functions
```

您可以通过应用 ResultPath 并输入指示在何处添加结果的参考路径，来将此结果添加到该状态的输入中，例如 $.taskresult：

借助此 ResultPath，以下是作为状态输出传递的 JSON。

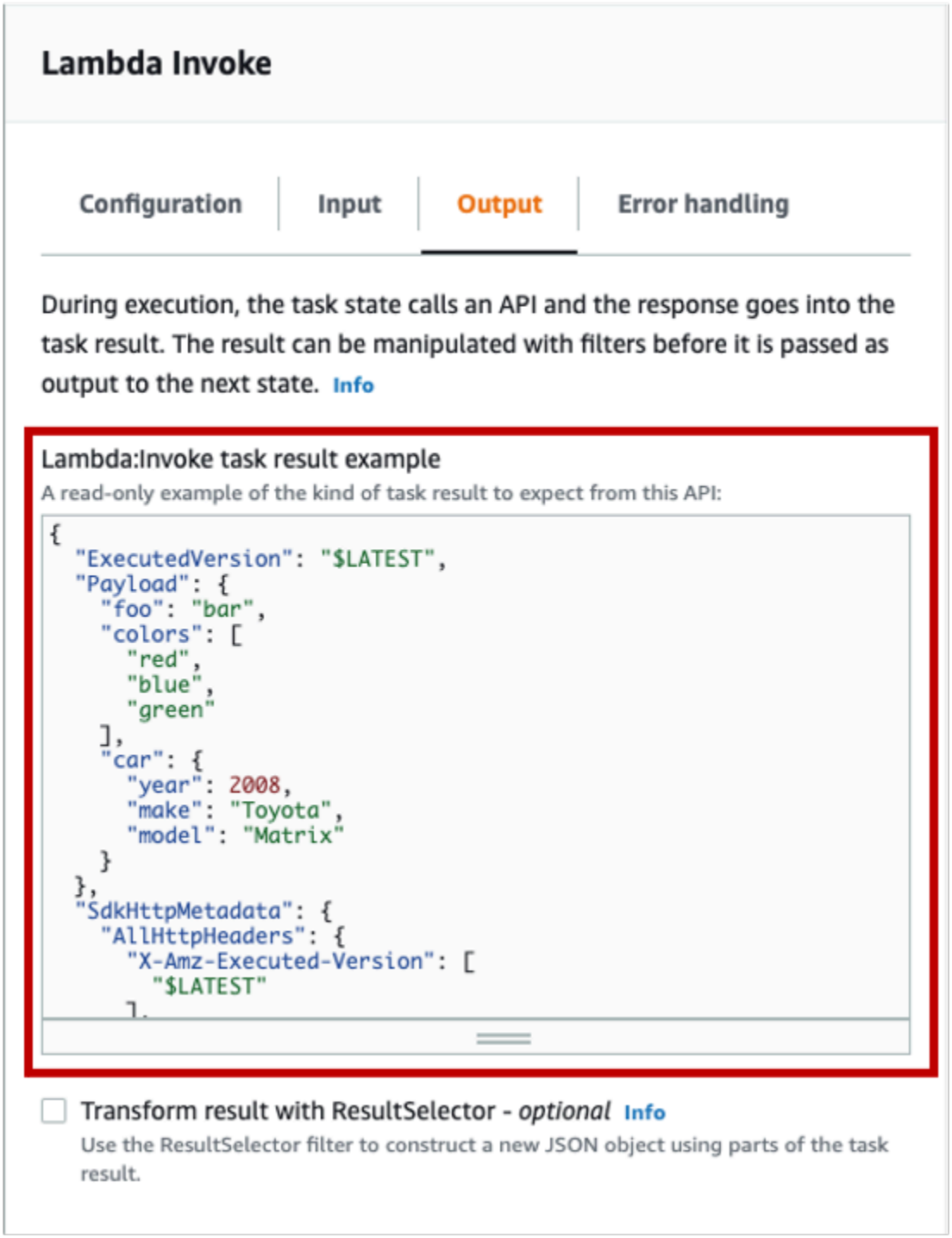
```
{
  "details": "Default example",
  "who": "AWS Step Functions",
  "taskresult": "Hello, AWS Step Functions!"
}
```

使用 OutputPath

`OutputPath` 筛选条件可用于筛选出不需要的信息，并且仅传递所需的 JSON 部分。`OutputPath` 是一个以 `$` 开头的字符串，用于标识 JSON 文本中的节点。

Example 使用 OutputPath 过滤器的示例

Lambda 调用 API 调用除了有效负载外，还会返回元数据，这是 Lambda 函数的结果。此 API 调用的响应示例显示在状态的输出选项卡下。



您可以使用 `OutputPath` 筛选掉其他元数据。默认情况下，通过 Workflow Studio 创建的 Lambda Invoke 状态OutputPath筛选器的值为。`$.Payload`此默认值会删除额外的元数据，并返回相当于直接运行 Lambda 函数的输出。

Lambda 调用任务结果示例和输出筛选条件的 `$.Payload` 值将以下 JSON 数据作为输出传递。

```
{
 "foo": "bar",
 "colors": [
      "red",
      "blue",
      "green"
 ],
 "car": {
      "year": 2008,
      "make": "Toyota",
      "model": "Matrix"
 }
}
```

> **Note**
>
> 由于 OutputPath 筛选条件是最后一个生效的输出筛选条件，因此如果您使用 ResultSelector 或 ResultPath 之类的其他输出筛选条件，则应相应地修改 OutputPath 筛选条件的 $.Payload 默认值。

Workflow Studio 中的执行角色

每台Step Functions状态机都需要一个 AWS Identity and Access Management (IAM) 角色，该角色向状态机授予对 AWS 服务 资源执行操作或调用第三方 API 的权限。此角色称为执行角色。对于每个操作，此角色必须包含 IAM 策略，例如，允许状态机调用 AWS Lambda 函数、运行 AWS Batch 作业或调用 Stripe API 的策略。在以下情况下，Step Functions 需要您提供执行角色：

- 您可以在控制台、 AWS 软件开发工具包或 AWS CLI 使用 CreateStateMachineAPI 创建状态机。
- 您可以在控制台、 AWS 软件开发工具包或AWS CLI使用 TestStateAPI 测试状态。

Workflow Studio 的功能可让您轻松管理工作流的执行角色。

主题

- 关于自动生成的角色
- 自动生成角色
- 解决角色生成问题
- 在 Workflow Studio 中测试 HTTP 任务的角色

- 在 Workflow Studio 中测试优化的服务集成的角色
- 在工作流工作室中测试 AWS SDK 服务集成的角色
- 在 Workflow Studio 中测试流状态的角色

关于自动生成的角色

当您在 Step Functions 控制台中创建状态机时，Workflow Studio 可以自动为您创建一个包含必需 IAM 策略的执行角色。Workflow Studio 会分析您的状态机定义，并生成具有执行工作流所需的最低权限的策略。

Workflow Studio 可以为以下各项生成 IAM 策略：

- 调用第三方 API 的 HTTP 任务。
- AWS 服务 使用经过优化的集成（例如 Invoke DynamoDB GetItem、或 AWS Glue StartJobRun）Lambda调用其他任务状态的任务状态。
- 运行嵌套工作流的 Task 状态。
- 分布式 Map 状态，包括用于启动子工作流执行、列出 Amazon S3 桶以及读取或写入 S3 对象的策略。
- X-Ray 跟踪。在 Workflow Studio 中自动生成的每个角色都包含一个策略，该策略向状态机授予向 X-Ray 发送跟踪的权限。
- 使用 CloudWatch Logs 进行日志记录（当在状态机上启用日志记录时）。

Workflow Studio 无法为 AWS 服务 使用 AWS SDK 集成调用其他任务状态的任务状态生成IAM策略。

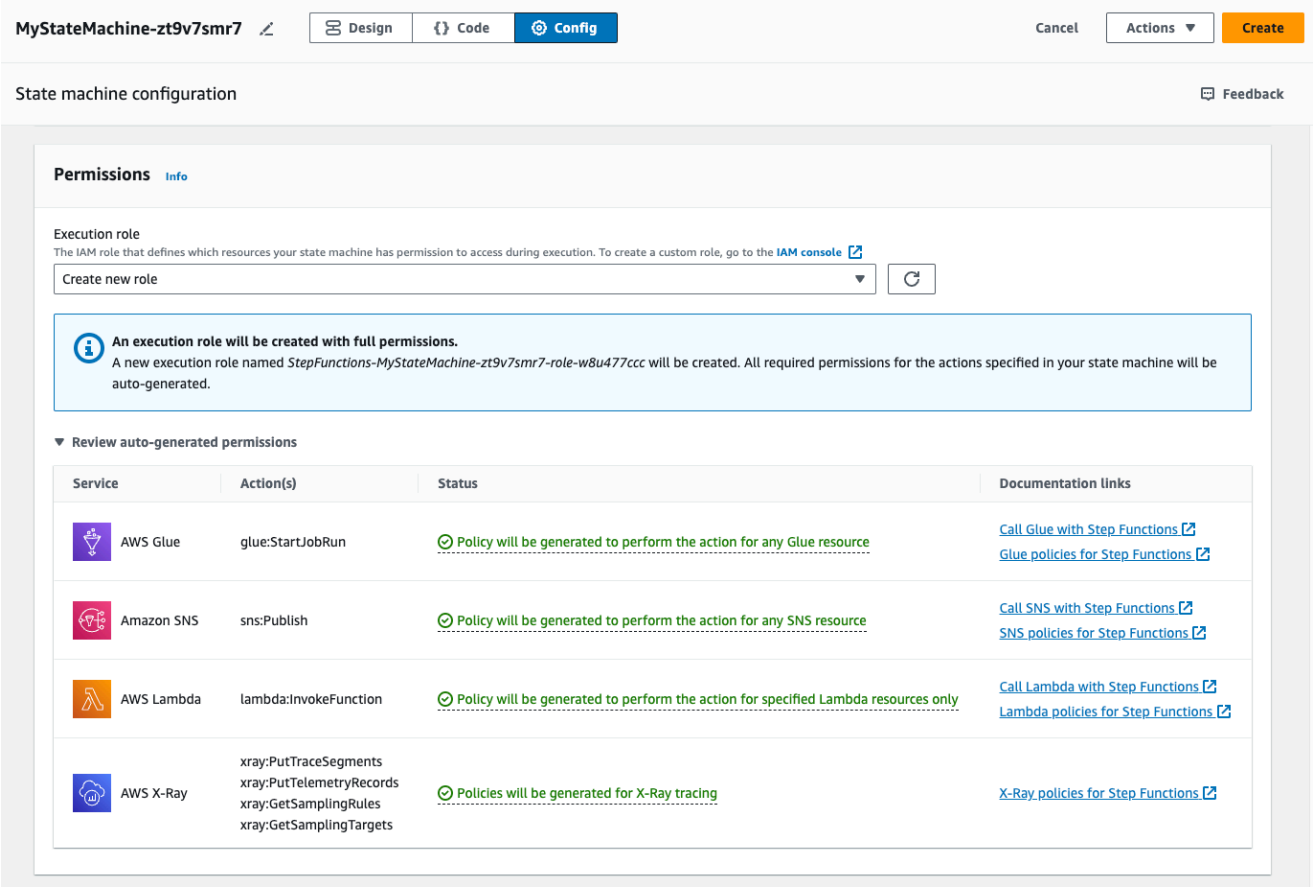
自动生成角色

1. 打开 Step Functions 控制台，然后选择创建状态机。

   您也可以更新现有状态机。如果您要更新状态机，请参阅步骤 4。
2. 在选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 选择配置选项卡。
5. 向下滚动到权限部分，然后执行以下操作：

   a. 对于执行角色，请确保保留默认选择创建新角色。

Workflow Studio 会自动为状态机定义中的每个有效状态生成所有必需的 IAM 策略。它会显示一条横幅，指明将创建具有完全权限的执行角色。



Tip

要查看 Workflow Studio 为状态机自动生成的权限，请选择查看自动生成的权限。

Note

如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

如果 Workflow Studio 无法生成所有必需的 IAM 策略，则会显示一条横幅，指明无法为某些操作自动生成权限。将创建仅具有部分权限的 IAM 角色。有关如何添加缺失权限的信息，请参阅解决角色生成问题。

b. 如果要创建状态机，请选择创建。否则，请选择保存。

c. 在出现的对话框中，选择确认。

Workflow Studio 会保存您的状态机并创建新的执行角色。

解决角色生成问题

在以下情况下，Workflow Studio 无法自动生成具有所有必需权限的执行角色：

- 状态机中出现错误。请务必在 Workflow Studio 中解决所有验证错误。另外，请务必解决在保存过程中遇到的任何服务器端错误。
- 您的状态机包含使用 AWS SDK 集成的任务。在这种情况下，Workflow Studio 无法自动生成 IAM 策略。Workflow Studio 会显示一条横幅，指明无法为某些操作自动生成权限。将创建仅具有部分权限的 IAM 角色。在查看自动生成的权限表中，选择状态中的内容，以了解有关您的执行角色缺少的策略的更多信息。Workflow Studio 仍然可以生成执行角色，但该角色不会包含所有操作的 IAM 策略。请参阅文档链接下的链接，编写您自己的策略，并在角色生成后将其添加到角色中。即使在保存状态机之后，这些链接也仍然可用。

在 Workflow Studio 中测试 HTTP 任务的角色

您需要一个执行角色来测试 HTTP 任务状态。如果您没有具有足够权限的角色，请使用以下选项之一创建角色：

- 通过 Workflow Studio 自动生成一个角色（推荐）- 这是安全的选项。关闭测试状态对话框并按照自动生成角色中的说明操作。这需要您先创建或更新状态机，然后返回 Workflow Studio 测试状态。
- 使用具有管理员权限的角色-如果您有权创建对中的所有服务和资源具有完全访问权限的角色 AWS，则可以使用该角色来测试工作流程中的任何类型的状态。为此，你可以创建一个Step Functions服务角色并在IAM控制台 https://console.aws.amazon.com/iam/ 中向其添加AdministratorAccess 策略。

在 Workflow Studio 中测试优化的服务集成的角色

您需要为调用优化的服务集成的 Task 状态指定执行角色。如果您没有具有足够权限的角色，请使用以下选项之一创建角色：

- 使用 Workflow Studio 中的文档链接来编写自己的 IAM 策略（推荐）- 这是安全的选项。关闭测试状态对话框并按照自动生成角色中的说明操作。这需要您先创建或更新状态机，然后返回 Workflow Studio 测试状态。
- 使用具有管理员权限的角色-如果您有权创建对中的所有服务和资源具有完全访问权限的角色 AWS，则可以使用该角色来测试工作流程中的任何类型的状态。为此，你可以创建一个Step Functions服务角色并在IAM控制台 https://console.aws.amazon.com/iam/ 中向其添加AdministratorAccess 策略。

在工作流工作室中测试 AWS SDK 服务集成的角色

您需要为调用 AWS SDK 集成的 Task 状态指定执行角色。如果您没有具有足够权限的角色，请使用以下选项之一创建角色：

- 使用 Workflow Studio 中的文档链接来编写自己的 IAM 策略（推荐）- 这是安全的选项。关闭测试状态对话框并按照自动生成角色中的说明操作。这需要您先创建或更新状态机，然后返回 Workflow Studio 测试状态。执行以下操作：
  1. 关闭测试状态对话框
  2. 选择配置选项卡以查看配置模式。
  3. 向下滚动到权限部分。
  4. Workflow Studio 会显示一条横幅，指明无法为某些操作自动生成权限。将创建仅具有部分权限的 IAM 角色。选择查看自动生成的权限。
  5. 查看自动生成的权限表会显示一行，列出与您要测试的 Task 状态相对应的操作。请参阅文档链接下的链接，将您自己的 IAM 策略编写到自定义角色中。
- 使用具有管理员权限的角色-如果您有权创建对中的所有服务和资源具有完全访问权限的角色 AWS，则可以使用该角色来测试工作流程中的任何类型的状态。为此，你可以创建一个Step Functions服务角色并在IAM控制台 https://console.aws.amazon.com/iam/ 中向其添加AdministratorAccess 策略。

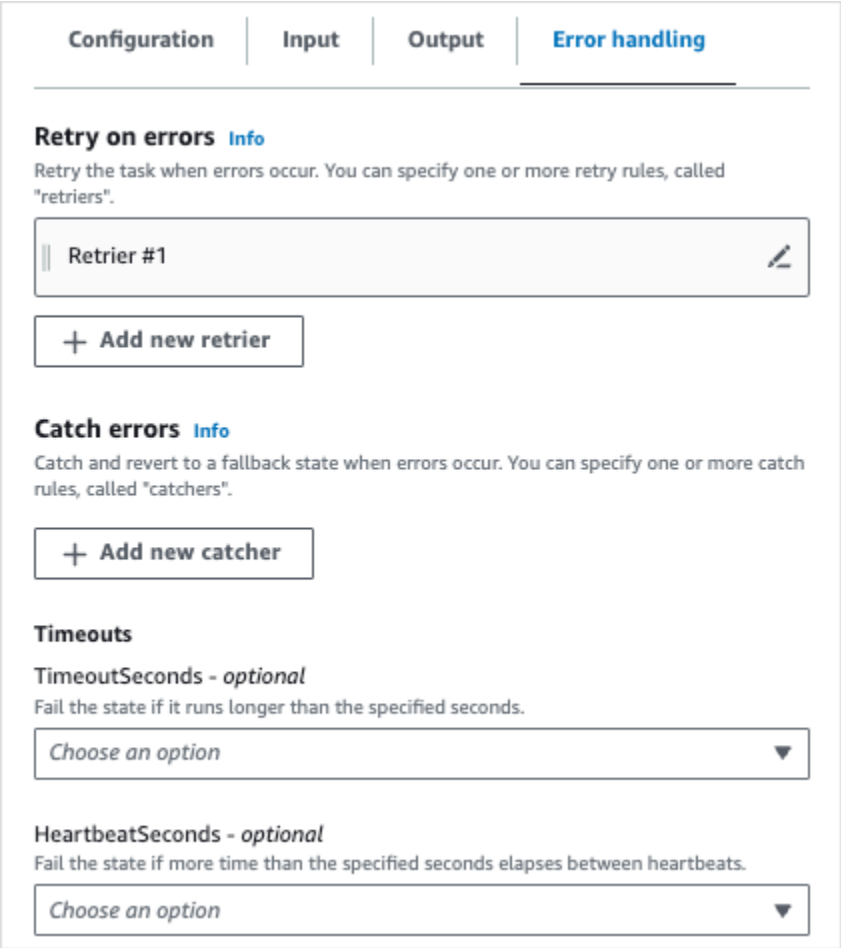
在 Workflow Studio 中测试流状态的角色

您需要一个执行角色，以在 Workflow Studio 中测试流状态。流状态是指定向执行流的状态，例如 Choice、Parallel、Map、Pass、Wait、Succeed 或 Fail。TestStateAPI 不适用于地图或平行状态。您可以使用以下选项之一创建用于测试流状态的角色：

- 在你的 AWS 账户 （推荐）中使用任何角色 — Flow 状态不需要任何特定的IAM策略，因为它们不调用 AWS 操作或资源。因此，您可以在中使用任何IAM角色 AWS 账户。
  1. 在测试状态对话框中，从执行角色下拉列表中选择任意角色。
  2. 如果下拉列表中没有显示任何角色，请执行以下操作：
     a. 在 IAM 控制台 https://console.aws.amazon.com/iam/ 中，选择角色。
     b. 从列表中选择一个角色，然后从角色详细信息页面复制其 ARN。您需要在测试状态对话框中提供此 ARN。
     c. 在测试状态对话框中，从执行角色下拉列表中选择输入角色 ARN。
     d. 将 ARN 粘贴到角色 ARN 中。
- 使用具有管理员权限的角色-如果您有权创建对中的所有服务和资源具有完全访问权限的角色 AWS，则可以使用该角色来测试工作流程中的任何类型的状态。为此，你可以创建一个Step Functions服务角色并在IAM控制台 https://console.aws.amazon.com/iam/ 中向其添加AdministratorAccess 策略。

错误处理

默认情况下，当某个状态报告错误时，Step Functions 会导致工作流执行完全失败。对于操作和某些流状态，您可以配置 Step Functions 处理错误的方式。即使您配置了错误处理，某些错误仍可能导致工作流执行失败。有关更多信息，请参阅 Step Functions 中的错误处理。在 Workflow Studio 中，在 Inspector 面板的错误处理选项卡中配置错误处理。



出错时重试

您可以向操作状态和 Parallel 流状态添加一条或多条规则，以便在发生错误时重试任务。这些规则被称为重试器。要添加重试器，请选择重试器 #1 框中的编辑图标，然后配置其选项：

- （可选）在注释字段中，添加您的注释。它不会影响工作流，但可以用来为您的工作流添加注释。
- 将光标置于错误字段中，选择将触发重试器的错误，或者输入自定义错误名称。您可以选择或添加多个错误。
- （可选）设置间隔。这是 Step Functions 首次重试之前的时间（以秒为单位）。随后将按间隔进行其他重试，您可以根据最大尝试次数和回退率进行配置。
- （可选）设置最大尝试次数。这是 Step Functions 导致执行失败之前的最大重试次数。
- （可选）设置回退率。这是一个乘数，它决定每次尝试的重试间隔将增加多少。

> **Note**
>
> 并非所有错误处理选项都适用于所有状态。默认情况下，Lambda 调用会配置了一个重试器。

捕获错误

您可以向操作状态以及 Parallel 和 Map 流状态添加一条或多条规则以捕获错误。这些规则被称为捕获器。要添加捕获器，请选择添加捕获器手，然后配置其选项：

- （可选）在注释字段中，添加您的注释。它不会影响工作流，但可以用来为您的工作流添加注释。
- 将光标置于错误字段中，选择将触发捕获器的错误，或者输入自定义错误名称。您可以选择或添加多个错误。
- 在回退状态字段中，选择一种回退状态。这是捕获到错误后，工作流将移至的下一个状态。
- （可选）在该ResultPath字段中，添加`ResultPath`过滤器以将错误添加到原始状态输入中。`ResultPath`必须是有效的JsonPath。这将被发送到回退状态。

超时

您可以为操作状态配置一个超时，用于设置状态在失败之前可以运行的最大秒数。使用超时可避免执行卡顿。要配置超时，请输入状态在执行失败之前应等待的秒数。有关超时的更多信息，请参阅 任务状态 状态中的 `TimeoutSeconds`。

HeartbeatSeconds

您可以配置任务发送的检测信号或定期通知。如果设置了检测信号间隔，并且状态未在配置的时间间隔内发送检测信号通知，则该任务将被标记为失败。要配置检测信号，请设置一个秒数，该值需为非零正整数。有关更多信息，请参阅 任务状态 状态中的 `HeartBeatSeconds`。

教程：学习使用 AWS Step Functions Workflow Studio

在本教程中，您将学习使用 AWS Step Functions Workflow Studio 的基础知识。在 Workflow Studio 的设计模式中，您将创建一个包含多个状态的状态机，包括`Pass`、`Choice`、`Fail`、`Wait` 和 `Parallel`。您将使用拖放特征来搜索、选择和配置这些状态。然后，您将查看工作流自动生成的 Amazon States Language (ASL) 定义。您还将使用 Workflow Studio 的代码模式来编辑工作流定义。然后，您将退出 Workflow Studio，运行状态机并查看执行详细信息。

在本教程中，您还将学习如何更新状态机以及如何查看执行输出中的更改。最后，您将执行清理步骤并删除状态机。

完成本教程后，您将知道如何使用 Workflow Studio 在设计和代码模式下创建和配置工作流程。您还将知道如何更新、运行和删除状态机。

> **Note**
>
> 在开始本教程之前，请确保完成本教程的先决条件。

主题

- 第 1 步：导航到 Workflow Studio
- 第 2 步：创建状态机
- 第 3 步：查看自动生成的 Amazon States Language 定义
- 第 4 步：在“代码”模式下编辑工作流定义
- 第 5 步：保存状态机
- 第 6 步：运行状态机
- 第 7 步：更新状态机
- 第 8 步：清除

第 1 步：导航到 Workflow Studio

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。

第 2 步：创建状态机

在 Workflow Studio 中，状态机是工作流的一种图形化表示。使用 Workflow Studio，您可以定义、配置和检查工作流的各个步骤。在以下步骤中，您将使用 Workflow Studio 的设计模式创建状态机。

创建状态机

1. 确保处于 Workflow Studio 的设计模式。

2. 从左侧状态浏览器中选择流选项卡。然后，将一个 Pass 状态拖到标有将第一个状态拖至此处的空白状态处。
3. 从流选项卡将一个 Choice 状态拖放到 Pass 状态下方。
4. 在状态名称中，替换 Choice 默认名称。在本教程中，请使用名称 **IsHelloWorldExample**。
5. 将另一个 Pass 状态拖放到该IsHelloWorldExample状态的一个分支上。然后，将失败状态拖放到该IsHelloWorldExample状态的另一个分支下方。
6. 选择 Pass (1) 状态，并将其重命名为 **Yes**。将 Fail 状态重命名为 **No**。
7. 使用布尔变量`IsHelloWorldExample`指定IsHelloWorldExample状态的分支逻辑。

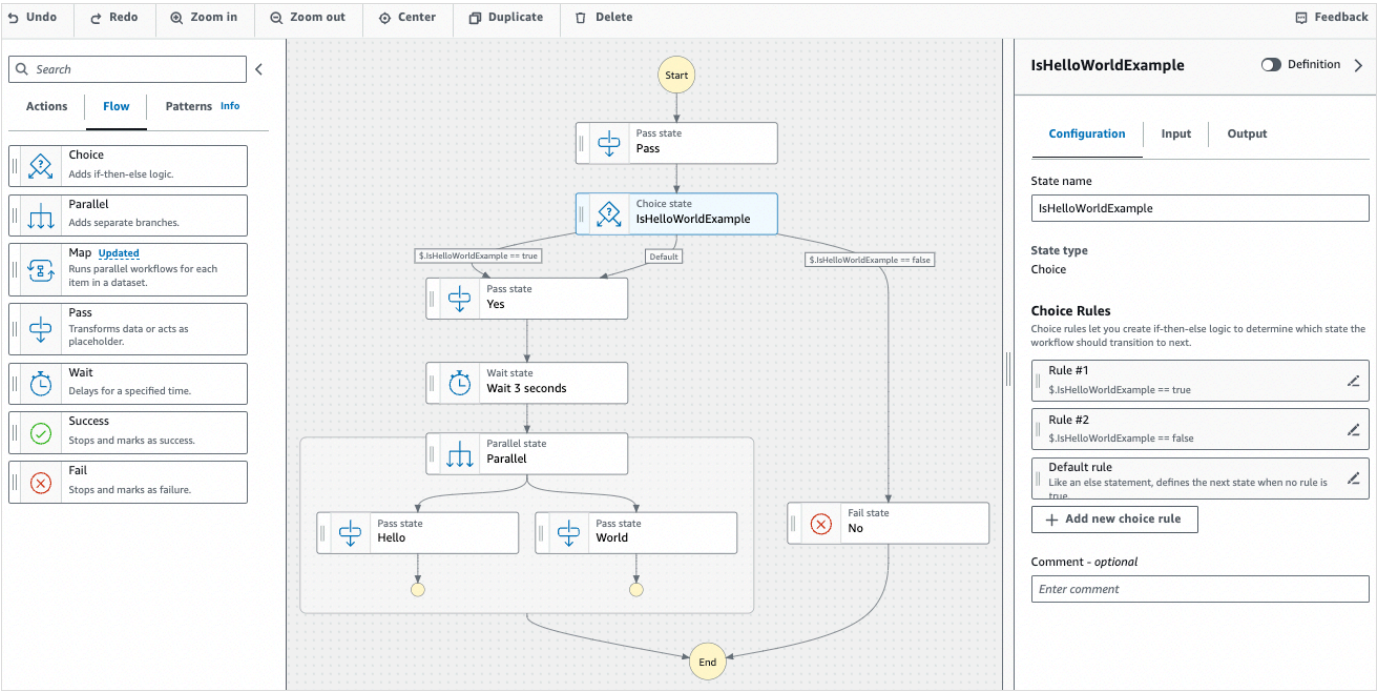
   如果 `IsHelloWorldExample` 是 `False`，则工作流将进入 No 状态。否则，工作流将在 Yes 状态下继续其执行流程。

   要定义分支逻辑，请执行以下操作：

   a. 在上选择IsHelloWorldExample状态画布，然后在 “选择规则” 下，选择 “规则 #1” 框中的编辑图标来定义首选规则。
   b. 选择添加条件。
   c. 在规则 #1 的条件对话框中，在变量下输入 **$.IsHelloWorldExample**。
   d. 在运算符下选择等于。
   e. 在值下选择布尔常数，然后从下拉列表中选择 true。
   f. 选择保存条件。
   g. 确保在然后下一个状态为：下拉列表中选择 Yes。
   h. 选择添加新选择规则，然后选择添加条件。
   i. 在规则 #2 框中，通过重复子步骤 7.c 到 7.f 来定义 `IsHelloWorldExample` 变量值为 false 时的第二个选择规则。对于步骤 7.e，请选择 false 而非 true。
   j. 在规则 #2 框中，从然后下一个状态为：下拉列表中选择 No。
   k. 在默认规则框中，选择编辑图标定义默认选择规则，然后从下拉列表中选择 Yes。
8. 在 Yes 状态之后添加一个 Wait 状态，并将其命名 **Wait 3 sec**。然后，通过执行以下步骤将等待时间配置为三秒：

   a. 在选项下，保留默认选择等待固定的时间间隔。
   b. 在秒下，确保选中输入秒，然后在框中输入 **3**。
9. Wait 3 sec 状态后，添加一个 Parallel 状态。在其两个分支中添加两个 Pass 状态。命名第一个 Pass 状态为 **Hello**。命名第二个 Pass 状态为 **World**。

完成的工作流将如下所示：



第 3 步：查看自动生成的 Amazon States Language 定义

当您将状态从流选项卡拖放到画布上时，Workflow Studio 会自动实时撰写工作流的 Amazon States Language (ASL) 定义。在 Inspector 面板中，选择定义切换按钮查看此定义，或切换到代码模式，根据需要编辑此定义。有关编辑工作流定义的信息，请参阅本教程的di's第 4 步。

- （可选）在 Inspector 面板上选择定义，然后查看状态机的工作流。

  以下示例代码显示了 `IsHelloWorldExample` 状态机自动生成的 Amazon States Language 定义。您在 Workflow Studio 中添加的 `Choice` 状态用于根据您在第 2 步中定义的分支逻辑来确定执行流程。

```
{
  "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
            }
          }
        }
      ]
    }
  }
}
```

第 4 步：在“代码”模式下编辑工作流定义

Workflow Studio 的代码模式提供了一个集成的代码编辑器，用于查看和编辑工作流的 ASL 定义。

1. 选择代码，切换到代码模式。
2. 在 Parallel状态的定义后，放置光标并按 **Enter** 键。
3. 按下 **Ctrl+space** 可查看可在 Parallel 状态之后添加的状态列表。
4. 从选项列表中选择 Pass 状态。代码编辑器为 Pass 状态添加样板代码。
5. 添加此状态会导致您的工作流定义出现错误。在 Parallel 状态的定义中，替换 `"End": true` 为 **`"Next": "PassState"`**。
6. 在您添加的 Pass 状态定义中，进行以下更改：

   a. 删除结果节点。

   b. 删除 `"ResultPath": "$.result"`，和 `"Next": "NextState"`。

   c. 在 `"Type": "Pass"`，后面，输入 **`"End": true`**。

   d. 在 Pass 状态定义之后添加一个 **`,`**。

现在，您的工作流定义应类似于以下定义。

```
{
```

```
  "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"
    }
  }
}
```

第 5 步：保存状态机

1. 选择 Config more 或选择默认状态机名称旁边的编辑图标MyStateMachine。在状态机配置中，指定一个名称。例如，输入 **HelloWorld**。
2. （可选）指定其他工作流设置，例如状态机类型及其执行角色。在本教程中，请保留状态机配置中的所有默认选项。
3. 选择创建。
4. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色配置，返回配置 模式。

有关配置模式的更多信息，请参阅 Workflow Studio 的配置模式。

第 6 步：运行状态机

状态机执行是指运行工作流执行任务的实例。

1. 在状态机页面上，选择HelloWorld状态机。

2. 在HelloWorld页面上，选择开始执行。
3. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

4. 在输入框中，以 JSON 格式输入执行的输入值。根据您的输入，`IsHelloWorldExample` 变量决定将执行哪个状态机流程。现在，请使用以下输入值：

```
{
   "IsHelloWorldExample": true
}
```

> **Note**
>
> 虽然指定执行输入是可选操作，但在本教程中，必须指定与上述示例输入类似的执行输入。运行状态机时，该输入值将在 `Choice` 状态中被引用。

5. 选择启动执行。
6. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

   在本教程中，如果您输入的输入值为 `"IsHelloWorldExample": true`，则应看到以下输出：

```
{
   "IsHelloWorldExample": true
},
{
   "IsHelloWorldExample": true
}
```

第 7 步：更新状态机

更新状态机时，更新内容最终会保持一致。短时间后，所有新启动的执行都将反映状态机更新后的定义。所有当前正在运行的执行将在先前的定义下运行完成。

在此步骤中，您将在 Workflow Studio 的设计模式下更新状态机。您将在名为 World 的Pass 状态中添加一个 `Result` 字段。

1. 在标题为您的执行 ID 的页面上，选择 编辑状态机。
2. 确保您处于设计模式。
3. 在画布上选择名为 World 的 Pass 状态，然后选择输出。
4. 在结果框中输入 **`"World has been updated!"`**。
5. 选择保存。
6. （可选）在定义区域，查看工作流中更新的 Amazon States Language 定义。

```
{
      "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. 选择执行。
8. 在新选项卡中打开的启动执行对话框中，提供以下执行输入。

```
{
   "IsHelloWorldExample": true
}
```

9. 选择启动执行。
10. （可选）在图表视图中，选择 World 步骤，然后选择输出。输出为 World has been updated!

第 8 步：清除

删除您的状态机

1. 在导航菜单中，选择状态机。
2. 在状态机页面上，选择 HelloWorld，然后选择删除。
3. 在删除状态机对话框中，键入 **`delete`** 确认删除。
4. 选择删除。

   如果成功删除，则屏幕顶部会出现一个绿色的状态条。绿色状态栏通知您状态机已标记为删除。当状态机中所有正在进行的执行都停止运行时，状态机将被删除。

删除执行角色

1. 打开 IAM 的角色页面。
2. 选择 Step Functions 为您创建的 IAM 角色。例如，StepFunctionsHelloWorld-角色示例。
3. 选择删除角色。
4. 选择是，删除。

Step Functions 教程

本部分中的教程可帮助您了解有关使用 AWS Step Functions 的不同方面。

要完成这些教程，你需要一个 AWS 帐户。如果你没有 AWS 账户，请导航至 https://aws.amazon.com/ 并选择 “创建 AWS 账户”。

主题

- 创建使用 Lambda 的 Step Functions 状态机
- 使用状态机处理 Step Functions 函数错误情形
- 使用内联 Map 状态重复操作
- 使用分布式 Map 复制大规模 CSV 数据
- 使用 Lambda 函数处理整批数据
- 使用 Lambda 函数处理单个数据项
- 启动状态机执行以响应 Amazon S3 事件
- 使用 API Gateway 创建 Step Functions API
- 使用 AWS SAM 创建 Step Functions 状态机
- 使用 Step Functions 创建活动状态机
- 使用 Lambda 迭代循环
- 将长时间运行的工作流执行作为新执行继续执行
- 部署示例人员批准项目
- 在 Step Functions 中查看 X-Ray 跟踪
- 使用 AWS 软件开发工具包服务集成收集 Amazon S3 存储桶信息

创建使用 Lambda 的 Step Functions 状态机

在本教程中，您将创建一个用于调用 AWS Lambda 函数的单步工作流程。 AWS Step Functions

> **Note**
>
> Step Functions 基于状态机和任务。在 Step Functions 中，状态机被称为工作流，这是一系列事件驱动的步骤。工作流程中的每个步骤都称为状态。例如，任务状态表示另一个 AWS 服务执行的工作单元，例如调用另一个服务 AWS 服务 或 API。
> 有关更多信息，请参阅：

- 什么是 AWS Step Functions？
- 致电其他 AWS 服务

Lambda 非常适合于 Task 状态，因为 Lambda 函数无服务器 且易于编写。你可以在 AWS Management Console 或你最喜欢的编辑器中编写代码。 AWS 处理为你的函数提供计算环境并运行它的细节。

本主题内容：

- 第 1 步：创建 Lambda 函数
- 第 2 步：测试 Lambda 函数
- 第 3 步：创建状态机
- 第 4 步：运行状态机

第 1 步：创建 Lambda 函数

您的 Lambda 函数接收事件数据并返回问候语。

> **Important**
>
> 确保您的 Lambda 函数与状态机位于同一个 AWS 账户和 AWS 区域下。

1. 打开 Lambda 控制台，然后选择创建函数。
2. 在创建函数页面上，选择从头开始创作。
3. 对于函数名称，请输入 `HelloFunction`。
4. 保留所有其他选项的默认选择，然后选择创建函数。
5. 创建 Lambda 函数后，复制页面右上角显示的该函数的 Amazon 资源名称 (ARN)。要复制 ARN，请单击

   。

   以下是示例 ARN：

   ```
   arn:aws:lambda:us-east-1:123456789012:function:HelloFunction
   ```
6. 将以下 Lambda 函数的代码复制到页面的***HelloFunction***代码源部分。

```
export const handler = async(event, context, callback) => {
    callback(null, "Hello from " + event.who + "!");
};
```

此代码将使用输入数据的 who 字段组装问候语，该输入数据是由传送到您的函数的 event 对象提供的。以后当您启动新的执行时，您将为此函数添加输入数据。callback 方法将从您的函数返回组装的问候语。

7. 选择部署。

第 2 步：测试 Lambda 函数

测试您的 Lambda 函数以查看其运行情况。

1. 选择测试。
2. 对于事件名称，输入 HelloEvent。
3. 使用以下内容替换时间 JSON。

```
{
    "who": "AWS Step Functions"
}
```

"who" 条目对应于您的 Lambda 函数中的 event.who 字段，用于编写问候语。运行状态机时，您将输入相同的输入数据。

4. 选择保存，然后选择测试。
5. 在执行结果下，展开详细信息以查看测试结果。

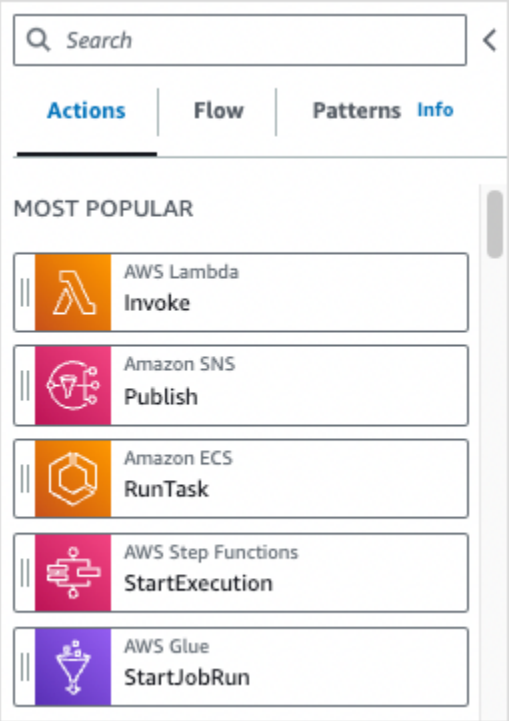
第 3 步：创建状态机

使用 Step Functions 控制台创建状态机，用于调用您在第 1 步中创建的 Lambda 函数。

1. 打开 Step Functions 控制台，然后选择创建状态机。

> **Important**
>
> 确保您的状态机与您之前创建的 Lambda 函数位于相同的 AWS 账户和区域下。

2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 在左侧的状态浏览器中，确保已选择操作选项卡。然后执行以下操作：
   - 将 AWS Lambda 调用 API 拖放到标有将第一个状态拖至此处的空白状态处。



5. 在右侧的 Inspector 面板中，配置 Lambda 函数：
   a. 在 API 参数部分，在函数名称下拉列表中选择您之前创建的 Lambda 函数。
   b. 保留有效负载下拉列表中的默认选择。
6. （可选）选择定义查看状态机 Amazon States Language (ASL) 的定义，该定义是根据您在操作选项卡和 Inspector 面板中的选择自动生成的。
7. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   例如，输入名称 **`LambdaStateMachine`**。

> **Note**
>
> 状态机、执行和活动任务的名称长度不得超过 80 个字符。对于您的账户和 AWS 地区，这些名称必须是唯一的，并且不得包含以下任何内容：

- 空格
- 通配符 (? *)
- 方括号字符 (< > { } [ ])
- 特殊字符 (" # % \ ^ | ~ ` $ & , ; : /)
- 控制字符 (\\u0000 - \\u001f 或 \\u007f - \\u009f)

如果您的状态机是快速类型，则可以为状态机的多个执行提供相同的名称。即使多个执行的名称相同，Step Functions 也会为每个快速状态机执行生成唯一的执行 ARN。
Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

8. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。

9. 选择创建。
10. 在确认角色创建对话框中，选择确认继续。

    您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 4 步：运行状态机

在创建状态机后，便可以运行它。

1. 在状态机页面上，选择LambdaStateMachine。
2. 选择启动执行。

   随即显示启动执行对话框。

3. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

4. 在输入区域中，将示例数据替换为以下内容。

```
{
    "who" : "AWS Step Functions"
}
```

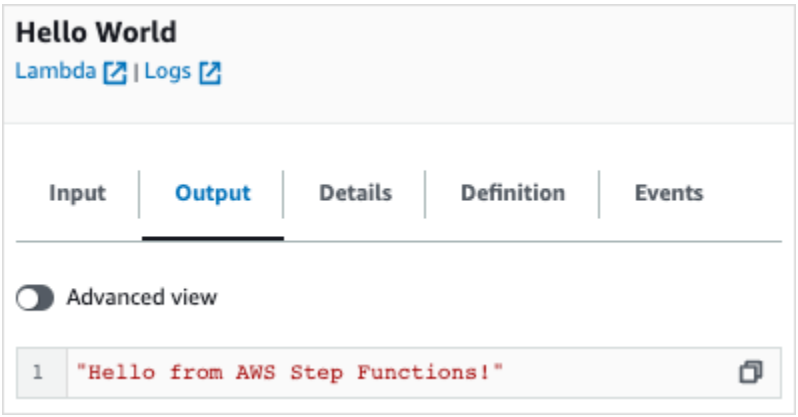
"who" 是键名称，您的 Lambda 函数将使用它来获取所问候人员的姓名。

5. 选择启动执行。

状态机的执行将启动，并显示一个说明正在运行的执行的新页面。

6. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。



> **Note**
>
> 您还可以在从状态机调用 Lambda 时传递有效负载。有关通过在 `Parameters` 字段中传递有效负载来调用 Lambda 的更多信息和示例，请参阅使用 Step Functions 调用 Lambda。

使用状态机处理 Step Functions 函数错误情形

在本教程中，您将创建一个带有回退状态字段的 AWS Step Functions 状态机。该`Catch`字段使用 AWS Lambda 函数根据错误消息类型使用条件逻辑进行响应。这是一种称为函数错误处理 的技术。

有关更多信息，请参阅《AWS Lambda 开发人员指南》中的 Node.js 中的AWS Lambda 函数错误。

> **Note**
>
> 您还可以创建在超时情况下执行重试的状态机，或者创建使用 `Catch` 在出错或超时情况下转换为特定状态的状态机。有关这些错误处理技术的示例，请参阅使用 Retry 和使用 Catch 示例。

本主题内容：

- 第 1 步：创建一个失败的 Lambda 函数
- 第 2 步：测试 Lambda 函数
- 第 3 步：创建使用 Catch 字段的状态机
- 第 4 步：运行状态机

第 1 步：创建一个失败的 Lambda 函数

使用 Lambda 函数来模拟错误情形。

> **Important**
>
> 确保您的 Lambda 函数与状态机位于同一个 AWS 账户和 AWS 区域下。

1. 打开 AWS Lambda 控制台，网址为 https://console.aws.amazon.com/lambda/。

2. 选择创建函数。
3. 选择使用蓝图，在搜索框中输入 `step-functions`，然后选择抛出自定义错误蓝图。
4. 对于 Function name（函数名称），请输入 `FailFunction`。
5. 对于角色，保留默认选择（使用基本 Lambda 权限创建新角色）。
6. 以下代码显示在 Lambda 函数代码窗格中。

```
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` 对象返回错误消息 `This is a custom error!`。
7. 选择创建函数。
8. 创建 Lambda 函数后，复制页面右上角显示的该函数的 Amazon 资源名称 (ARN)。要复制 ARN，请单击

   。

   以下是示例 ARN：

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

9. 选择部署。

第 2 步：测试 Lambda 函数

测试您的 Lambda 函数以查看其运行情况。

1. 在FailFunction页面上，选择 “测试” 选项卡，然后选择 “测试”。您无需创建测试事件。
2. 在执行结果下，展开详细信息以查看测试结果（模拟的错误）。

第 3 步：创建使用 Catch 字段的状态机

通过 Step Functions 控制台创建使用 任务状态 状态和 Catch 字段的状态机。在 Task 状态中添加对您 Lambda 函数的引用。状态机调用 Lambda 函数，该函数将在执行过程中失败。Step Functions 重试函数两次，在两次重试之间使用指数回退。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 选择代码，打开代码编辑器。在代码编辑器中，您可以编写和编辑工作流的 Amazon States Language (ASL) 定义。
5. 粘贴以下代码，但要替换之前在 Resource 字段中创建的 Lambda 函数的 ARN。

```
{
   "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
         }
      }
}
```

这是使用 Amazon States Language 的状态机的说明。它定义了名为 `CreateAccount` 的单个 `Task` 状态。有关更多信息，请参阅状态机结构。

有关 `Retry` 字段的语法的更多信息，请参阅使用 Retry 和使用 Catch 的状态机示例。

> **Note**
>
> Lambda 中未处理的错误在错误输出中报告为 `Lambda.Unknown`。其中包括 out-of-memory 错误和函数超时。您可以匹配 `Lambda.Unknown`、`States.ALL` 或 `States.TaskFailed` 来处理这些错误。当 Lambda 达到最大调用次数时，会出现 `Lambda.TooManyRequestsException` 错误。有关 Lambda 函数错误的更多信息，请参阅《AWS Lambda 开发者指南》中的错误处理和自动重试。

6. （可选）在图表可视化窗格中，查看工作流的实时图形可视化。
7. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   在本教程中，请输入 **`Catchfailure`**。
8. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。
9. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 4 步：运行状态机

在创建状态机后，便可以运行它。

1. 在状态机页面上，选择 Catchfailure。
2. 在 Catchfailure 页面上，选择启动执行。随即显示启动执行对话框。
3. 在启动执行对话框中，执行以下操作：

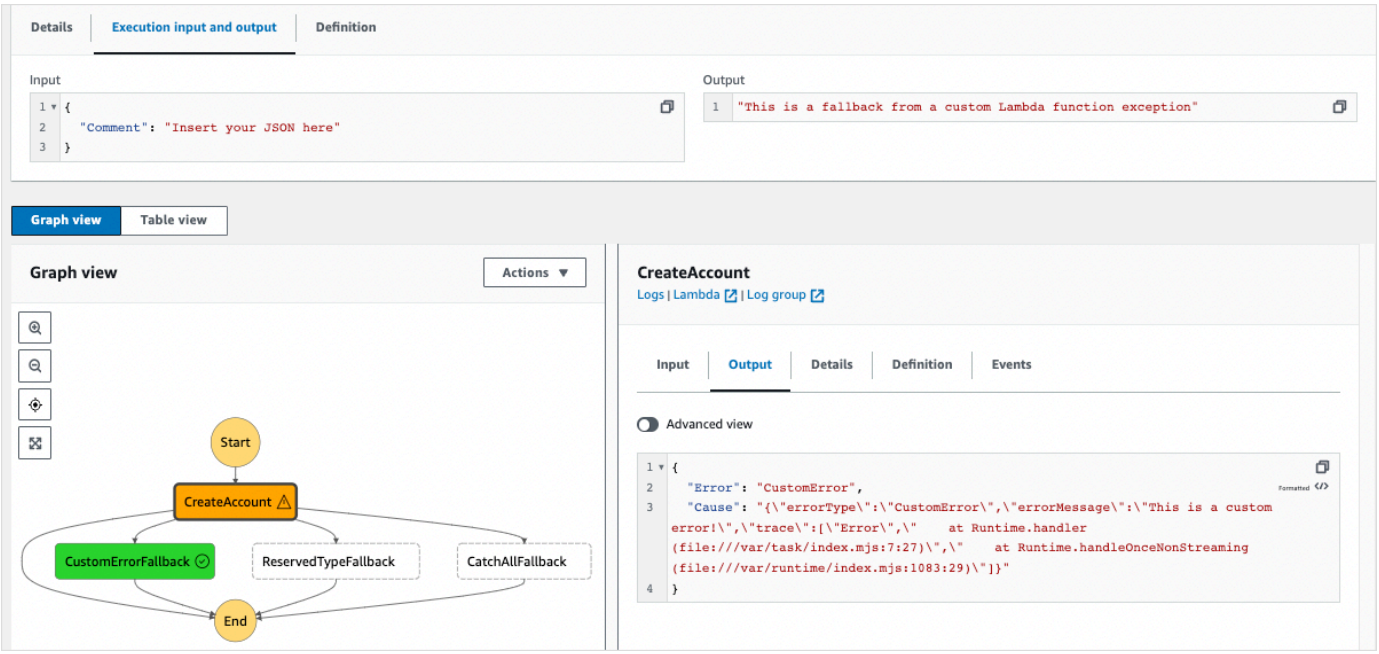
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。
   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

   例如，要查看您的自定义错误消息，请在图表视图中选择CreateAccount步骤，然后选择输出选项卡。



> **Note**
>
> 可以通过使用 `ResultPath` 来保留状态输入以及错误。请参阅 [ResultPath 用于在 a 中同时包含错误和输入 Catch](#)。

使用内联 Map 状态重复操作

本教程可帮助您开始在内联模式中使用 Map 状态。您可以在工作流中使用内联 Map 状态来重复执行操作。有关内联模式的更多信息，请参阅[内联模式下的 Map 状态](#)。

在本教程中，您将使用内联 Map 状态重复生成版本 4 的通用唯一标识符 (v4 UUID)。首先，在 Workflow Studio 中创建一个包含两个 [Pass](#) 状态和一个内联 Map 状态的工作流。然后，配置输入和输出，包括 Map 状态的输入 JSON 数组。Map 状态会返回一个输出数组，其中包含为输入数组中的每个项目生成的 v4 UUID。

内容

- [第 1 步：创建工作流原型](#)
- [第 2 步：配置输入和输出](#)
- [第 3 步：查看自动生成的 Amazon States Language 定义并保存工作流](#)
- [第 4 步：运行状态机](#)

第 1 步：创建工作流原型

在此步骤中，您将使用 Workflow Studio 为工作流创建原型。Workflow Studio 是一款可视化工作流设计器，可在 Step Functions 控制台中使用。您将从流选项卡中选择所需的状态，然后使用 Workflow Studio 的拖放特征来创建工作流原型。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 从流选项卡中，将 Pass 状态拖放到标有将第一个状态拖至此处的空白状态处。
5. 将 Map 状态拖放到 Pass 状态下方。将 Map 状态重命名为 **`Map demo`**。
6. 将第二个 Pass 状态拖放到 Map demo 状态内。
7. 将第二个 Pass 状态重命名为 **`Generate UUID`**。

第 2 步：配置输入和输出

在此步骤中，您将为工作流原型中的所有状态配置输入和输出。首先，使用第一个 Pass 状态将一些固定数据注入工作流。此 Pass 状态将这些数据作为输入传递给 Map demo 状态。在此输入中，您将指定包含 Map demo 状态应迭代的输入数组的节点。然后定义 Map demo 状态应重复的步骤，用于生成 v4 UUID。最后，配置输出，用于在每次重复时返回。

1. 在工作流原型中选择第一个 Pass 状态。在输出选项卡中，在结果下输入以下内容：

```
{
  "foo": "bar",
  "colors": [
    "red",
    "green",
    "blue",
    "yellow",
    "white"
  ]
}
```

2. 选择 Map demo 状态，然后在配置选项卡中执行以下操作：

   a. 选择提供项目数组的路径。

   b. 指定以下参考路径，用于选择包含输入数组的节点：

```
$.colors
```

3. 选择 Generate UUID 状态，然后在输入选项卡中执行以下操作：

   a. 选择使用参数转换输入。

   b. 输入以下 JSON 输入，为每个输入数组项目生成 v4 UUID。您可以使用 States.UUID 内置函数生成 UUID。

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

4. 对于 Generate UUID 状态，选择输出选项卡并执行以下操作：

   a. 选择 “过滤输出” OutputPath。

   b. 输入以下参考路径，选择包含输出数组项目的 JSON 节点：

```
$.uuid
```

第 3 步：查看自动生成的 Amazon States Language 定义并保存工作流

当您将状态从流面板拖放到画布上时，Workflow Studio 会自动实时撰写工作流的 Amazon States Language (ASL) 定义。您可以根据需要编辑此定义。

1. （可选）在 Inspector 面板上选择定义，查看工作流中自动生成的 Amazon States Language 定义。

> **Tip**
>
> 您也可以在 Workflow Studio 的代码编辑器中查看 ASL 的定义。在代码编辑器中，还可以编辑工作流的 ASL 定义。

以下示例显示了为您的工作流自动生成的 Amazon States Language定义。

```
{
    "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. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **InlineMapDemo**。

3. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机配置中的所有默认选项。

4. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 4 步：运行状态机

状态机执行是指运行工作流执行任务的实例。

1. 在InlineMapDemo页面上，选择 “开始执行”。
2. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。
   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

要查看执行输入和输出 side-by-side，请选择执行输入和输出。在输出下，查看 Map 状态返回的输出数组。下面是输出数组的一个示例：

```
[
  "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"
]
```

使用分布式 Map 复制大规模 CSV 数据

本教程将帮助您开始在分布式模式下使用 Map 状态。设置为分布式的 Map 状态被称为分布式 Map 状态。您可以在工作流中使用分布式 Map 状态来迭代大规模 Amazon S3 数据来源。Map 状态将每次迭代作为子工作流执行来运行，从而实现高并发数。有关分布式模式的更多信息，请参阅分布式模式下的 Map 状态。

在本教程中，您将使用分布式地图状态对 Amazon S3 存储桶中的 CSV 文件进行迭代。然后，您将其内容以及子工作流执行的 ARN 返回到另一个 Amazon S3 存储桶中。首先，在 Workflow Studio 中创建一个工作流原型。接下来，将 Map 状态的处理模式设置为“分布式”，将 CSV 文件指定为数据集，然后将其位置提供给该 Map 状态。您还可以为子工作流执行指定工作流类型，分布式 Map 状态以快速方式启动。

除了这些设置外，您还可以为本教程中使用的示例工作流指定其他配置，例如并发子工作流执行的最大数量和导出 Map 结果的位置。

内容

- 先决条件
- 第 1 步：创建工作流原型
- 第 2 步：配置 Map 状态的必填字段
- 第 3 步：配置其他选项
- 第 4 步：配置 Lambda 函数
- 第 5 步：更新工作流原型
- 第 6 步：查看自动生成的 Amazon States Language 定义并保存工作流

- 第 7 步：运行状态机

先决条件

- 将 CSV 文件上传到 Amazon S3 存储桶。您必须在 CSV 文件中定义标题行。有关对 CSV 文件的大小限制以及如何指定标题行的信息，请参阅 Amazon S3 存储桶中的 CSV 文件。
- 创建另一个 Amazon S3 存储桶，并在其中创建的一个文件夹，以便将 Map 状态结果导出到该存储桶中。

> **Important**
>
> 确保您的 Amazon S3 存储桶 AWS 账户 与状态机 AWS 区域 相同。

第 1 步：创建工作流原型

在此步骤中，您将使用 Workflow Studio 为工作流创建原型。Workflow Studio 是一款可视化工作流设计器，可在 Step Functions 控制台中使用。您可以分别从流和操作选项卡中选择所需的状态和 API 操作。您将使用 Workflow Studio 的拖放特征来创建工作流原型。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 从流选项卡中，将 Map 状态拖放到标有将第一个状态拖至此处的空白状态处。
5. 在配置选项卡下，在状态名称中输入 **`Process data`**。
6. 从操作选项卡中，将 AWS Lambda 调用 API 操作拖放到 Process data 状态中。
7. 将 AWS Lambda 调用状态重命名为 **`Process CSV data`**。

第 2 步：配置 Map 状态的必填字段

在此步骤中，您将配置分布式 Map 状态的以下必填字段：

- ItemReader— 指定数据集及其位置，该Map州可以从中读取输入。
- ItemProcessor – 指定以下值：

- `ProcessorConfig` – 将 `Mode` 和 `ExecutionType` 分别设置为 `DISTRIBUTED` 和 `EXPRESS`。这将为分布式 Map 状态启动的子工作流执行设置 Map 状态的处理模式和工作流类型。
- `StartAt` – Map 工作流中的第一个状态。
- `States` – 定义 Map 工作流程，这是在每个子工作流执行中要重复的一组步骤。

- ResultWriter— 指定 Step Functions 写入分布式地图状态结果的 Amazon S3 位置。

> **Important**
>
> 确保用于导出 Map Run 结果的 Amazon S3 存储桶 AWS 账户 与 AWS 区域 您的状态机相同。否则，您的状态机执行将因 `States.ResultWriterFailed` 错误而失败。

要配置必填句字段，请执行以下操作：

1. 选择 Process data 状态，然后在配置选项卡中执行以下操作：
   a. 对于处理模式，选择分布式。
   b. 对于项目来源，选择 Amazon S3，然后从 S3 项目来源下拉列表中选择 S3 中的 CSV 文件。
   c. 执行以下操作来指定 CSV 文件的 Amazon S3 位置：
      i. 对于 S3 对象，请从下拉列表中选择输入存储桶和密钥。
      ii. 对于存储桶，输入包含 CSV 文件的 Amazon S3 存储桶的名称。例如，**`sourceBucket`**。
      iii. 对于密钥，输入保存 CSV 文件的 Amazon S3 对象的名称。您还必须在此字段中指定 CSV 文件的名称。例如，**`csvDataset/ratings.csv`**。
   d. 对于 CSV 文件，还必须指定列标题的位置。为此，请选择其他配置，如果 CSV 文件的第一行是标题，对于 CSV 标题位置，请保留第一行的默认选择。否则，请选择给定以在状态机定义中指定标题。有关更多信息，请参阅 `ReaderConfig`。
   e. 对于子执行类型，请选择快速。
2. 在导出位置中，要将 Map Run 结果导出到特定的 Amazon S3 位置，请选择将 Map 状态的输出导出到 Amazon S3。
3. 执行以下操作：
   a. 对于 S3 存储桶，请从下拉列表中选择输入存储桶名称和前缀。
   b. 对于存储桶，输入要将结果导出到的 Amazon S3 存储桶的名称。例如，**`mapOutputs`**。

c. 对于前缀，输入要将结果保存到的文件夹名称。例如，**resultData**。

第 3 步：配置其他选项

除了分布式 Map 状态 所需的设置外，您还可以指定其他选项。这些选项可以包括子工作流并发执行的最大数量和导出 Map 状态结果的位置。

1. 选择 Process data 状态。然后，在项目来源中，选择其他配置。
2. 执行以下操作：

   a. 选择 “修改项目”， ItemSelector为每个子工作流程执行指定自定义 JSON 输入。

   b. 输入以下 JSON 输入：

   ```
   {
     "index.$": "$$.Map.Item.Index",
     "value.$": "$$.Map.Item.Value"
   }
   ```

   有关如何创建自定义输入的信息，请参阅 ItemSelector。
3. 在运行时设置中，对于并发限制，指定分布式 Map 状态 可以启动的并发子工作流执行数量。例如，输入 **100**。
4. 在浏览器中打开一个新窗口或选项卡，完成您将在此工作流中使用的 Lambda 函数的配置，如第 4 步：配置 Lambda 函数中所述。

第 4 步：配置 Lambda 函数

> **Important**
>
> 确保您的 Lambda 函数与状态机处于 AWS 区域 同一位置。

1. 打开 Lambda 控制台，然后选择创建函数。
2. 在创建函数页面上，选择从头开始创作。
3. 在基本信息部分中，配置您的 Lambda 函数：

   a. 对于 Function name（函数名称），请输入 **distributedMapLambda**。

b. 对于运行时系统，选择 Node.js 16.x。

c. 保留所有默认选项，然后选择创建函数。

d. 创建 Lambda 函数后，复制页面右上角显示的函数 Amazon 资源名称 (ARN)。您需要在工作流原型中提供此信息。要复制 ARN，请单击

。

以下是示例 ARN：

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

4. 复制以下 Lambda 函数的代码，然后将其粘贴到页面的distributedMapLambda代码源部分。

```
exports.handler = async function(event, context) {
    console.log("Received Input:\n", event);

    return {
        'statusCode' : 200,
        'inputReceived' : event //returns the input that it received
    }
};
```

5. 选择部署。函数部署后，选择测试，查看您的 Lambda 函数的输出。

第 5 步：更新工作流原型

在 Step Functions 控制台中，您将更新工作流，添加 Lambda 函数的 ARN。

1. 返回到创建工作流原型的选项卡或窗口。
2. 选择处理 CSV 数据步骤，然后在配置选项卡中执行以下操作：

   a. 对于集成类型，请选择已优化。

   b. 对于函数名称，输入 Lambda 函数名称。从出现的下拉列表中选择函数，或者选择输入函数名称并提供 Lambda 函数 ARN。

第 6 步：查看自动生成的 Amazon States Language 定义并保存工作流

当您将状态从操作和流选项卡拖放到画布上时，Workflow Studio 会自动实时撰写工作流的 Amazon States Language 定义。您可以根据需要编辑此定义。

1. （可选）在 Inspector 面板上选择定义，然后查看状态机定义。

> **Tip**
>
> 您也可以在 Workflow Studio 的代码编辑器中查看 ASL 的定义。在代码编辑器中，还可以编辑工作流的 ASL 定义。

以下示例代码显示了为您的工作流自动生成的 Amazon States Language 定义。

```
{
  "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. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **DistributedMapDemo**。

3. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机配置中的所有默认选项。

4. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 7 步：运行状态机

执行是状态机的一个实例，您可以在其中运行工作流来执行任务。

1. 在DistributedMapDemo页面上，选择开始执行。
2. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。
   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

   例如，选择 Map 状态，然后选择 Map Run，打开 Map Run 详细信息页面。在此页面上，您可以查看分布式 Map 状态的所有执行细节及其启动的子工作流执行。有关该页面的信息，请参阅检查 Map Run。

使用 Lambda 函数处理整批数据

在本教程中，您将使用分布式 Map 状态的 ItemBatcher 字段来处理 Lambda 函数中的整批项目。每批最多包含三个项目。分布式 Map 状态启动四个子工作流执行，其中每个执行处理三个项目，而一个执行处理单个项目。每个子工作流执行都会调用一个 Lambda 函数，该函数对批次中存在的各个项目进行迭代。

您将创建一个对整数数组执行乘法的状态机。假设输入的整数数组是 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`，乘法因子是 7。这些整数与乘法因子 7 相乘后形成的数组将是 `[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]`。

主题

- 第 1 步：创建状态机
- 第 2 步：创建 Lambda 函数
- 第 3 步：运行状态机

第 1 步：创建状态机

在此步骤中，您将创建状态机的工作流原型，该原型将整批数据传递给将在第 2 步中创建的 Lambda 函数。

- 使用以下定义通过 Step Functions 控制台创建状态机。有关创建状态机的信息，请参阅开始使用分布式 Map 状态教程中的第 1 步：创建工作流原型。

  在此状态机中，您可以定义一个分布式 Map 状态，该状态接受 10 个整数的数组作为输入，并以 3 个整数为一批将该数组传递给一个 Lambda 函数。Lambda 函数迭代批次中在的各个项目，并返回名为 `multiplied` 的输出数组。输出数组包含对输入数组中传递的项目执行乘法的结果。

  > **Important**
  >
  > 确保将以下代码中 Lambda 函数的 Amazon 资源名称 (ARN) 替换为您将在第 2 步中创建的函数的 ARN。

```
{
  "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"
```

```
    }
  }
}
```

第 2 步：创建 Lambda 函数

在此步骤中，您将创建 Lambda 函数，该函数可处理批次传递的所有项目。

> **Important**
>
> 确保您的 Lambda 函数与状态机处于 AWS 区域 同一位置。

创建 Lambda 函数

1. 使用 Lambda 控制台创建一个 Python 3.9 Lambda 函数，命名为 **ProcessEntireBatch**。有关创建 Lambda 函数的信息，请参阅开始使用分布式 Map 状态教程中的第 4 步：配置 Lambda 函数。
2. 复制以下 Lambda 函数代码，并将其粘贴到 Lambda 函数的代码源部分。

```
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. 创建 Lambda 函数后，复制显示在页面右上角的函数 ARN。要复制 ARN，请单击

   。

   以下是一个 ARN 示例，其中 *function-name* 是 Lambda 函数的名称（在本例中为 ProcessEntireBatch）：

```
arn:aws:lambda:us-east-1:123456789012:function:function-name
```

您需要在第 1 步中创建的状态机中提供函数 ARN。

4. 选择部署，以部署更改。

第 3 步：运行状态机

运行状态机时，分布式 Map 状态会启动四个子工作流程执行，其中每个执行处理三个项目，而一个执行处理单个项目。

以下示例显示了由其中一个子工作流执行传递给 ProcessEntireBatch 函数的数据。

```
{
  "BatchInput": {
    "MyMultiplicationFactor": 7
  },
  "Items": [1, 2, 3]
}
```

给定此输入，以下示例显示了 Lambda 函数返回的名为 multiplied 的输出数组。

```
{
  "statusCode": 200,
  "multiplied": [7, 14, 21]
}
```

状态机返回以下输出，其中包含为四个子工作流执行四个数组，名为 multiplied。这些数组包含各个输入项的乘法结果。

```
[
  {
    "statusCode": 200,
    "multiplied": [7, 14, 21]
  },
  {
    "statusCode": 200,
    "multiplied": [28, 35, 42]
  },
  {
    "statusCode": 200,
    "multiplied": [49, 56, 63]
```

```
  },
  {
    "statusCode": 200,
    "multiplied": [70]
  }
]
```

要将返回的所有数组项合并到一个输出数组中，可以使用 ResultSelector 字段。在分布式 Map 状态中定义此字段，用于查找所有 multiplied 数组，提取这些数组中的所有项目，然后将它们组合成一个输出数组。

要使用 ResultSelector 字段，请更新您的状态机定义，如以下示例所示。

```
{
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
      ...
      ...
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied[*]"
      }
    }
  }
}
```

更新的状态机返回统一的输出数组，如以下示例所示。

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

使用 Lambda 函数处理单个数据项

在本教程中，您将使用分布式 Map 状态的 ItemBatcher 字段，使用 Lambda 函数迭代批次中的单个项目。分布式 Map 状态将启动四个子工作流执行。每个子工作流都运行一个内联 Map 状态。内联 Map

状态每次迭代都会调用一个 Lambda 函数，并将批次中的单个项目传递给该函数。然后，Lambda 函数处理该项目并返回结果。

您将创建一个对整数数组执行乘法的状态机。假设输入的整数数组是 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`，乘法因子是 7。这些整数与乘法因子 7 相乘后形成的数组将是 `[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]`。

主题

- 第 1 步：创建状态机
- 第 2 步：创建 Lambda 函数
- 第 3 步：运行状态机

第 1 步：创建状态机

在此步骤中，您将创建状态机的工作流原型，该原型会将一批项目中的单个项目传递给您将在第 2 步中创建的 Lambda 函数的每次调用。

- 使用以下定义通过 Step Functions 控制台创建状态机。有关创建状态机的信息，请参阅开始使用分布式 Map 状态教程中的第 1 步：创建工作流原型。

  在此状态机中，您将定义一个分布式 Map 状态，该状态接受 10 个整数的数组作为输入，并将这些数组项分批传递给子工作流执行。每个子工作流程执行都会接收一批三个项目作为输入，并运行一个内联 Map 状态。内联 Map 状态的每次迭代都会调用 Lambda 函数，并将批次中的一个项目传递给该函数。然后，此函数将该项与系数 7 相乘并返回结果。

  每个子工作流执行的输出都是一个 JSON 数组，其中包含每个传递项的乘法结果。

  > **Important**
  >
  > 确保将以下代码中 Lambda 函数的 Amazon 资源名称 (ARN) 替换为您将在第 2 步中创建的函数的 ARN。

```
{
  "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"
      }
    }
  }
}
```

第 2 步：创建 Lambda 函数

在此步骤中，您将创建 Lambda 函数，用于处理批次传递的每个项目。

> **Important**
>
> 确保您的 Lambda 函数与状态机处于 AWS 区域 同一位置。

创建 Lambda 函数

1. 使用 Lambda 控制台创建一个 Python 3.9 Lambda 函数，命名为 **`ProcessSingleItem`**。有关创建 Lambda 函数的信息，请参阅开始使用分布式 Map 状态教程中的第 4 步：配置 Lambda 函数。

2. 复制以下 Lambda 函数代码，并将其粘贴到 Lambda 函数的代码源部分。

```
import json

def lambda_handler(event, context):

    multiplication_factor = event['MyMultiplicationFactor']
    item = event['MyItem']

    result = multiplication_factor * item

    return {
        'statusCode': 200,
        'multiplied': result
    }
```

3. 创建 Lambda 函数后，复制显示在页面右上角的函数 ARN。要复制 ARN，请单击

。

以下是一个 ARN 示例，其中 *function-name* 是 Lambda 函数的名称（在本例中为 `ProcessSingleItem`）：

```
arn:aws:lambda:us-east-1:123456789012:function:function-name
```

您需要在第 1 步中创建的状态机中提供函数 ARN。

4. 选择部署，以部署更改。

第 3 步：运行状态机

运行状态机时，分布式 Map 状态会启动四个子工作流程执行，其中每个执行处理三个项目，而一个执行处理单个项目。

以下示例显示了在子工作流执行中传递给其中一个 `ProcessSingleItem` 函数调用的数据。

```
{
  "MyMultiplicationFactor": 7,
  "MyItem": 1
}
```

给定此输入，以下示例显示了 Lambda 函数返回的输出。

```
{
  "statusCode": 200,
  "multiplied": 7
}
```

下面的示例显示了其中一个子工作流执行的输出 JSON 数组。

```
[
  {
    "statusCode": 200,
    "multiplied": 7
  },
  {
    "statusCode": 200,
    "multiplied": 14
  },
  {
    "statusCode": 200,
    "multiplied": 21
  }
]
```

状态机返回以下输出，其中包含四个子工作流执行的四个数组。这些数组包含各个输入项的乘法结果。

最后，状态机输出是一个名为 `multiplied` 的数组，组合了为四个子工作流执行返回的所有乘法结果。

```
[
  [
    {
      "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
    }
  ]
]
```

要将子工作流执行返回的所有乘法结果合并到单个输出数组中，可以使用 ResultSelector 字段。在分布式 Map 状态 中定义此字段以查找所有结果，提取单个结果，然后将它们组合成一个名为 `multiplied` 的输出数组。

要使用 `ResultSelector` 字段，请更新您的状态机定义，如以下示例所示。

```
{
```

```
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
      ...
      ...
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
        }
      },
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied"
      }
    }
  }
}
```

更新的状态机返回统一的输出数组，如以下示例所示。

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

启动状态机执行以响应 Amazon S3 事件

您可以执行一个 AWS Step Functions 状态机以响应 Amazon EventBridge 规则。

本教程演示如何将状态机配置作为 Amazon EventBridge 规则的目标。将文件添加至 Amazon Simple Storage Service (Amazon S3) 存储桶时，该规则将启动状态机执行。

对于实际应用程序，您可以启动状态机，该状态机会对添加到存储桶的文件执行操作，例如创建缩略图或者对图像和视频文件运行 Amazon Rekognition 分析。

在本教程中，您通过将文件上传至 Amazon S3 存储桶来启动一个 `Helloworld` 状态机执行。然后，您将查看该执行的示例输入，确定发送到 EventBridge 的 Amazon S3 事件通知的输入中包含的信息。

主题

- 先决条件：创建状态机
- 第 1 步：在 Amazon S3 中创建一个存储桶
- 第 2 步：使用 EventBridge 启用 Amazon S3 事件通知
- 第 3 步：创建 Amazon EventBridge 规则
- 第 4 步：测试规则
- 执行输入的示例

先决条件：创建状态机

在您将状态机配置为 Amazon EventBridge 目标前，必须创建状态机。

- 要创建基本状态机，请参阅创建使用 Lambda 函数的状态机教程。
- 如果您已有 `Helloworld` 状态机，请继续到下一个步骤。

第 1 步：在 Amazon S3 中创建一个存储桶

现在您已有 `Helloworld` 状态机，则需要创建一个用于存储文件的 Amazon S3 存储桶。在本教程中的第 3 步中，您将设置一个规则，这样当有文件上传到此存储桶时，EventBridge 会触发状态机执行。

1. 导航到 Amazon S3 控制台，然后选择创建存储桶，创建要在其中存储文件的存储桶并触发 Amazon S3 事件规则。
2. 输入存储桶名称，如 *username*`-sfn-tutorial`。

   > **Note**
   >
   > 存储桶名称在 Amazon S3 的所有 AWS 区域中的所有现有存储桶名称之间必须唯一。使用您自己的`###`以使此名称唯一。您必须在相同的 AWS 区域中创建所有资源。

3. 保留此页面上的所有默认选项，然后选择创建存储桶。

第 2 步：使用 EventBridge 启用 Amazon S3 事件通知

创建 Amazon S3 存储桶后，将其配置为在 S3 存储桶中发生某些事件（例如文件上传）时向 EventBridge 发送事件。

1. 导航到 Amazon S3 控制台。

2. 在存储桶列表中，请选择要为其启用事件的存储桶的名称。
3. 选择属性。
4. 向下滚动页面，查看活动通知部分，然后在Amazon EventBridge 子部分中选择编辑。
5. 在为此存储桶中的所有事件向 Amazon EventBridge 发送通知下方，请选择打开。
6. 选择保存更改。

> **Note**
>
> 在启用 EventBridge 后，所做的更改需要大约五分钟才能生效。

第 3 步：创建 Amazon EventBridge 规则

在您拥有状态机，创建 Amazon S3 存储桶并将其配置为向 EventBridge 发送事件通知后，创建一个 EventBridge 规则。

> **Note**
>
> 您必须在 Amazon S3 存储桶所在 AWS 区域中配置 EventBridge 规则。

创建规则

1. 导航至 Amazon EventBridge 控制台，选择创建规则。

> **Tip**
>
> 或者，在 EventBridge 控制台的导航窗格中，选择总线下的规则，然后选择创建规则。

2. 输入规则的名称（例如 *S3Step Functions*），然后根据需要输入规则的描述。
3. 对于事件总线和规则类型，请保留默认选择。
4. 选择下一步。这将打开构建事件模式页面。
5. 向下滚动到事件模式部分，然后执行以下操作：
   a. 对于事件源，保留默认选择：AWS 事件或 EventBridge 合作伙伴事件。
   b. 对于 AWS 服务，选择 Simple Storage Service (S3)。

c. 对于事件类型，选择 Amazon S3 事件通知。

d. 选择特定事件，然后选择已创建对象。

e. 选择按名称的特定存储桶，然后输入您在第 1 步中创建的存储桶名称 (*username*-sfn-tutorial)。

f. 选择下一步。这将打开选择目标页面。

创建目标

1. 在目标 1 中，保留 AWS 服务的默认选择。
2. 在选择目标下拉列表中，选择 Step Functions 状态机。
3. 在状态机列表中，选择您之前创建的状态机（例如 Helloworld）。
4. 保留页面上的所有默认选项，然后选择下一步。这将打开配置标签页面。
5. 再次选择下一步。这将打开查看并创建页面。
6. 查看规则详细信息并选择创建规则。

   此时将创建规则并显示规则页面，其中列出您的所有 Amazon EventBridge 规则。

第 4 步：测试规则

现在一切就绪，试验将文件添加到 Amazon S3 存储桶，然后查看生成的状态机执行的输入。

1. 将一个文件添加到 Amazon S3 存储桶中。

   导航到 Amazon S3 控制台，选择您创建的存储文件的存储桶 (*username*-sfn-tutorial)，然后选择上传。
2. 添加文件，例如 *test.png*，然后选择上传。

   这将启动一个状态机执行，并以输入的形式传递来自 AWS CloudTrail 的信息。
3. 检查您的状态机的执行情况。

   导航到 Step Functions 控制台，然后选择在您的 Amazon EventBridge 规则中使用的状态机 (Helloworld)。
4. 选择最近的状态机执行并展开执行输入部分。

   此输入包含诸如存储桶名称和对象名称之类的信息。在真实使用案例中，状态机可以使用此输入对该对象执行操作。

执行输入的示例

下面的示例展示了状态机执行的一个典型输入。

```
{
    "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"
    }
  }
```

使用 API Gateway 创建 Step Functions API

您可以使用 Amazon API Gateway 将您 AWS Step Functions 的 API 与 API Gateway API 中的方法相关联。在 HTTPS 请求发送到 API 方法后，API Gateway 将会调用您的 Step Functions API 操作。

本教程说明如何创建 API 来使用一个资源和 POST 方法与 StartExecution API 操作进行通信。您将使用 AWS Identity and Access Management (IAM) 控制台为 API Gateway 创建角色。然后，您将使用 API Gateway 控制台创建 API Gateway API，创建资源和方法，并将方法映射到 StartExecution API 操作。最后，您将部署和测试您的 API。

> **Note**
>
> 虽然 Amazon API Gateway 可以通过调用 `StartExecution` 启动 Step Functions 执行，但您必须调用 `DescribeExecution` 才能获得结果。

主题

- 第 1 步：为 API Gateway 创建 IAM 角色
- 第 2 步：创建 API Gateway API
- 第 3 步：测试和部署 API Gateway API

第 1 步：为 API Gateway 创建 IAM 角色

在创建您的 API Gateway API 之前，您需要向 API Gateway 授予调用 Step Functions API 操作的权限。

为 API Gateway 设置权限

1. 登录到 IAM 控制台，然后依次选择角色、创建角色。
2. 在选择受信任的实体页面上，请执行以下操作：
   a. 对于可信实体类型，保留 AWS 服务的默认选择。
   b. 对于使用案例，从下拉菜单中选择 API Gateway。
3. 选择 API Gateway，然后选择下一步。
4. 在添加权限页面上，选择下一步。
5. （可选）在名称、查看和创建页面上，输入角色名称等详细信息。例如，输入 **`APIGatewayToStepFunctions`**。
6. 选择创建角色。

   IAM 角色显示在角色列表中。
7. 选择角色的名称并记下角色 ARN，如以下示例所示。

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

将策略附加到 IAM 角色

1. 在角色页面上，搜索您的角色 (APIGatewayToStepFunctions)，然后选择该角色。
2. 在权限选项卡上，选择添加权限，然后选择附加策略。
3. 在附加策略页面上，搜索 AWSStepFunctionsFullAccess，选择该策略，然后选择添加权限。

第 2 步：创建 API Gateway API

在创建 IAM 角色后，您可以创建自定义 API Gateway API。

创建 API

1. 打开 Amazon API Gateway 控制台，然后选择创建 API。
2. 在选择 API 类型页面上的 REST API 窗格中，选择构建。
3. 在 “创建 REST API” 页面上，选择 “新建 API”，然后输入 ***StartExecutionAPI*** 作为 API 名称。
4. 将 API 端点类型保留为区域性，然后选择创建 API。

创建资源

1. 在 ***StartExecutionAPI*** 的资源页面上，选择创建资源。
2. 在创建资源页面上，输入 **execution** 作为资源名称，然后选择创建资源。

创建 POST 方法

1. 选择 /execution 资源，然后选择创建方法。
2. 对于方法类型，选择 POST。
3. 对于集成类型，选择 AWS 服务。
4. 对于 AWS 区域，从列表中选择一个区域。

> **Note**
>
> 有关当前支持 Step Functions 的区域，请参阅支持的区域。

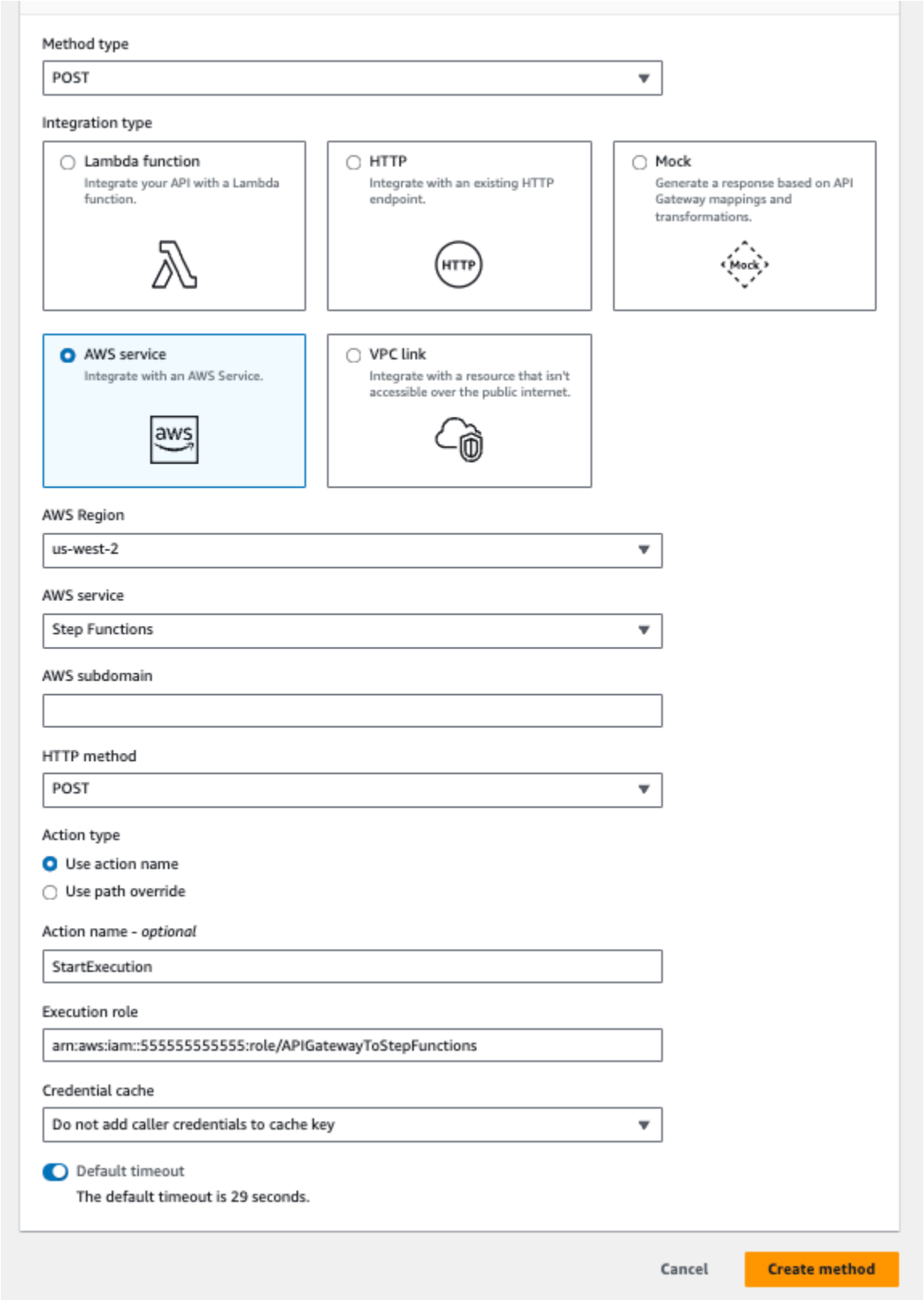
5. 对于 AWS 服务，从列表中选择 Step Functions。

6. 将 AWS 子域保留为空白。
7. 对于 HTTP 方法，从列表中选择 POST。

> **Note**
>
> 所有 Step Functions API 操作都使用 HTTP POST 方法。

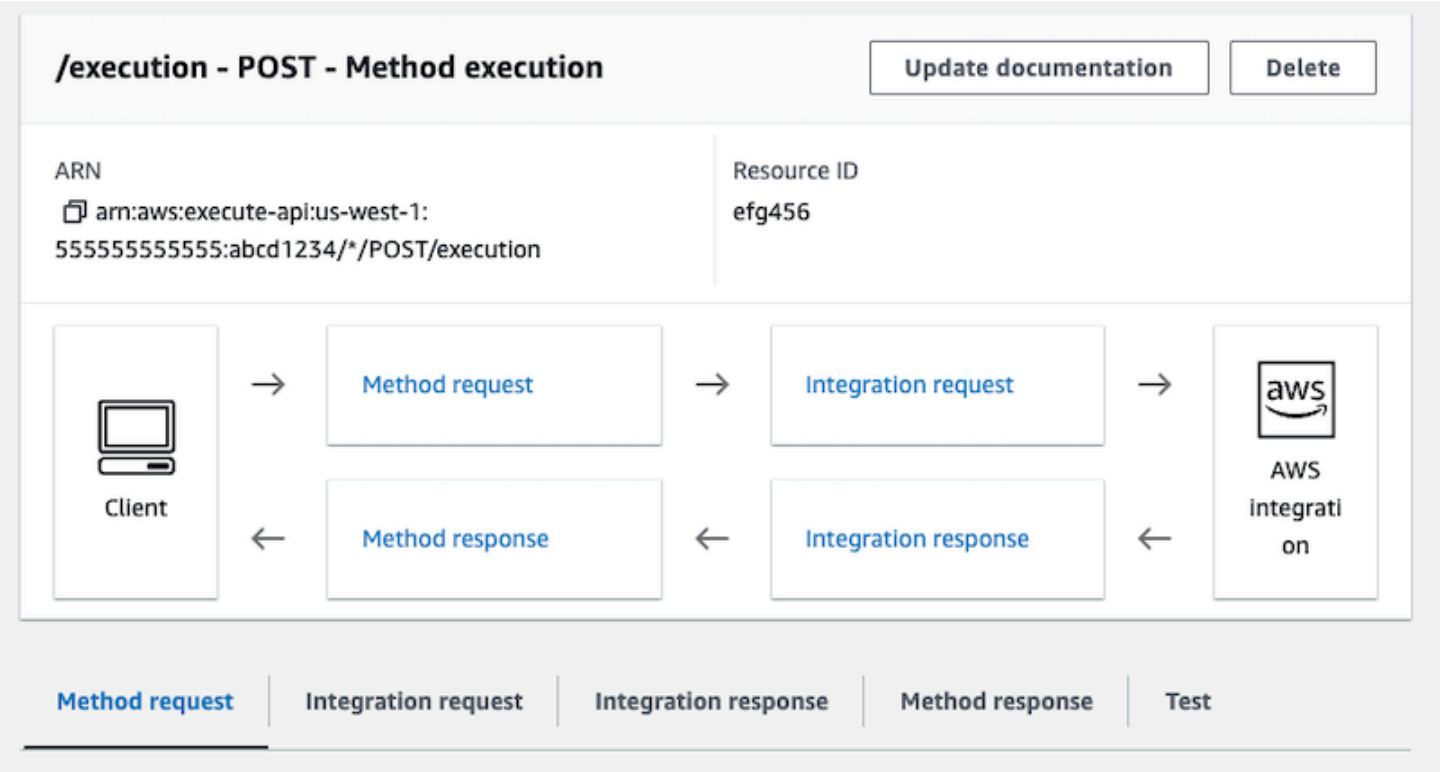
8. 对于操作类型，选择使用操作名称。
9. 对于操作名称，输入 **`StartExecution`**。
10. 对于执行角色，输入您之前创建的 IAM 角色的角色 ARN，如以下示例所示。

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



11. 对于凭证缓存和默认超时，保留默认选项，然后选择保存。

API Gateway 与 Step Functions 之间的可视映射显示在 /execution - POST - 方法执行页面上。



第 3 步：测试和部署 API Gateway API

创建 API 后，您可以测试和部署它。

测试 API Gateway 和 Step Functions 之间的通信

1. 在 /execution - POST - 方法执行页面上，选择测试选项卡。您可能需要选择右箭头按钮，以显示该选项卡。
2. 在 /execution - POST - 方法测试选项卡上，使用现有状态机的 ARN 将以下请求参数复制到请求正文部分（或者创建使用 Lambda 函数的新状态机），然后选择测试。

```
{
   "input": "{}",
   "name": "MyExecution",
   "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld"
}
```

有关更多信息，请参阅《AWS Step Functions API 参考》中的请求语法。

> **Note**
>
> 如果您不希望在 API Gateway 调用正文中包含状态机的 ARN，可以在集成请求选项卡中配置映射模板，如以下示例所示。
>
> ```
> {
>     "input": "$util.escapeJavaScript($input.json('$'))",
>     "stateMachineArn": "$util.escapeJavaScript($stageVariables.arn)"
> }
> ```
>
> 利用这种方法，可以根据您的开发阶段（例如 `dev`、`test` 和 `prod`）指定不同状态机的 ARN。有关在映射模板中指定阶段变量的更多信息，请参阅《API Gateway 开发者指南》中的 $stageVariables。

3. 执行将会启动，并且执行 ARN 及其纪元日期显示在响应正文下。

```
{
   "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution",
   "startDate": 1486768956.878
}
```

> **Note**
>
> 可以通过在 AWS Step Functions 控制台上选择您的状态机来查看执行。

部署 API

1. 在 ***StartExecutionAPI*** 的资源页面上，选择部署 API。
2. 对于阶段，选择新建阶段。
3. 对于阶段名称，输入 **alpha**。
4. （可选）对于描述，输入描述。
5. 选择部署。

测试部署

1. 在 ***StartExecutionAPI*** 的 “阶段” 页面上，展开 alpha、/、/exec ution、POST，然后选择 POST 方法。
2. 在方法覆盖下，选择复制图标以复制您 API 的调用 URL。完整 URL 应类似于以下示例。

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

3. 在命令行中，使用您状态机的 ARN 运行 `curl` 命令，然后调用您的部署的 URL，如以下示例所示。

```
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
```

将返回执行 ARN 及其纪元日期，如以下示例所示。

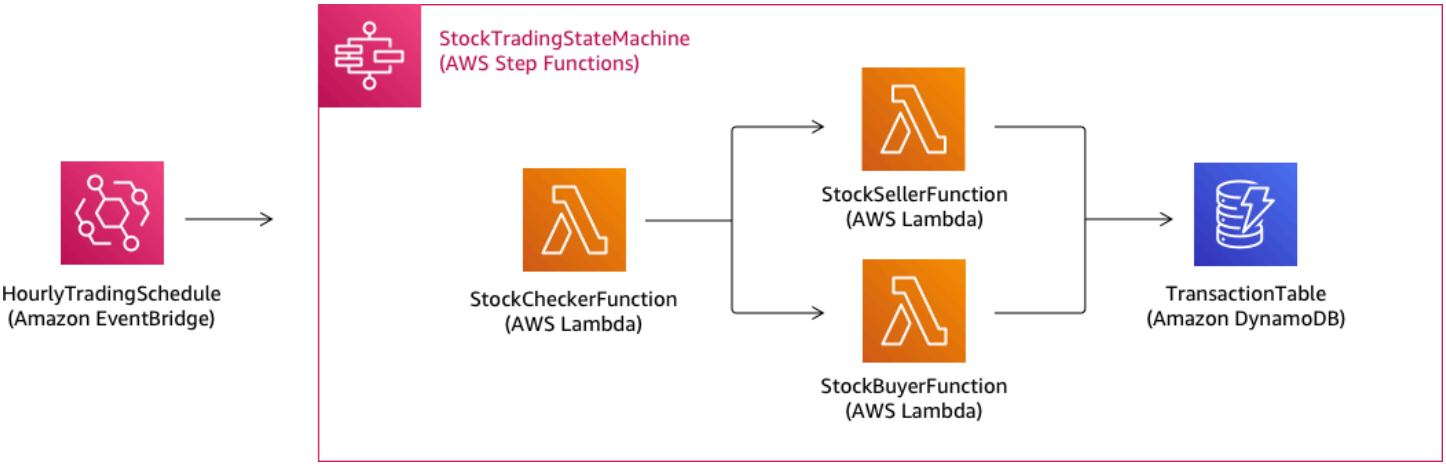
```
{"executionArn":"arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution","startDate":1.486772644911E9}
```

> **Note**
>
> 如果您收到“缺少身份验证令牌”错误，请确保调用 URL 以 /execution 结尾。

使用 AWS SAM 创建 Step Functions 状态机

在本指南中，您将下载、构建和部署包含 AWS Step Functions 状态机的示例 AWS SAM 应用程序。此应用程序创建一个模拟股票交易工作流程，该工作流程按预定义的时间表运行（请注意，该时间表默认情况下处于禁用状态以免产生费用）。

下图显示此应用程序的组件：



以下是为创建示例应用程序而运行的命令的预览。有关其中每个命令的更多详细信息，请参阅本页后面的各个部分

```
# 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
```

先决条件

本指南假定您已完成为操作系统安装 AWS SAM CLI 中的步骤。它假定您已经完成了以下操作：

1. 已创建 AWS 账户。
2. 已配置 IAM 权限
3. 已安装 Homebrew。注意：Homebrew 只是 Linux 和 macOS 的先决条件。
4. 已安装 AWS SAM CLI。注意：请确保您具有版本 0.52.0 或更高版本。您可以通过执行命令 `sam --version` 来检查您具有的版本。

第 1 步：下载示例 AWS SAM 应用程序

要运行的命令：

```
sam init
```

按照屏幕上的提示选择以下内容：

1. 模板：AWS 快速入门模板
2. 语言：Python、Ruby、NodeJS、Go、Java 或 .NET
3. 项目名称：（您选择的名称 - 默认为 sam-app）
4. 快速入门应用程序：多步骤工作流

AWS SAM 正在执行：

此命令使用您为“项目名称”提示符提供的名称创建一个目录（默认为 sam-app）。目录的具体内容将取决于您选择的语言。

以下是选择其中一个 Python 运行时时的目录内容：

```
### 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
```

您可以看到两个特别有趣的文件，：

- `template.yaml`：包含定义应用程序的 AWS 资源的 AWS SAM 模板。
- `statemachine/stockTrader.asl.json`：包含在Amazon States Language中编写的应用程序的状态机定义。

您可以在 `template.yaml` 文件中看到以下条目，该条目指向状态机定义文件：

```
    Properties:
      DefinitionUri: statemachine/stock_trader.asl.json
```

将状态机定义保存为单独的文件而不是将其嵌入到 AWS SAM 模板中会很有帮助。例如，如果您不将状态机定义嵌入到模板中，跟踪状态机定义的更改会更容易。您可以使用 Workflow Studio 创建和维护状态机定义，并将定义直接从控制台导出到 Amazon States Language 规范文件中，无需将其合并到模板中。

有关示例应用程序的更多信息，请参阅项目目录中的 `README.md` 文件。

第 2 步：构建应用程序

要运行的命令：

首先转到项目目录（也即，示例应用程序的 `template.yaml` 文件所在的目录；默认情况下是 `sam-app`），然后运行以下命令：

```
sam build
```

输出示例：

```
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 正在执行：

AWS SAM CLI 提供多个 Lambda 运行时的抽象，以构建您的依赖项，并将所有构建构件复制到暂存文件夹中，以便所有内容都准备好打包和部署。`sam build` 命令将构建您的应用程序具有的任何依赖项，并将构建构件复制到 `.aws-sam/build` 下的文件夹中。

第 3 步：将应用程序部署到 AWS 云

要运行的命令：

```
sam deploy --guided
```

按照屏幕上的提示操作。您只需通过 `Enter` 进行响应，即可接受交互式体验中提供的默认选项。

AWS SAM 正在执行：

此命令将您的应用程序部署到 AWS 云。它将采用您通过 `sam build` 命令构建的部署构件，将它们打包和上传到通过 AWS SAM CLI 创建的 Amazon S3 存储桶，然后使用 AWS CloudFormation 部署应用程序。在部署命令的输出中，您可以看到正在对 AWS CloudFormation 堆栈进行的更改。

您可以通过以下步骤验证示例 Step Functions 状态机是否已成功部署：

1. 登录 AWS Management Console 并通过以下网址打开 Step Functions 控制台：https://console.aws.amazon.com/states/。
2. 在左侧导航中，选择 State machines (状态机)。
3. 在列表中查找并选择您的新状态机。它将命名为 StockTradingStateMachine-*<unique-hash>*。
4. 选择 Definition (定义) 选项卡。

您现在应该看到状态机的可视化表示。您可以验证可视化表示是否与在项目目录的 `statemachine/stockTrader.asl.json` 文件中找到的状态机定义相匹配。

故障排除

SAM CLI 错误：“没有此类选项：--guided”

执行 `sam deploy` 时，您会看到以下错误：

```
Error: no such option: --guided
```

这意味着您使用的是不支持 `--guided` 参数的旧版本 AWS SAM CLI。要解决此问题，您可以将 AWS SAM CLI 版本更新为 0.33.0 或更高版本，也可以省略 `sam deploy` 命令中的 `--guided` 参数。

SAM CLI 错误：“无法创建托管资源：无法找到凭证”

执行 `sam deploy` 时，您会看到以下错误：

```
Error: Failed to create managed resources: Unable to locate credentials
```

这意味着，您尚未设置 AWS 凭证以使 AWS SAM CLI 能够进行 AWS 服务调用。要解决此问题，您必须设置 AWS 凭证。有关更多信息，请参阅《AWS Serverless Application Model 开发人员指南》中的设置 AWS 凭证。

清除

如果您不再需要通过运行本教程创建的 AWS 资源，则可以通过删除您部署的 AWS CloudFormation 堆栈来删除这些资源。

要使用 AWS Management Console删除使用本教程创建的 AWS CloudFormation 堆栈，请执行以下步骤：

1. 登录到 AWS Management Console 并打开 AWS CloudFormation 控制台 https://console.aws.amazon.com/cloudformation。
2. 在左侧导航窗格中，选择 Stacks (堆栈)。
3. 在堆栈列表中，选择 sam-app（或您创建的堆栈的名称）。
4. 选择删除。

完成后，堆栈的状态将更改为 DELETE_COMPLETE。

或者，您可以通过执行以下 AWS CLI 命令删除 AWS CloudFormation 堆栈：

```
aws cloudformation delete-stack --stack-name sam-app --region region
```

验证已删除堆栈

对于删除 AWS CloudFormation 堆栈的两种方法，您可以通过转到 https://console.aws.amazon.com/cloudformation，在左侧导航窗格中选择堆栈，然后在搜索文本框右侧的下拉列表中选择已删除来验证堆栈已删除。您应该在已删除堆栈的列表中看到您的堆栈名称 sam-app（或您创建的堆栈的名称）。

使用 Step Functions 创建活动状态机

本教程介绍如何使用 Java 和 AWS Step Functions创建基于活动的状态机。活动允许您控制在状态机中的其他位置运行的工作程序代码。有关概述，请参阅 Step Functions 的工作原理中的活动。

要完成本教程，您需要：

- 适用于 Java 的 开发工具包。本教程中的示例活动是一个 Java 应用程序，它使用与 AWS SDK for Java 进行通信 AWS。
- AWS 环境或标准 AWS 配置文件中的凭据。有关更多信息，请参阅《AWS SDK for Java 开发者指南》中的设置您的 AWS 凭证。

主题

- 第 1 步：创建活动
- 第 2 步：创建状态机
- 第 3 步：实现工作线程
- 第 4 步：运行状态机
- 第 5 步：运行和停止工作线程

第 1 步：创建活动

您必须让 Step Functions 知道某项活动，您希望创建该活动的工作线程（一个程序）。Step Functions 使用为活动建立身份的 Amazon 资源名称 (ARN) 作为响应。可以使用此身份协调状态机与工作线程之间传递的信息。

> **Important**
>
> 确保您的活动任务与状态机属于同一个 AWS 帐户。

1. 在 Step Functions 控制台左侧的导航窗格中，选择活动。
2. 选择创建活动。
3. 输入活动的名称，例如 *get-greeting*，然后选择创建活动。
4. 创建活动任务时，请记下其 ARN，如以下示例所示。

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

第 2 步：创建状态机

创建一个状态机，该状态机确定什么时候调用活动以及什么时候工作线程执行其主要工作，收集其结果并返回这些结果。要创建状态机，您需要使用 Workflow Studio 的代码编辑器。

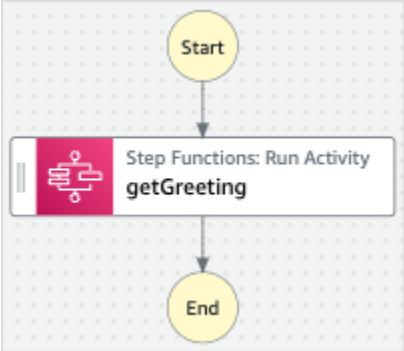
1. 在 Step Functions 控制台左侧的导航窗格中，选择状态机。
2. 在状态机页面上，选择创建状态机。
3. 在 选择模板对话框中，选择空白。
4. 选择选择。这将在设计模式中打开 Workflow Studio。
5. 在本教程中，您将在代码编辑器中编写状态机的 Amazon States Language (ASL) 定义。要执行此操作，请选择代码。
6. 删除现有的样板代码并粘贴以下代码。请记住将此代码中的示例 ARN 替换为您之前在 Resource 字段创建的活动任务的 ARN。

```
{
  "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
    }
  }
}
```

这是使用 (ASL) 的状态机的说明。它定义了名为 `getGreeting` 的单个 `Task` 状态。有关更多信息，请参阅状态机结构。

7. 在 图表可视化窗格 上，确保您添加的 ASL 定义的工作流图表与下图相似。



8. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **`ActivityStateMachine`**。

9. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

10. 在确认角色创建对话框中，选择确认继续。

    您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 3 步：实现工作线程

创建工作线程。工作线程是负责以下事务的程序：

- 使用 GetActivityTask API 操作在 Step Functions 中轮询活动。
- 使用您的代码执行活动的工作（例如，以下代码中的 getGreeting() 方法）。
- 使用 SendTaskSuccess、SendTaskFailure 和 SendTaskHeartbeat API 操作返回结果。

> **Note**
>
> 有关活动工作线程的更完整示例，请参阅使用 Ruby 编写的示例活动工作线程。该示例提供了一个基于最佳实操的实现，您可以参考它创建自己的活动工作线程。这段代码实现了消费者/生产者模型，并且允许对轮询器和活动工作线程的线程数进行配置。

实施工作线程

1. 创建一个名为 GreeterActivities.java的文件。
2. 将以下代码添加到其中。

```
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**
>
> 该示例中的 `EnvironmentVariableCredentialsProvider` 类假定已设置了 `AWS_ACCESS_KEY_ID`（或 `AWS_ACCESS_KEY`）和 `AWS_SECRET_KEY`（或 `AWS_SECRET_ACCESS_KEY`）环境变量。有关向工厂提供所需凭证的更多信息，请参阅AWSCredentialsProvider《AWS SDK for Java API 参考》和《AWS SDK for Java 开发人员指南》中的 “设置 AWS 证书和开发区域”。
> 默认情况下，AWS SDK 最多会等待 50 秒才能从服务器接收任何操作的数据。`GetActivityTask` 操作是一个长时间运行的轮询操作，对于下一个可用任务将等待多达 60 秒。为防止收到`SocketTimeoutException`错误，请设置 SocketTimeout 为 70 秒。

3. 在 `GetActivityTaskRequest().withActivityArn()` 构造函数的参数列表中，将 `ACTIVITY_ARN` 值替换为您之前创建的活动任务的 ARN。

第 4 步：运行状态机

开始执行状态机时，您的工作线程在 Step Functions 中轮询活动，执行其工作（使用您提供的输入）并返回其结果。

1. 在***`ActivityStateMachine`***页面上，选择 “开始执行”。

   随即显示启动执行对话框。

2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   b. 在输入框中，输入以下 JSON 输入，运行您的工作流。

   ```
   {
   ```

```
  "who": "AWS Step Functions"
}
```

c. 选择启动执行。

d. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

第 5 步：运行和停止工作线程

要让工作线程在您的状态机中轮询活动，必须运行工作线程。

1. 在命令行上，导航到您在其中创建 `GreeterActivities.java` 的目录。
2. 要使用 AWS SDK，请将`lib`和`third-party`目录的完整路径添加到构建文件的依赖项和 Java 中`CLASSPATH`。有关更多信息，请参阅《AWS SDK for Java 开发人员指南》中的下载和解压缩开发工具包。
3. 编译文件。

```
$ javac GreeterActivities.java
```

4. 运行文件。

```
$ java GreeterActivities
```

5. 在 Step Functions 控制台中，导航到执行详细信息页面。
6. 执行完成后，检查执行结果。
7. 停止工作线程。

使用 Lambda 迭代循环

在本教程中，将实现使用状态机和 AWS Lambda 函数迭代循环特定次数的设计模式。

无论何时您需要跟踪状态机中的循环数量，都可以使用此设计模式。此实现可帮助您将大型任务或长时间运行的执行分解为较小的块，或者在特定数量的事件之后结束执行。您可以使用类似的实现来定期结

束和重启长时间运行的执行，以避免超过 AWS Step Functions AWS Lambda、或其他 AWS 服务的服务配额。

在开始之前，请仔细阅读创建使用 Lambda 的 Step Functions 状态机教程，确保您熟悉如何同时使用 Lambda 和 Step Functions。

第 1 步：创建迭代计数的 Lambda 函数

通过使用 Lambda 函数，您可以跟踪状态机中的循环迭代次数。以下 Lambda 函数接收 `count`、`index` 和 `step` 的输入值。它返回这些值及更新的 `index` 和一个名为 `continue` 的布尔值。如果 `continue` 小于 `true`，Lambda 函数将 `index` 设置为 `count`。

然后，状态机实现 `Choice` 状态：在 `continue` 为 `true` 时执行一些应用程序逻辑，为 `false` 时，则退出。

创建 Lambda 函数

1. 登录 Lambda 控制台，然后选择创建函数。
2. 在创建函数页面上，选择从头开始创作。
3. 在基本信息部分中，配置您的 Lambda 函数：
   a. 对于函数名称，请输入 `Iterator`。
   b. 对于 Runtime (运行时)，选择 Node.js。
   c. 在更改默认执行角色中，选择创建具有基本 Lambda 权限的新角色。
   d. 选择创建函数。
4. 将以下 Lambda 函数的代码复制到代码源中。

```
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
  })
```

```
}
```

此代码接受 count、index 和 step 的输入值。它将 index 递增 step 的值，并返回这些值及布尔值 continue。如果 index 小于 count，continue 的值为 true。

5. 选择部署。

第 2 步：测试 Lambda 函数

使用数值运行您的 Lambda 函数，以查看其运行情况。您可以为 Lambda 函数提供模拟迭代的输入值。

测试 Lambda 函数

1. 选择测试。
2. 在 配置测试事件对话框中，在事件名称框中输入 TestIterator。
3. 使用以下内容替换示例数据。

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

这些值模拟在迭代期间来自状态机的内容。Lambda 函数将增加索引，并在索引小于count时返回true。count在此测试中，索引已增加到 5。测试将递增index到，6并设置continue为。true

4. 选择创建。
5. 选择 “测试” 以测试您的 Lambda 函数。

测试结果显示在执行结果选项卡中。

6. 要查看执行结果，请选择输出选项卡。

```
{
  "index": 6,
  "step": 1,
```

```
  "count": 10,
  "continue": true
}
```

> **Note**
>
> 如果设置index为9并再次测试，则index增量为10、continue将为。false

第 3 步：创建状态机

> **在你离开 Lambda 控制台之前...**
>
> 复制 Lambda 函数 ARN。将其粘贴到便条中。下一步中需要使用该值。

接下来，您将创建一个具有以下状态的状态机：

- ConfigureCount— 设置countindex、和的默认值step。
- Iterator— 指您之前创建的 Lambda 函数，传入中配置的值。ConfigureCount
- IsCountReached— 一种根据Iterator函数返回的值继续循环或继续进入Done状态的选择状态。
- ExampleWork— 需要完成的工作的存根。在此示例中，工作流程有一个Pass状态，但在实际解决方案中，您可能会使用Task。
- Done— 工作流程的结束状态。

要在控制台中创建状态机，请执行以下操作：

1. 打开 Step Functions 控制台，然后选择创建状态机。

   > **Important**
   >
   > 您的状态机必须与您的 Lambda 函数位于同一个 AWS 账户和区域中。

2. 选择 “空白” 模板。
3. 在 “代码” 窗格中，粘贴以下定义状态机的 JSON。

   有关 Amazon States Language 的更多信息，请参阅状态机结构。

```
{
    "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. 将该`Iterator Resource`字段替换为您之前创建的 Lambd `Iterator` a 函数的 ARN。
5. 选择 Config，然后输入状态机的名称，例如*`IterateCount`*。

> **Note**
>
> 状态机、执行和活动任务的名称长度不得超过 80 个字符。对于您的账户和 AWS 地区，这些名称必须是唯一的，并且不得包含以下任何内容：
>
> - 空格
> - 通配符 (? *)
> - 方括号字符 (< > { } [ ])
> - 特殊字符 (" # % \ ^ | ~ ` $ & , ; : /)
> - 控制字符 (\\u0000 - \\u001f 或 \\u007f - \\u009f)
>
> 如果您的状态机是快速类型，则可以为状态机的多个执行提供相同的名称。即使多个执行的名称相同，Step Functions 也会为每个快速状态机执行生成唯一的执行 ARN。
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

6. 对于 “类型”，接受默认值 “标准”。在 “权限” 中，选择 “创建新角色”。
7. 选择 “创建”，然后选择 “确认角色创建”。

第 4 步：启动新的执行

在创建您的状态机后，可以开始执行。

1. 在IterateCount页面上，选择 “开始执行”。
2. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

3. 选择启动执行。

   此时将开始状态机的新执行，并显示正在运行的执行。
   状态机图表视图，以蓝色显示 Iterator 状态以表示正在进行状态。

   执行逐步递增，使用您的 Lambda 函数跟踪计数。在每次迭代时，它都会执行在状态机中处于 `ExampleWork` 状态引用的示例工作。

   计数达到您的状态机中 `ConfigureCount` 状态中指定的数值后，执行停止迭代并结束。
   状态机图表视图，以绿色显示 “迭代器” 状态和 “完成” 状态，表示两者均已成功。

将长时间运行的工作流执行作为新执行继续执行

AWS Step Functions 旨在运行持续时间和步骤数量有限的工作流程。执行的最长持续时间为一年，最多为 2.5 万个事件（请参阅配额）。

对于长时间运行的执行，为了避免在执行事件历史记录中达到 2.5 万个条目的硬性限额条件，我们建议您直接从状态机的 `Task` 状态开始新的工作流执行。可借此将工作流分解为较小的状态机，并在新执行中继续正在进行的工作。要启动这些工作流执行，请从 `Task`状 态调用 `StartExecution` API 操作并传递必要的参数。

您也可以实现另外一种模式，即使用 Lambda 函数启动状态机的新执行，将正在进行的工作分散到多个工作流执行中。

本教程将向您展示在不超过服务限额的情况下继续执行工作流的两种方法。

主题

- 使用 Step Functions API 操作继续新的执行（推荐）
- 使用 Lambda 函数继续新的执行

使用 Step Functions API 操作继续新的执行（推荐）

Step Functions 可以通过调用自己的 API 以作为集成服务来启动工作流执行。我们建议您使用这种方法来避免长时间运行的执行超出服务限额。

第 1 步：创建一个长时间运行的状态机

创建一个长时间运行的状态机，您希望该状态机从另一个状态机的 Task 状态开始。在本教程中，将使用使用 Lambda 函数的状态机。

> **Note**
>
> 请务必将此状态机的名称和 Amazon 资源名称复制到文本文件中以备稍后使用。

第 2 步：创建状态机以调用 Step Functions API 操作

从一个 **Task** 状态启动工作流执行

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 从 “操作” 选项卡中，将 StartExecutionAPI 操作拖放到标有 “将第一个状态拖到此处” 的空白状态。
5. 选择StartExecution状态并在的 “配置” 选项卡中执行以下操作设计模式：

   a. 将状态重命名为 **Start nested execution**。

   b. 对于集成类型，从下拉列表中选择 AWS 开发工具包 - 新建。

   c. 在 API 参数 中，执行以下操作：

      i. 对于 StateMachineArn，请将示例 Amazon 资源名称替换为您的状态机 ARN。例如，输入使用 Lambda 的状态机的 ARN。

      ii. 对于 Input 节点，将现有占位符文本替换为以下值：

      ```
      "Comment": "Starting workflow execution using a Step Functions API action"
      ```

      iii. 确保 API 参数中的输入类似于以下内容：

```
{
  "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. （可选）在 Inspector 面板上选择定义，查看工作流中自动生成的 Amazon States Language (ASL) 定义。

> **Tip**
>
> 您也可以在 Workflow Studio 的代码编辑器中查看 ASL 的定义。在代码编辑器中，还可以编辑工作流的 ASL 定义。

7. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **ParentStateMachine**。

8. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

9. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 3 步：更新 IAM 策略

要确保您的状态机有权限启动使用 Lambda 函数的状态机执行，您需要将内联策略附加到状态机的 IAM 角色。有关更多信息，请参阅《IAM 用户指南》中的嵌入内联策略。

1. 在该ParentStateMachine页面上，选择 IAM 角色 ARN 以导航到状态机的 IAM 角色页面。
2. 为的 IAM 角色分配适当的权限，使其能够开始执行另一台状态机。ParentStateMachine要添加该权限，请执行以下操作：

   a. 在 IAM 角色页面上，选择添加权限，然后选择创建内联策略。

   b. 在创建策略页面上，选择 JSON 选项卡。

   c. 使用以下策略替换现有文本：

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "states:StartExecution"
          ],
          "Resource": [
              "arn:aws:states:us-
east-2:123456789012:stateMachine:LambdaStateMachine"
          ]
      }
  ]
}
```

   d. 选择查看策略。

   e. 指定策略名称，然后选择创建策略。

第 4 步：运行状态机

状态机执行是指运行工作流执行任务的实例。

1. 在ParentStateMachine页面上，选择开始执行。

   随即显示启动执行对话框。

2. 在启动执行对话框中，执行以下操作：

a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

b. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

c. 选择启动执行。

d. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

3. 打开LambdaStateMachine页面，注意由触发的新执行ParentStateMachine。

使用 Lambda 函数继续新的执行

您可以创建一个使用 Lambda 函数的状态机，用于在当前执行终止之前启动新的执行。使用这种方法在新执行中继续您正在进行的工作，可以凭借状态机将大型作业拆分成较小的工作流程，或使您的状态机无限期运行。

本教程背后的理念是，使用外部 Lambda 函数修改您的工作流程，详情请见使用 Lambda 迭代循环教程。您将使用相同的 Lambda 函数 (`Iterator`) 将循环迭代特定次数。此外，您将创建另一个 Lambda 函数来启动工作流程的新执行，并在每次启动新执行之后将计数减一。在输入中设置执行次数后，此状态机将结束并重新开始执行达指定的次数。

您将要创建的状态机可实施以下状态。

状态	用途
`ConfigureCount`	配置了 `count`、`index` 和 `step` 值的 `Pass` 状态，`Iterator` Lambda 函数使用这些值来分步执行工作的迭代。

状态	用途
Iterator	引用 Task Lambda 函数的 Iterator 状态。
IsCountReached	Choice 状态，使用 Iterator 函数的布尔值来决定状态机是否应继续示例工作，或移动到 ShouldRestart 状态。
ExampleWork	Pass 状态，表示 Task 状态将在实际实施中执行工作。
ShouldRestart	Choice 状态，使用 executionCount 值来决定是否应结束一个执行并启动另一个，或直接结束。
Restart	Task 状态，使用 Lambda 函数来启动状态机的新执行。与 Iterator 函数一样，此函数的计数也会递减。Restart 状态将计数的递减值传递给新执行的输入。

先决条件

在开始之前，请仔细阅读创建使用 Lambda 的 Step Functions 状态机教程，确保您熟悉如何同时使用 Lambda 和 Step Functions。

主题

- 第 1 步：创建迭代计数的 Lambda 函数
- 第 2 步：创建一个 Restart Lambda 函数，启动新的 Step Functions 执行
- 第 3 步：创建状态机
- 第 4 步：更新 IAM 策略
- 第 5 步：运行状态机

第 1 步：创建迭代计数的 Lambda 函数

> **Note**
>
> 如果您已完成使用 Lambda 迭代循环教程，可以跳过此步骤并使用该 Lambda 函数。

本节和使用 Lambda 迭代循环教程展示了如何使用 Lambda 函数来跟踪计数，例如，状态机中循环的迭代次数。

以下 Lambda 函数接收 count、index 和 step 的输入值。它返回这些值及更新的 index 和一个名为 continue 的布尔值。如果 continue 小于 true，Lambda 函数将 index 设置为 count。

然后，状态机实施 Choice 状态：在 continue 为 true 时执行一些应用程序逻辑，或在 continue 为 false 时继续执行 ShouldRestart。

创建迭代 Lambda 函数

1. 打开 Lambda 控制台，然后选择创建函数。
2. 在创建函数页面上，选择从头开始创作。
3. 在基本信息部分中，配置您的 Lambda 函数：

   a. 对于 Function name（函数名称），请输入 Iterator。

   b. 对于运行时系统，选择 Node.js 16.x。

   c. 保留页面上的所有默认选项，然后选择创建函数。

   创建 Lambda 函数后，记下函数位于页面右上角的 Amazon 资源名称 (ARN)，例如：

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

4. 将以下 Lambda 函数代码复制到 Lambda 控制台 ***Iterator*** 页面的代码源部分中。

   ```
   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
     })
   }
   ```

   此代码接受 count、index 和 step 的输入值。它将 index 递增 step 的值，并返回这些值及布尔值 continue。如果 index 小于 count，continue 的值为 true。

5. 选择部署，部署代码。

测试迭代 Lambda 函数

使用数字值运行 Iterate 函数，查看它的工作情况。您可以为模拟迭代的 Lambda 函数提供输入值，以查看使用特定输入值得到的输出。

测试 Lambda 函数

1. 在配置测试事件对话框中，选择创建新测试事件，然后键入 TestIterator 作为事件名称。
2. 使用以下内容替换示例数据。

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

这些值模拟在迭代期间来自状态机的内容。Lambda 函数递增索引并将 continue 返回为 true。当索引不小于 count 时，它会将 continue 返回为 false。在此测试中，索引已增加到 5。结果应将 index 增加到 6，并将 continue 设置为 true。

3. 选择 创建。
4. 在 Lambda 控制台的 “###” 页面上，TestIterator确保已列出，然后选择 “测试”。

测试结果将显示在页面顶部。选择详细信息并查看结果。

```
{
  "index": 6,
  "step": 1,
  "count": 10,
  "continue": true
}
```

> **Note**
>
> 如果在此测试中将 index 设置为 9，则 index 增加到 10，continue 为 false。

第 2 步：创建一个 Restart Lambda 函数，启动新的 Step Functions 执行

1. 打开 Lambda 控制台，然后选择创建函数。
2. 在创建函数页面上，选择从头开始创作。
3. 在基本信息部分中，配置您的 Lambda 函数：
   a. 对于 Function name（函数名称），请输入 `Restart`。
   b. 对于运行时系统，选择 Node.js 16.x。
4. 保留页面上的所有默认选项，然后选择创建函数。

   创建 Lambda 函数后，记下函数位于页面右上角的 Amazon 资源名称 (ARN)，例如：

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

5. 将以下 Lambda 函数代码复制到 Lambda 控制台 ***Restart*** 页面的代码源部分中。

   以下代码可将执行次数的计数递减，并启动状态机的新执行，包括已递减的值。

   ```
   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. 选择部署，部署代码。

第 3 步：创建状态机

现在，您已创建了两个 Lambda 函数，可以创建状态机。在此状态机中，ShouldRestart 和 Restart 状态表示如何将工作拆分为多个执行。

Example ShouldRestart 选择状态

下面的示例显示 ShouldRestart Choice 状态。此状态决定是否应重新开始执行。

```
"ShouldRestart": {
"Type": "Choice",
"Choices": [
  {
    "Variable": "$.restart.executionCount",
    "NumericGreaterThan": 1,
    "Next": "Restart"
  }
],
```

初始执行的输入中包含 $.restart.executionCount 值。每次调用 Restart 函数后它会减 1，并置入每个后续执行的输入中。

Example Restart Task 状态

下面的示例显示 Restart Task 状态。此状态使用您之前创建的 Lambda 函数重新启动执行，并递减计数来跟踪剩余的执行启动次数。

```
"Restart": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Restart",
  "Next": "Done"
},
```

创建状态机

1. 打开 Step Functions 控制台，然后选择创建状态机。

> **Important**
>
> 确保您的状态机与您之前在步骤 1 和步骤 2 中创建的 Lambda 函数位于相同的 AWS 账户和区域下。

2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 在本教程中，您将在代码编辑器中编写状态机的 Amazon States Language (ASL) 定义。要执行此操作，请选择代码。
5. 删除现有的样板代码并粘贴以下代码。请记住将此代码中的 ARN 替换为您创建的 Lambda 函数的 ARN。

```
{
    "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. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **ContinueAsNew**。

7. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，请保留状态机设置中的所有默认选项。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

8. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

9. 将这个状态机的 Amazon 资源名称 (ARN) 保存在一个文本文件中。您需要提供 ARN，同时向 Lambda 函数提供权限才能启动新的 Step Functions 执行。

第 4 步：更新 IAM 策略

要确保您的 Lambda 函数有启动 Step Functions 新执行的权限，请为您的 `Restart` Lambda 函数所用的 IAM 角色附加内联策略。有关更多信息，请参阅《IAM 用户指南》中的嵌入内联策略。

> **Note**
>
> 您可以更新上个示例的 `Resource` 行，引用 `ContinueAsNew` 状态机的 ARN。这会限制该策略，使它只能启动特定状态机的执行。

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": [
             "states:StartExecution"
         ],
         "Resource": "arn:aws:states:us-east-2:123456789012stateMachine:ContinueAsNew"
     }
 ]
}
```

第 5 步：运行状态机

要开始执行，请在输入中包含状态机的 ARN，并提供 `executionCount` 以规定启动新执行的次数。

1. 在ContinueAsNew页面上，选择开始执行。

随即显示启动执行对话框。

2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   b. 在输入框中，输入以下 JSON 输入，运行您的工作流。

   ```
   {
     "restart": {
       "StateMachineArn": "arn:aws:states:us-
   east-1:123456789012:stateMachine:ContinueAsNew",
       "executionCount": 4
     }
   }
   ```

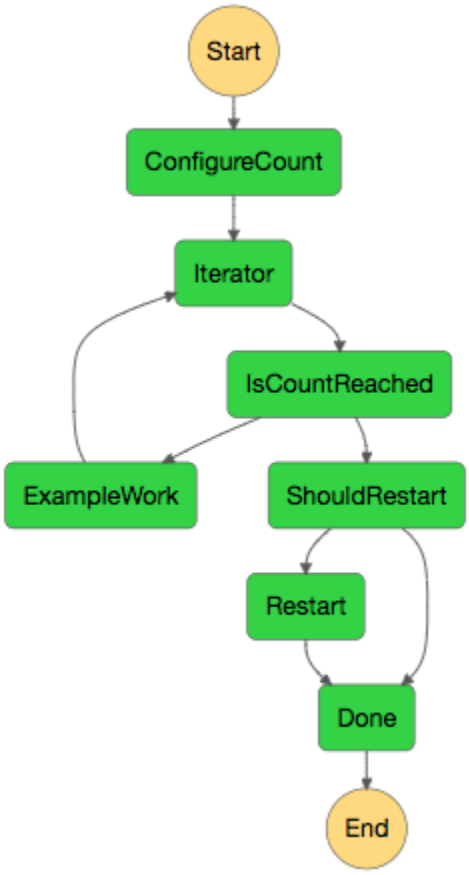
   c. 使用您的 `ContinueAsNew` 状态机的 ARN 更新 `StateMachineArn` 字段。

   d. 选择启动执行。

   e. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

      图标视图显示四个执行中的第一个。在完成之前，它将传递 `Restart` 状态并启动新执行。



此执行完成后，您可以查看下一个正在运行的执行。选择顶部的ContinueAsNew链接以查看执行列表。您应该可以看到最近关闭的执行，以及 `Restart` Lambda 函数启动的正在进行的执行。

所有执行全部完成后，您应在列表中看到四个成功完成的执行。启动的首个执行显示您所选的名称，后续执行的名称是自动生成的。

部署示例人员批准项目

本教程介绍如何部署允许 AWS Step Functions 执行在任务期间暂停并等待用户对电子邮件做出响应的人员批准项目。一旦用户批准任务继续，工作流程就会进入下一个状态。

部署本教程中包含的 AWS CloudFormation 堆栈将创建所有必要的资源，包括：

- Amazon API Gateway 资源
- 一个 AWS Lambda 函数
- 一台 AWS Step Functions 状态机
- Amazon Simple Notification Service 电子邮件主题
- 相关 AWS Identity and Access Management 角色和权限

> **Note**
>
> 在创建 AWS CloudFormation 堆栈时，您需要提供一个可以访问的有效电子邮件地址。

有关更多信息，请参阅《AWS CloudFormation 用户指南》中的使用 CloudFormation 模板和`AWS::StepFunctions::StateMachine`资源。

主题

- 步骤 1：创建 AWS CloudFormation 模板
- 第 2 步：创建一个堆栈
- 第 3 步：批准 Amazon SNS 订阅

- 第 4 步：运行状态机
- AWS CloudFormation 模板源代码

步骤 1：创建 AWS CloudFormation 模板

1. 从 AWS CloudFormation 模板源代码 部分复制示例代码。

2. 将 AWS CloudFormation 模板的源文件粘贴到本地计算机上的文件中。

   在此示例中，文件名为 `human-approval.yaml`。

第 2 步：创建一个堆栈

1. 登录 AWS CloudFormation 控制台。
2. 选择创建堆栈，然后选择使用新资源（标准）。
3. 在创建堆栈页面上，执行以下操作：

   a. 在先决条件 - 准备模板部分，确保选中模板已就绪。

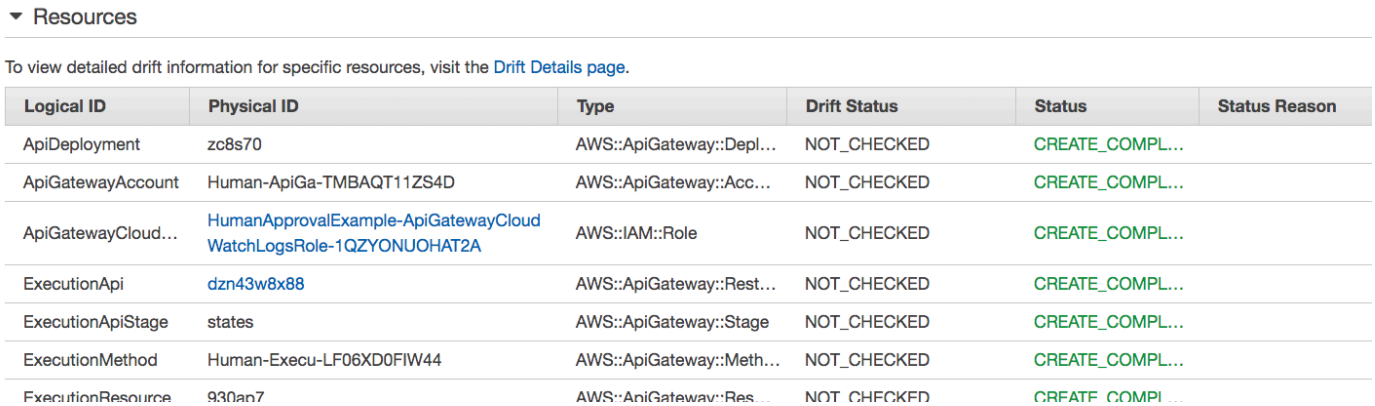
   b. 在指定模板部分，选择上传模板文件，然后选择选择文件，上传您之前创建的包含模板源代码的 `human-approval.yaml` 文件。
4. 选择下一步。
5. 在指定堆栈详细信息页面中，请执行以下操作：

   a. 在堆栈名称中，输入堆栈的名称。

   b. 在参数下，输入一个有效的电子邮件地址。您将使用此电子邮件地址订阅 Amazon SNS 主题。
6. 选择下一步，然后再次选择下一步。
7. 在查看页面上，选择我确认 AWS CloudFormation 可能会创建 IAM 资源，然后选择创建。

   AWS CloudFormation 开始创建您的堆栈并显示 CREATE_IN_ PROGRESS 状态。该过程完成后， AWS CloudFormation 将显示 “创建_完成” 状态。

8. （可选）要显示您的堆栈中的资源，请选择堆栈，然后选择资源选项卡。

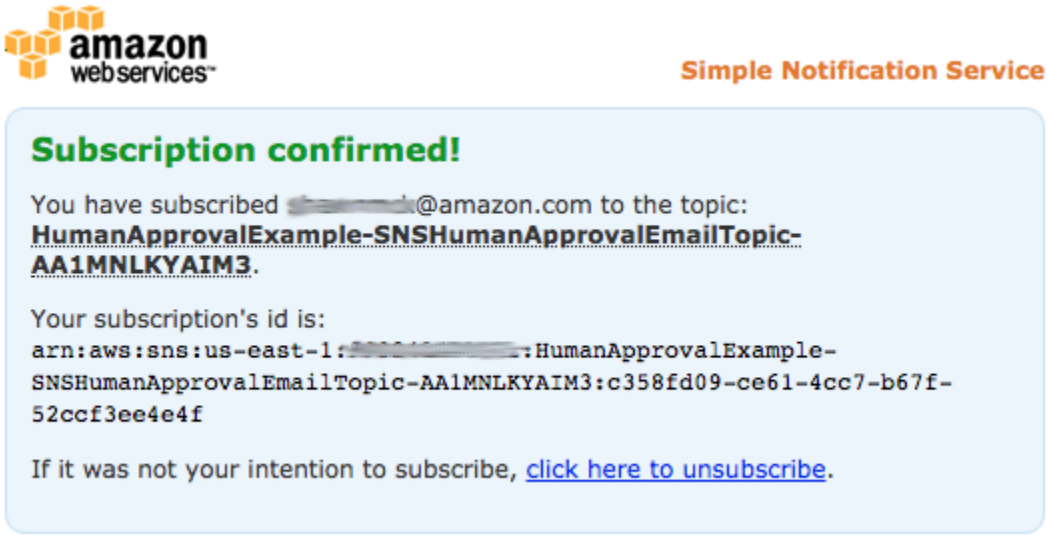


第 3 步：批准 Amazon SNS 订阅

创建 Amazon SNS 主题后，您将收到一封电子邮件，要求您确认订阅。

1. 打开您在创建 AWS CloudFormation 堆栈时提供的电子邮件帐户。
2. 打开来自 no-reply@sns.amazonaws.com 的 AWS 通知 – 订阅确认电子邮件。

   该电子邮件将列出 Amazon SNS 主题的 Amazon 资源名称，以及确认链接。
3. 选择确认订阅链接。



第 4 步：运行状态机

1. 在HumanApprovalLambdaStateMachine页面上，选择开始执行。

随即显示启动执行对话框。

2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   b. 在输入框中，输入以下 JSON 输入，运行您的工作流。

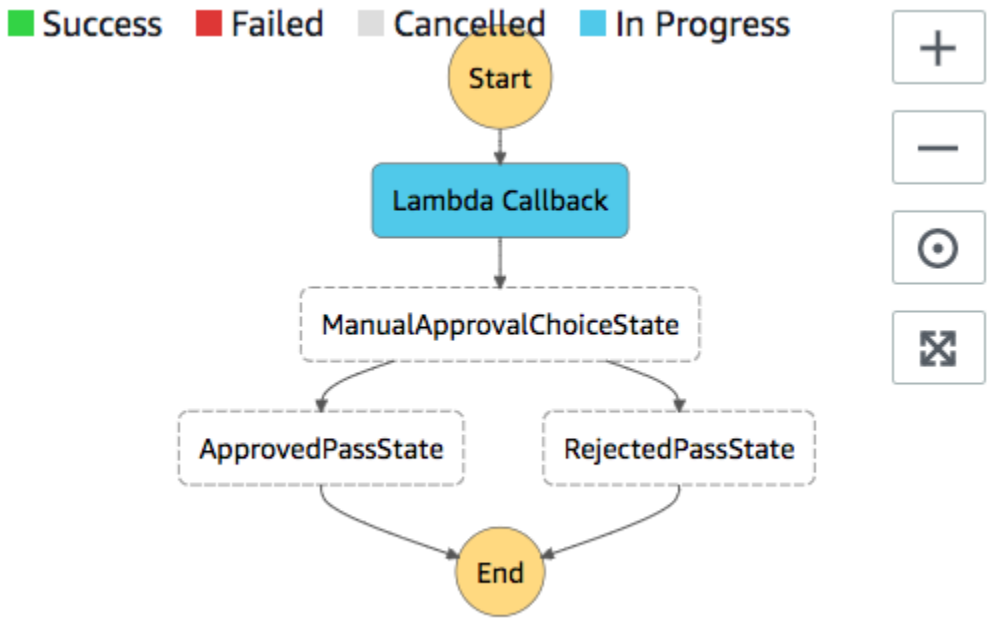
   ```
   {
       "Comment": "Testing the human approval tutorial."
   }
   ```

   c. 选择启动执行。

   ApprovalTest状态机开始执行，并在 Lambda 回调任务处暂停。

   d. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

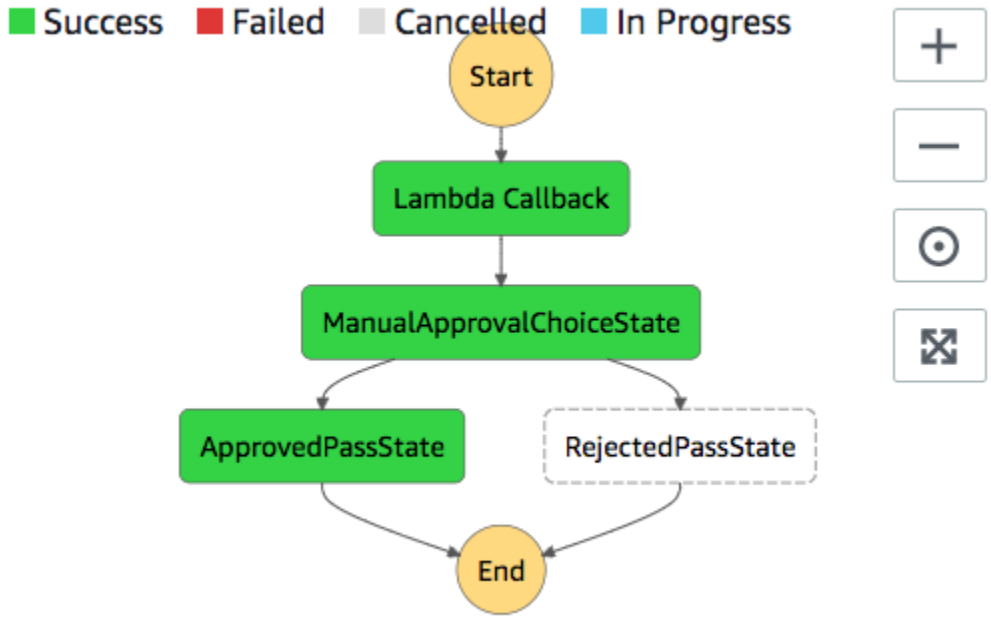


3. 在您之前用于 Amazon SNS 主题的电子邮件账户中，打开主题为 “需要审批” 的邮件。 AWS Step Functions

   此消息包括用于批准和拒绝的独立 URL。

4. 选择批准 URL。

   工作流程根据您的选择继续。



AWS CloudFormation 模板源代码

使用此 AWS CloudFormation 模板来部署人工批准流程工作流的示例。

```
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
```

在 Step Functions 中查看 X-Ray 跟踪

在本教程中，您将学习如何使用 X-Ray 来跟踪运行状态机时发生的错误。您可以使用 AWS X-Ray 可视化状态机的组件、确定性能瓶颈以及对导致错误的请求进行故障排除。在本教程中，您将创建多个随机生成错误的 Lambda 函数，然后可以使用 X-Ray 对其进行跟踪和分析。

创建使用 Lambda 的 Step Functions 状态机教程将引导您创建一个调用 Lambda 函数的状态机。如果您已完成该教程，请跳至第 2 步，并使用之前创建的 AWS Identity and Access Management (IAM) 角色。

主题

- 第 1 步：为 Lambda 创建 IAM 角色。
- 第 2 步：创建 Lambda 函数
- 第 3 步：再创建两个 Lambda 函数
- 第 4 步：创建状态机
- 第 5 步：运行状态机

第 1 步：为 Lambda 创建 IAM 角色。

AWS Lambda 和 AWS Step Functions 都可以执行代码和访问 AWS 资源（例如，存储在 Amazon S3 存储桶中的数据）。为了保持安全性，您必须授予 Lambda 和 Step Functions 对这些资源的访问权。

Lambda 要求您在创建 Lambda 函数时分配一个 AWS Identity and Access Management (IAM) 角色，就像 Step Functions 要求您在创建状态机时分配 IAM 角色一样。

您可使用 IAM 控制台创建服务相关角色。

创建角色（控制台）

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择角色。然后选择创建角色。
3. 选择 AWS 服务角色类型，然后选择 Lambda。
4. 选择 Lambda 使用案例。使用案例由服务定义以包含服务所需的信任策略。然后选择下一步：权限。
5. 选择一个或多个要附加到角色的许可策略（例如，`AWSLambdaBasicExecutionRole`）。请参阅 AWS Lambda 权限模型。

   选中用于分配您希望角色拥有的权限的策略旁的框，然后选择下一步: 审核。
6. 输入角色名称。
7. （可选）对于角色描述，编辑新服务相关角色的描述。
8. 检查该角色，然后选择创建角色。

第 2 步：创建 Lambda 函数

您的 Lambda 函数将随机抛出错误或超时，生成要在 X-Ray 中查看的示例数据。

> **Important**
>
> 确保您的 Lambda 函数与状态机位于同一个 AWS 账户和 AWS 区域下。

1. 打开 Lambda 控制台，然后选择创建函数。
2. 在创建函数部分中，选择从头开始创作。
3. 在基本信息部分中，配置您的 Lambda 函数：
   a. 对于 Function name（函数名称），请输入 `TestFunction1`。
   b. 对于运行时系统，选择 Node.js 18.x。
   c. 对于角色，选择选择现有角色。
   d. 对于现有角色，选择您之前创建的 Lambda 角色。

> **Note**
>
> 如果您创建的 IAM 角色未显示在列表中，该角色可能仍需要几分钟才能传播到 Lambda。

e. 选择创建函数。

创建您的 Lambda 函数后，记下页面右上角的 Amazon 资源名称 (ARN)。例如：

```
arn:aws:lambda:us-east-1:123456789012:function:TestFunction1
```

4. 将以下 Lambda 函数的代码复制到第 ***TestFunction1*** 页的函数代码部分。

```
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 }
};
```

此代码会创建随机定时的故障，这些故障将用于在状态机中生成示例错误，可使用 X-Ray 跟踪进行查看和分析。

5. 选择保存。

第 3 步：再创建两个 Lambda 函数

再创建两个 Lambda 函数。

1. 重复第 2 步，再创建两个 Lambda 函数。对于第二个函数，在函数名称中输入 TestFunction2。对于第三个函数，在函数名称中输入 TestFunction3。

2. 在 Lambda 控制台中，检查您现在是否有三个 Lambda 函数：TestFunction1、TestFunction2 和 TestFunction3。

第 4 步：创建状态机

在本步骤中，您将使用 Step Functions 控制台创建具有三个 Task 状态的状态机。每个 Task 状态都将引用您的三个 Lambda 函数中的一个。

1. 打开 Step Functions 控制台，然后选择创建状态机。

> **Important**
>
> 确保您的状态机与您之前在步骤 2 和步骤 3 中创建的 Lambda 函数位于相同的 AWS 账户和区域下。

2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 在本教程中，您将在代码编辑器中编写状态机的 Amazon States Language (ASL) 定义。要执行此操作，请选择代码。
5. 删除现有的样板代码并粘贴以下代码。在 Task 状态定义中，请记住将示例 ARN 替换为您创建的 Lambda 函数的 ARN。

```
{
  "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"
    }
  }
}
```

这是使用 Amazon States Language 的状态机的说明。它定义了三个名为 `CallTestFunction1`、`CallTestFunction2` 和 `CallTestFunction3` 的 `Task` 状态。每个状态都会调用您的三个 Lambda 函数中的一个。有关更多信息，请参阅状态机结构。

6. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **`TraceFunctions`**。

7. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   在本教程中，在其他配置下，选择启用 X-Ray 跟踪。保留状态机设置中的所有其他默认选项。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

8. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

> **Note**
>
> 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

第 5 步：运行状态机

状态机执行是指运行工作流执行任务的实例。

1. 在***`TraceFunctions`***页面上，选择 “开始执行”。

   此时将显示新执行页面。

2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

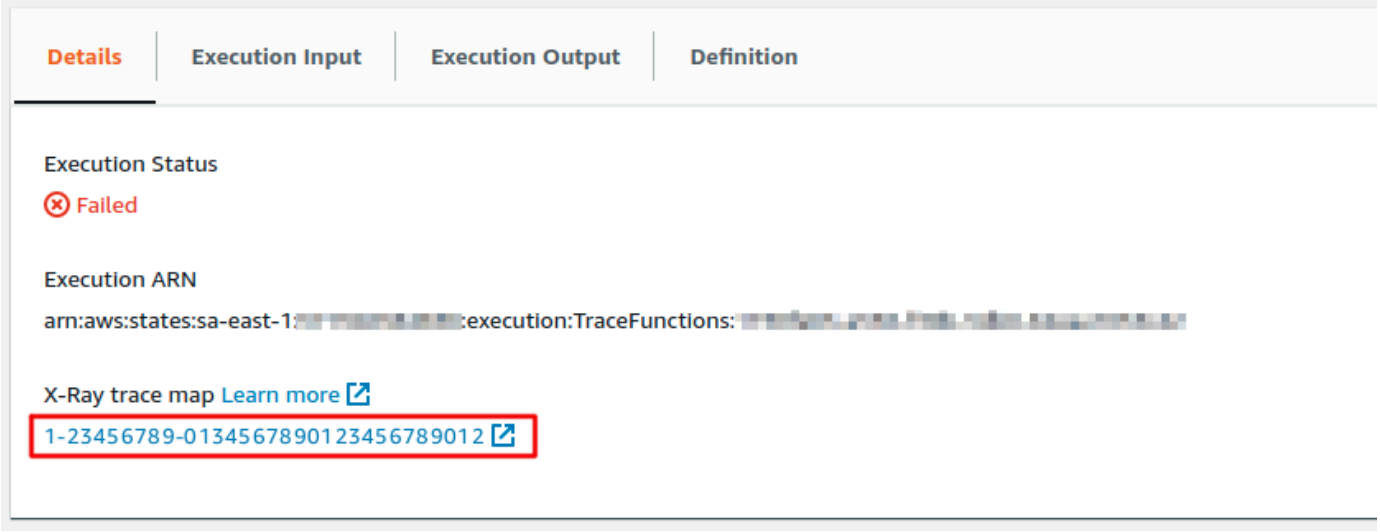
b. 选择启动执行。

c. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

运行几次（至少三次）执行。

3. 执行完成后，点击 X-Ray 追踪映射链接。您可以在执行仍在运行时查看跟踪，但您可能希望在查看 X-Ray 跟踪映射之前先查看执行结果。



4. 查看服务图来识别错误发生的位置、具有高延迟的连接或针对不成功请求的跟踪。在此示例中，您可以看到每个函数接收了多少流量。`TestFunction2` 被调用的频率高于 `TestFunction3`，`TestFunction1` 被调用的频率是 `TestFunction2` 的两倍多。

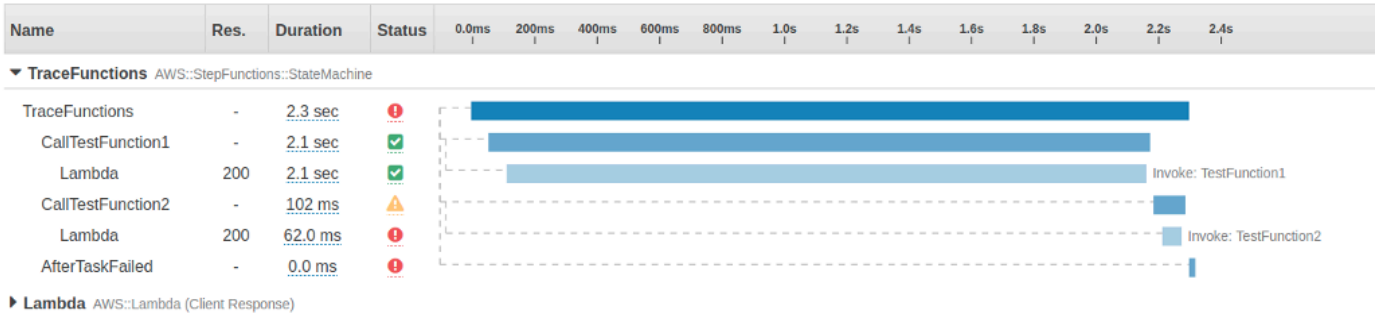
服务地图根据成功调用与错误和故障的比率为每个节点显示颜色，从而指示节点的运行状况：

- 绿色表示成功调用
- 红色表示服务器故障（500 系列错误）

- 黄色表示客户端错误（400 系列错误）
- 紫色表示限制错误（429 请求过多）

您也可以选择一个服务节点来查看该节点的请求，或选择两个节点之间的边缘来查看经过该连接的请求。

5. 查看 X-Ray 跟踪映射，了解每次执行时发生了什么。时间线视图显示分段和子分段的层次结构。列表中的第一个条目为分段，表示服务为单个请求记录的所有数据。分段的下一级是子分段。此示例显示了 Lambda 函数记录的子分段。



有关理解 X-Ray 跟踪以及使用 X-Ray 和 Step Functions 的更多信息，请参阅 AWS X-Ray 和 Step Functions

使用 AWS 软件开发工具包服务集成收集 Amazon S3 存储桶信息

本教程向您展示了如何使用 Amazon Simple Storage Service 执行 AWS 开发工具包集成。您在本教程中创建的状态机可以收集有关您的 Amazon S3 存储桶的信息，然后列出存储桶以及当前区域中每个存储桶的版本信息。

主题

- 第 1 步：创建状态机
- 第 2 步：添加必要的 IAM 角色权限
- 第 3 步：启动一个标准状态机执行
- 第 4 步：运行一个快速状态机执行

第 1 步：创建状态机

使用 Step Functions 控制台，您将创建一个包含 Task 状态的状态机，用于列出当前账户和区域中的所有 Amazon S3 存储桶。然后，您将添加另一个调用 HeadBucket API 的 Task 状态，用于验证返回的存储桶是否可在当前区域访问。如果无法访问存储桶，HeadBucket API 调用将返回 S3.S3Exception 错误。您将包括一个用于捕获此异常的 Catch 块和一个作为后备状态的 Pass 状态。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在 选择模板对话框中，选择空白。
3. 选择选择。这将在设计模式中打开 Workflow Studio。
4. 在本教程中，您将在代码编辑器中编写状态机的 Amazon States Language (ASL) 定义。要执行此操作，请选择代码。
5. 删除现有的样板代码并粘贴以下状态机定义。

```
{
  "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. 为状态机指定一个名称。为此，请选择默认状态机名称旁边的编辑图标MyStateMachine。然后，找到状态机配置，在状态机名称框中指定一个名称。

   对于本教程，请输入名称 **`Gather-S3-Bucket-Info-Standard`**。

7. （可选）在状态机配置中，指定其他工作流设置，例如状态机类型及其执行角色。

   保留状态机设置中的所有默认选项。

   如果您之前为状态机创建了具有正确权限的 IAM 角色并想使用该角色，请在权限中选择选择现有角色，然后从列表中选择一个角色。或者选择输入角色 ARN，然后为该 IAM 角色的 ARN 获取该角色。

8. 在确认角色创建对话框中，选择确认继续。

   您也可以选择查看角色设置，返回至状态机配置。

   > **Note**
   >
   > 如果您删除了 Step Functions 创建的 IAM 角色，Step Functions 在以后无法重新创建该角色。同样，如果您修改了该角色（例如，通过在 IAM 策略中从主体中删除 Step Functions），Step Functions 在以后也无法还原其原始设置。

   在第 2 步 中，您将向状态机角色添加缺少的权限。

第 2 步：添加必要的 IAM 角色权限

要收集有关您当前区域的 Amazon S3 存储桶的信息，您必须向状态机提供访问 Amazon S3 存储桶所需的权限。

1. 在状态机页面上，选择 IAM 角色 ARN，打开状态机角色的角色页面。
2. 选择添加权限，然后选择创建内联策略。
3. 选择 JSON 选项卡，然后将以下权限粘贴到 JSON 编辑器中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*"
        }
    ]
}
```

4. 选择查看策略。
5. 在查看策略下，为策略名称输入 **s3-bucket-permissions**。
6. 选择创建策略。

第 3 步：启动一个标准状态机执行

1. 在 Gather-S3-Bucket-Info-Standard 页面上，选择启动执行。
2. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   b. 选择启动执行。

   c. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

第 4 步：运行一个快速状态机执行

1. 使用第 1 步中提供的状态机定义创建一个快速状态机。请确保您还包括第 2 步 中所述的必要的 IAM 角色权限。

   > **Tip**
   >
   > 要与之前创建的标准状态机区分开来，请将快速状态机命名为 **`Gather-S3-Bucket-Info-Express`**。

2. 在 Gather-S3-Bucket-Info-Standard 页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   a. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   b. 选择启动执行。

   c. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

开发人员工具

以下资源包含有关构建无服务器工作流和使用状态机的其他信息：

- AWS CDK
- AWS 适用于 VS Code 的工具包

以下主题包含学习如何创建、测试和调试状态机的信息。

主题

- 开发选项
- AWS Step Functions 和 AWS SAM
- 使用 Application Composer 中的 Workflow Studio
- 使用 AWS CloudFormation为 Step Functions 创建一个 Lambda 状态机
- 使用 AWS CDK 为 Step Functions 创建一个 Lambda 状态机
- 使用同步 Express 状态机创建 API Gateway REST API 使用 AWS CDK
- AWS Step Functions 适用于 Python 的数据科学软件开发工具包
- 使用 Terraform 部署状态机

开发选项

您可以通过多种方式实现 AWS Step Functions 状态机，例如使用控制台、软件开发工具包或本地版本的 Step Functions 进行测试和开发。

主题

- Step Functions 控制台
- AWS 软件开发工具包
- 标准和快速工作流
- HTTPS 服务 API
- 开发环境
- 端点
- AWS CLI

- Step Functions Local
- AWS Toolkit for Visual Studio Code
- AWS Serverless Application Model 和 Step Functions
- Terraform 和 Step Functions
- 定义格式支持

Step Functions 控制台

您可以使用 Step Functions 控制台定义状态机。通过使用为任务提供代码，可以在云中编写复杂的状态机，而无需使用 AWS Lambda 本地开发环境。编写完成后，您可以使用 Step Functions 控制台使用 Amazon States Language 定义状态机。

创建 Lambda 状态机教程使用此方法创建简单的状态机、执行该状态机并查看其结果。

数据流模拟器

您可以在 Step Functions 控制台中设计、实施和调试工作流。您还可以使用 `JsonPath` 输入和输出处理来控制工作流中的数据流。使用 Step Functions 控制台中的数据流模拟器来了解信息是如何从一个状态流向另一个状态的，并了解如何筛选和操作数据。此工具模拟 Step Functions 用于处理数据的每个字段，例如`InputPath`、`Parameters`、`ResultSelector`、`OutputPath` 和 `ResultPath`。

有关更多信息，请参阅 数据流模拟器。

AWS 软件开发工具包

适用于 Java、.NET、Ruby、PHP、Python（Boto 3）、Go 和 C++ 的 AWS 软件开发工具包支持 Step F JavaScript unctions。这些开发工具包提供了在各种编程语言中使用 Step Functions HTTPS API 操作的便捷方式。

您可以利用这些开发工具包库提供的 API 操作开发状态机、活动或状态机启动器。您还可以使用这些库访问可见性操作，以开发您自己的 Step Functions 监控和报告工具。

要将 Step Functions 与其他 AWS 服务一起使用，请参阅适用于亚马逊 Web AWS Services 的当前软件开发工具包和工具的参考文档。

> **Note**
>
> Step Functions 仅支持 HTTPS 端点。

标准和快速工作流

创建新的状态机时，您必须在 Type 处选择标准或快速。在这两种情况下，都可以使用 Amazon States Language 定义状态机。状态机的执行方式将有所不同，具体取决于您选择的类型。创建状态机后，无法更改您选择的类型。

请参阅使用 CloudWatch Logs 进行日志记录了解更多信息。

HTTPS 服务 API

Step Functions 提供可通过 HTTPS 请求访问的服务操作。您可以使用这些操作直接与 Step Functions 通信，还可以使用任何可通过 HTTPS 与 Step Functions 通信的语言来开发自己的库。

您可以使用服务 API 操作开发状态机、工作线程或状态机启动器。您还可以通过 API 操作访问可见性操作，以开发您自己的监控和报告工具。

有关 API 操作的详细信息，请参阅 AWS Step Functions API 参考。

开发环境

您必须设置与计划使用的编程语言兼容的开发环境。

例如，要使用 Java 为 Step Functions 进行开发，必须在每个开发工作站上安装 Java 开发环境 AWS SDK for Java，例如。如果您使用 Eclipse IDE for Java Developers，则还应安装 AWS Toolkit for Eclipse。此 Eclipse 插件提供了在 AWS开发中非常有用的功能。

如果您的编程语言需要运行时环境，则必须在运行这些进程的每台计算机上设置该环境。

端点

为了减少延迟并将数据存储在符合您要求的位置，Step Functions 提供了不同 AWS 区域的终端节点。

Step Functions 中的每个端点都完全独立。一个状态机或活动仅存在于创建它的区域中。您在一个区域中创建的任意状态机或活动，不会与在其他区域中创建的状态机或活动分享任何数据或属性。例如，您可以在两个不同区域中注册名为 STATES-Flows-1 的状态机。一个区域中的 STATES-Flows-1 状态机不会与另一个区域的 STATES-Flow-1 状态机共享数据或属性。

有关 Step Functions 端点的列表，请参阅《AWS 一般参考》中的AWS Step Functions 区域和端点。

AWS CLI

您可以从 AWS Command Line Interface (AWS CLI) 访问许多 Step Functions 功能。可以替代使用 St AWS CLI ep Functions 控制台，或者在某些情况下，也可以替代使用 Step Functions API 操作进行编程。例如，您可以使用 AWS CLI 创建状态机，然后列出您现有的状态机。

您可以在 AWS CLI 中使用 Step Functions 命令来启动和管理执行、轮询活动、记录任务检测信号等。有关 Step Functions 命令的完整列表、可用参数的说明以及展示它们使用方法的示例，请参阅 AWS CLI 命令参考。

AWS CLI 命令严格遵循亚马逊州立大学的语言，因此您可以使用 AWS CLI 来了解 Step Functions API 操作。您还可以使用现有 API 知识创建代码原型，或者从命令行执行 Step Functions 操作。

Step Functions Local

要进行测试和开发，您可以在本地计算机上安装并运行 Step Functions。使用 Step Functions Local，您可以在任何计算机上启动执行。

本地版本的 Step Functions 可以在本地运行 AWS 和运行时调用 AWS Lambda 函数。您还可以协调其他支持的 AWS 服务。有关更多信息，请参阅 在本地测试状态机。

> **Note**
>
> Step Functions Local 使用虚拟账户来工作。

AWS Toolkit for Visual Studio Code

您可以使用 VS Code 与远程状态机交互，并在本地开发状态机。您可以创建或更新状态机、列出现有状态机、执行或下载这些状态机。VS Code 还允许您从模板创建新的状态机，查看状态机的可视化效果，并提供代码片段、代码完成和代码验证。

有关更多信息，请参阅《 AWS Toolkit for Visual Studio Code 用户指南》

AWS Serverless Application Model 和 Step Functions

Step Functions 与集成 AWS Serverless Application Model，允许您将工作流程与 Lambda 函数、API 和事件集成在一起，以创建无服务器应用程序。

您也可以将 AWS SAM CLI 与结合使用 AWS Toolkit for Visual Studio Code ，作为集成体验的一部分。

有关更多信息，请参阅 AWS Step Functions 和 AWS SAM。

Terraform 和 Step Functions

Terraform by HashiCorp 是一个使用基础设施即代码 (IaC) 构建应用程序的框架。借助 Terraform，您可以创建状态机并使用特征，例如预览基础架构部署和创建可重复使用的模板。Terraform 模板通过将代码分解为多个较小的块来帮助您维护和重用代码。

有关更多信息，请参阅 使用 Terraform 部署状态机。

定义格式支持

Step Functions 提供了多种工具，可让您以不同的格式提供状态机定义。指定状态机详细信息的 Amazon States Language (ASL) 定义可以作为字符串提供，也可以使用 JSON 或 YAML 作为序列化对象提供。

> **Note**
>
> YAML 允许使用单行注释。模板的状态机定义部分中提供的任何 YAML 注释都不会延续到所创建资源的定义中。相反，您可以在状态机定义中使用 `Comment` 属性。有关更多信息，请参阅状态机结构页面。

下表显示了哪些工具支持基于 ASL 的定义。

各工具支持定义格式

	JSON	YAML	Stringified Amazon States Language		
Step Functions 控制台	✓				
HTTPS 服务 API			✓		
AWS CLI			✓		

	JSON	YAML	Stringified Amazon States Language		
Step Functions Local			✓		
AWS Toolkit for Visual Studio Code	✓	✓			
AWS SAM	✓	✓			
AWS CloudFormation	✓	✓	✓		

> **Note**
>
> AWS CloudFormation AWS SAM 还允许您将状态机定义以 JSON 或 YAML 格式上传到 Amazon S3，并在模板中提供定义的 Amazon S3 位置。当状态机定义很复杂时，这可以提高模板的可读性。欲了解更多信息，请参阅 AWS::StepFunctions::StateMachine S3Location 页面。

以下示例 AWS CloudFormation 模板展示了如何使用不同的输入格式提供相同的状态机定义。

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 和 AWS SAM

AWS Toolkit for Visual Studio Code 作为集成体验的一部分，您可以将 AWS SAM CLI 与结合使用，在本地创建状态机。您可以使用 AWS SAM构建无服务器应用程序，然后在 VS Code IDE 中构建状态机。然后，您可以验证、打包和部署您的资源。或者，您也可以发布到 AWS Serverless Application Repository。

> **Tip**
>
> 要部署一个用于启动 Step Functions 工作流程的示例无服务器应用程序 AWS 账户，请参阅 The Worksho AWS Step Functions p 的模块 11-使用 AWS SAM AWS SAM部署。

主题

- 为什么要将 Step Functions 配合使用 AWS SAM？
- Step Functions 与 AWS SAM 规范集成
- Step Functions 与 SAM CLI 集成
- DefinitionSubstitutions 在AWS SAM模板中
- 后续步骤

为什么要将 Step Functions 配合使用 AWS SAM？

当你将 Step Functions 与配合使用时， AWS SAM 你可以：

- 开始使用 AWS SAM 示例模板。
- 将状态机构建到无服务器应用程序中。
- 使用变量替换在部署时将 ARN 替换到状态机中。

  AWS CloudFormation 支持 `DefinitionSubstitutions`，它们可让您将工作流定义中的动态引用添加到您在 CloudFormation 模板中提供的值。您可以使用 `${dollar_sign_brace}` 表示法将替换项添加到工作流定义中，从而添加动态引用。您还需要在CloudFormation模板中 StateMachine 资源的`DefinitionSubstitutions`属性中定义这些动态引用。在 CloudFormation 堆栈创建流程中，替换项将替换为实际值。有关更多信息，请参阅 DefinitionSubstitutions 在AWS SAM模板中。
- 使用 AWS SAM 策略模板指定状态机的角色。
- 使用 API Gateway、 EventBridge 事件或按 AWS SAM 模板中的计划启动状态机执行。

Step Functions 与 AWS SAM 规范集成

您可以使用 AWS SAM 策略模板向状态机添加权限。借助这些权限，您能够编排 Lambda 函数和其他 AWS 资源，以形成复杂而强大的工作流程。

Step Functions 与 SAM CLI 集成

Step Functions 已与 AWS SAM CLI 集成。这样，就可以快速将状态机开发到无服务器应用程序中。

试试本使用 AWS SAM 创建 Step Functions 状态机教程，学习 AWS SAM 如何使用创建状态机。

支持的 AWS SAM CLI 功能包括：

CLI 命令	描述
sam init	使用模板初始化无服务器应用程序。 AWS SAM 可以与 SAM 模板一起使用以实现 Step Functions。
sam validate	验证 AWS SAM 模板。
sam package	打包 AWS SAM 应用程序。它创建一个包含您的代码和依赖项的 ZIP 文件，然后将其上传到 Amazon S3。然后，它返回 AWS SAM 模板的副本，并将对本地构件的引用替换为此命令已将构件上传到的 Amazon S3 位置。

CLI 命令	描述
sam deploy	部署 AWS SAM 应用程序。
sam publish	将 AWS SAM 应用程序发布到 AWS Serverless Application Repository。此命令采用打包的 AWS SAM 模板并将应用程序发布到指定区域。

> **Note**
>
> 使用 AWS SAM 本地模式时，您可以在本地模拟 Lambda 和 API Gateway。但是，你不能使用 AWS SAM在本地模拟 Step Functions。

DefinitionSubstitutions 在AWS SAM模板中

您可以结合使用 CloudFormation 模板和 AWS SAM 来定义状态机。使用 AWS SAM，您可以在模板或单独的文件中定义内联状态机。以下 AWS SAM 模板包含模拟股票交易工作流的状态机。该状态机调用三个 Lambda 函数来查看股票的价格并确定是买入还是卖出股票。然后，相应交易会记录在 Amazon DynamoDB 表中。在以下模板中，Lambda 函数的 ARN 以及 DynamoDB 表是使用 `DefinitionSubstitutions` 指定的。

```
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
```

以下代码是使用 AWS SAM 创建 Step Functions 状态机教程中使用的文件 stock_trader.asl.json 内的状态机定义。此状态机定义包含几个用 ${dollar_sign_brace} 表示法表示的 DefinitionSubstitutions。例如，使用替换项 ${StockCheckerFunctionArn}，而不是为 Check Stock Value 任务指定静态 Lambda 函数

ARN。此替换项在模板的 DefinitionSubstitutions 属性中定义。DefinitionSubstitutions 是状态机资源的键值对的映射。在中DefinitionSubstitutions，$ {StockCheckerFunctionArn} 使用CloudFormation内部函数映射到StockCheckerFunction资源的 ARN。!GetAtt部署 AWS SAM 模板时，模板中的 DefinitionSubstitutions 将替换为实际值。

```
{
    "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
```

```
        }
    }
}
```

后续步骤

您可以通过以下资源了解有关使用 Step Funct AWS SAM ions 的更多信息：

- 完成教使用 AWS SAM 创建 Step Functions 状态机程以使用创建状态机 AWS SAM。
- 指定AWS::Serverless::StateMachine资源。
- 查找要使用的 AWS SAM 策略模板。
- 将 AWS Toolkit for Visual Studio Code 与 Step Functions 搭配使用。
- 查看 AWS SAM CLI 参考，详细了解中 AWS SAM提供的特征。

您还可以使用 Application Composer 中的 Workflow Studio 等可视化构建器在基础设施即代码 (IaC) 中设计和构建工作流。有关更多信息，请参阅 使用 Application Composer 中的 Workflow Studio。

使用 Application Composer 中的 Workflow Studio

AWS 应用程序编辑器 是一个可视化生成器，可帮助您使用简单的图形界面开发 AWS SAM 和 AWS CloudFormation 模板。借助 Application Composer，您可以通过在可视化画布中拖动、分组以及连接 AWS 服务来设计应用程序架构。然后，Application Composer 会根据您的设计创建基础设施即代码 (IaC) 模板，您可以使用该模板通过 AWS SAM 命令行界面 (AWS SAM CLI) 或 CloudFormation 部署应用程序。要了解有关 Application Composer 的更多信息，请参阅什么是 Application Composer。

Application Composer 中提供的 Workflow Studio 旨在帮助您设计和构建工作流。Application Composer 中的 Workflow Studio 提供一个视觉 IaC 环境，可让您轻松地将工作流整合到使用 CloudFormation 模板等 IaC 工具构建的无服务器应用程序中。当您在 Application Composer 中使用 Workflow Studio 时，它会将各个工作流步骤连接到 AWS 资源，并在 AWS SAM 模板中生成资源配置。它还会添加工作流运行所需的 IAM 权限。使用 Application Composer 中的 Workflow Studio，您可以创建应用程序原型，并将其转化为生产就绪的应用程序。

使用 Application Composer 中的 Workflow Studio 时，您可以在 Application Composer 画布和 Workflow Studio 之间来回切换。

主题

- 使用 Application Composer 中的 Workflow Studio 构建无服务器工作流
- 在 Workflow Studio 中使用 CloudFormation 定义替换项动态引用资源
- 将服务集成任务连接到增强型组件卡
- 导入现有项目并在本地同步它们
- AWS 应用程序编辑器 中不可用的 Workflow Studio 功能

使用 Application Composer 中的 Workflow Studio 构建无服务器工作流

1. 打开 Application Composer 控制台，然后选择创建项目来创建项目。
2. 在资源选项板的搜索字段中输入 **`state machine`**。
3. 将 Step Functions 状态机资源拖到画布上。
4. 选择在 Workflow Studio 中编辑，以编辑状态机资源。

   以下动画显示了如何切换到 Workflow Studio 来编辑状态机定义。
   演示如何使用 Application Composer 中的 Workflow Studio 的动画。

   为编辑在 Application Composer 中创建的状态机资源而与 Workflow Studio 的集成仅适用于 `AWS::Serverless::StateMachine` 资源。此集成不适用于使用 `AWS::StepFunctions::StateMachine` 资源的模板。

在 Workflow Studio 中使用 CloudFormation 定义替换项动态引用资源

在 Workflow Studio 中，您可以在工作流定义中使用 CloudFormation 定义替换项来动态引用您在 IaC 模板中定义的资源。您可以使用 `${dollar_sign_brace}` 表示法向工作流定义中添加占位符替换项，在 CloudFormation 堆栈创建过程中，它们会替换为实际值。有关定义替换项的更多信息，请参阅 DefinitionSubstitutions 在AWS SAM模板中。

以下动画显示了如何在状态机定义中为资源添加占位符替换项。
一个动画，演示了使用 Application Composer 中的 Workflow Studio 时如何动态引用资源，例如 AWS Lambda 函数、定义替换项。

将服务集成任务连接到增强型组件卡

您可以在 Application Composer 画布中将调用优化的服务集成的任务连接到增强型组件卡。这样做会自动映射工作流定义中通过 `${dollar_sign_brace}` 表示法指定的任何占位符替换项和

StateMachine 资源的 DefinitionSubstitution 属性。它还会为状态机添加适当的 AWS SAM 策略。

如果您映射优化的服务集成任务与标准组件卡，则 Application Composer 画布上不会显示连接线。

以下动画显示了如何将优化的任务连接到增强型组件卡，并在更改检查器中查看更改。
一个动画，演示了使用 Application Composer 中的 Workflow Studio 时，如何将调用优化的服务集成的任务连接到增强型组件卡。

您无法将处于 Task 状态的 AWS SDK 集成与增强型组件卡连接，也无法将优化的服务集成与标准组件卡连接。对于这类任务，您可以在 Application Composer 画布的资源属性面板中映射替换项，并在 AWS SAM 模板中添加策略。

> **Tip**
>
> 或者，您也可以在资源属性面板的定义替换项下为状态机映射占位符替换项。执行此操作时，您必须针对 Task 状态调用的 AWS 服务，为状态机执行角色添加必需的权限。有关执行角色可能需要的权限的信息，请参阅 Workflow Studio 中的执行角色。

以下动画显示了如何在资源属性面板中手动更新占位符替换项映射。
一个动画，演示了使用 Application Composer 中的 Workflow Studio 时，如何在资源属性面板中手动更新占位符替换项映射。

导入现有项目并在本地同步它们

您可以在 Application Composer 中打开现有 CloudFormation 和 AWS SAM 项目，直观查看，以便更好地了解，而且可以修改其设计。使用 Application Composer 的本地同步功能，您可以自动同步模板和代码文件，并将其保存到本地构建机器上。使用本地同步模式可以对您现有的开发流形成补充。请确保您的浏览器支持 File System Access API，它允许 Web 应用程序在本地文件系统中读取、写入和保存文件。我们建议使用 Google Chrome 或 Microsoft Edge。

AWS 应用程序编辑器 中不可用的 Workflow Studio 功能

当您在 Application Composer 中使用 Workflow Studio 时，某些 Workflow Studio 功能不可用。
此外，Inspector 面板中的 API 参数部分支持 CloudFormation 定义替换项。您可以在代码模式中使用 ${dollar_sign_brace} 表示法添加替换项。有关此表示法的更多信息，请参阅 DefinitionSubstitutions 在AWS SAM模板中。

以下列表说明了在 Application Composer 中使用 Workflow Studio 时不可用的 Workflow Studio 功能：

- 入门模板 - 入门模板是随时可以运行的示例项目，可自动创建工作流原型和定义。这类模板会将您的项目需要的所有相关 AWS 资源部署到您的 AWS 账户。
- 配置模式 - 此模式可让您管理状态机的配置。您可以在 IaC 模板中更新状态机配置，也可以使用 Application Composer 画布中的资源属性面板。有关在资源属性面板中更新配置的信息，请参阅 将服务集成任务连接到增强型组件卡。
- TestState API
- 在 Workflow Studio 中，通过操作下拉按钮导入或导出工作流定义的选项。不过，可以在 Application Composer 菜单中，依次选择打开 > 项目文件夹。请确保您已启用本地同步模式，以自动将 Application Composer 画布中的更改直接保存到本地机器。
- 执行按钮。当您在 Application Composer 中使用 Workflow Studio 时，Application Composer 会为工作流生成 IaC 代码。因此，您必须先部署模板。然后，在控制台中或通过 AWS Command Line Interface(AWS CLI) 运行工作流。

使用 AWS CloudFormation为 Step Functions 创建一个 Lambda 状态机

本教程向您展示如何使用创建基本 AWS Lambda 函数 AWS CloudFormation。您将使用 AWS CloudFormation 控制台和 YAML 模板来创建堆栈（IAM 角色、Lambda 函数和状态机）。然后，您将使用 AWS Step Functions 控制台开始执行状态机。

有关更多信息，请参阅《AWS CloudFormation 用户指南》中的使用 CloudFormation 模板和`AWS::StepFunctions::StateMachine`资源。

主题

- 第 1 步：设置您的 AWS CloudFormation 模板
- 步骤 2：使用 AWS CloudFormation 模板创建 Lambda 状态机
- 第 3 步：启动状态机执行

第 1 步：设置您的 AWS CloudFormation 模板

在使用示例模板之前，您应该了解如何声明 AWS CloudFormation 模板的不同部分。

主题

- 为 Lambda 创建 IAM 角色
- 创建 Lambda 函数
- 创建用于状态机执行的 IAM 角色
- 创建 Lambda 状态机

为 Lambda 创建 IAM 角色

定义与 Lambda 函数的 IAM 角色关联的信任策略。以下示例使用 YAML 或 JSON 定义信任策略。

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"
                      }
                  ]
              }
          }
```

创建 Lambda 函数

为将打印 `Hello World` 消息的 Lambda 函数定义以下属性。

> **Important**
>
> 确保您的 Lambda 函数与状态机位于同一个 AWS 账户和 AWS 区域下。

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"
```

```
            }
        },
```

创建用于状态机执行的 IAM 角色

定义与状态机执行的 IAM 角色关联的信任策略。

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": "*"
                                }
                            ]
                        }
                    }
                ]
            }
        },
```

创建 Lambda 状态机

定义 Lambda 状态机。

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"
                    ]
                }
```

```
            }
        }
```

步骤 2：使用 AWS CloudFormation 模板创建 Lambda 状态机

了解 AWS CloudFormation 模板的组件后，就可以将它们组合在一起并使用模板创建 AWS CloudFormation 堆栈。

创建 Lambda 状态机

1. 将以下示例数据复制到名为 `MyStateMachine.yaml`（适用于 YAML 示例）或 `MyStateMachine.json`（适用于 JSON）的文件中。

   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

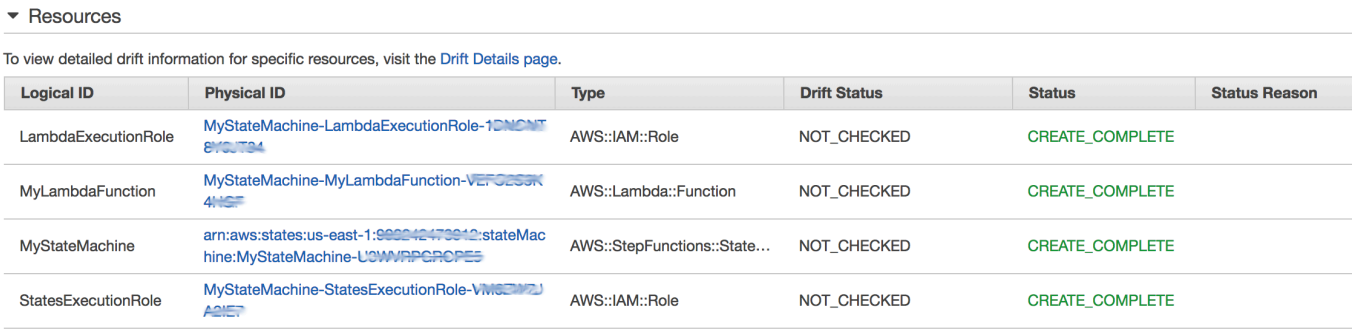
```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "An example template with an IAM role for a Lambda state
 machine.",
    "Resources": {
        "LambdaExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "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. 打开 AWS CloudFormation 控制台并选择创建堆栈。
3. 在选择模板页面上，选择将模板上传到 Amazon S3。选择您的 `MyStateMachine` 文件，然后选择下一步。
4. 在指定详细信息页面上，为堆栈名称输入 `MyStateMachine`，然后选择下一步。
5. 在选项页面上，选择下一步。
6. 在审核页面上，选择我确认， AWS CloudFormation 可能创建 IAM 资源。，然后选择创建。

   AWS CloudFormation 开始创建`MyStateMachine`堆栈并显示 CREATE_IN_ PROGRESS 状态。在此过程完成后， AWS CloudFormation 将显示 CREATE_COMPLETE 状态。
7. （可选）要显示您的堆栈中的资源，请选择堆栈，然后选择资源选项卡。



▼ Resources

To view detailed drift information for specific resources, visit the Drift Details page.

Logical ID	Physical ID	Type	Drift Status	Status	Status Reason
LambdaExecutionRole	MyStateMachine-LambdaExecutionRole-1[redacted]	AWS::IAM::Role	NOT_CHECKED	CREATE_COMPLETE	
MyLambdaFunction	MyStateMachine-MyLambdaFunction-[redacted]	AWS::Lambda::Function	NOT_CHECKED	CREATE_COMPLETE	
MyStateMachine	arn:aws:states:us-east-1:[redacted]:stateMachine:MyStateMachine-[redacted]	AWS::StepFunctions::State…	NOT_CHECKED	CREATE_COMPLETE	
StatesExecutionRole	MyStateMachine-StatesExecutionRole-[redacted]	AWS::IAM::Role	NOT_CHECKED	CREATE_COMPLETE	

第 3 步：启动状态机执行

在创建您的 Lambda 状态机后，可以开始执行。

启动状态机执行

1. 打开 Step Functions 控制台，然后选择你使用创建的状态机的名称 AWS CloudFormation。
2. 在 ***MyStateMachine-ABCDEFGHIJ1*** K 页面上，选择 “新建执行”。

   此时将显示新执行页面。
3. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。
4. 选择启动执行。

   此时将启动新的状态机执行，并显示一个说明正在运行的执行的新页面。
5. （可选）在执行详细信息中，查看执行状态以及已开始和已关闭时间戳。
6. 要查看执行结果，请选择输出。

使用 AWS CDK 为 Step Functions 创建一个 Lambda 状态机

本教程将介绍如何使用 AWS Cloud Development Kit (AWS CDK) 创建包含 AWS Lambda 函数的 AWS Step Functions 状态机。AWS CDK是一个基础设施即代码 (IAC) 框架，允许您使用成熟的编程语言定义 AWS 基础架构。您可以用 CDK 支持的语言之一编写包含一个或多个堆栈的应用程序。然后，您可以将其合成到AWS CloudFormation模板并将其部署到您的 AWS 帐户。我们将使用此方法来定义包含Lambda函数的Step Functions状态机，然后使用 AWS Management Console 来运行状态机。

在开始本教程之前，您必须按照《AWS Cloud Development Kit (AWS CDK) 开发人员指南》中开始使用 AWS CDK - 先决条件所述，设置您的 AWS CDK 开发环境。然后，在 AWS CLI 中使用以下命令安装 AWS CDK：

```
npm install -g aws-cdk
```

本教程产生的结果与the section called “使用创建 Lambda 状态机 AWS CloudFormation”相同。但是，在本教程中，AWS CDK 不需要您创建任何 IAM 角色，AWS CDK 会为您创建。AWS CDK 版本还包括一个 Succeed 步骤，用于说明如何向状态机添加其他步骤。

> **Tip**
>
> 要部署一个使用和启动Step Functions工作流程的示例无服务器应用程序，请参阅 T AWS CDK h TypeScript e AWS 账户 Worksho AWS Step Functions p 的模块 10-使用AWS CDK部署。

主题

- 第 1 步：设置您的 AWS CDK 项目
- 第 2 步：使用 AWS CDK 创建状态机
- 第 3 步：启动状态机执行
- 第 4 步：清除
- 后续步骤

第 1 步：设置您的 AWS CDK 项目

1. 在您的主目录或其他目录中，运行以下命令为您的新 AWS CDK 应用程序创建一个目录。

> **Important**
>
> 请确保将目录命名为 `step`。AWS CDK 应用程序模板使用目录的名称来生成源文件和类的名称。如果您使用其他名称，则您的应用将与本教程不匹配。

TypeScript

```
mkdir step && cd step
```

JavaScript

```
mkdir step && cd step
```

Python

```
mkdir step && cd step
```

Java

```
mkdir step && cd step
```

C#

确保您已安装.NET 版本 6.0 或更高版本。有关信息，请参阅受支持的版本。

```
mkdir step && cd step
```

2. 使用 cdk init 命令初始化应用程序。指定所需的模板（“应用程序”）和编程语言，如以下示例所示。

TypeScript

```
cdk init --language typescript
```

JavaScript

```
cdk init --language javascript
```

Python

```
cdk init --language python
```

初始化项目后，激活项目的虚拟环境并安装 AWS CDK 的基线依赖关系。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
cdk init --language java
```

C#

```
cdk init --language csharp
```

第 2 步：使用 AWS CDK 创建状态机

首先，我们将展示定义 Lambda 函数和 Step Functions 状态机的单独代码片段。然后，我们将解释如何在您的 AWS CDK 应用程序中将它们组合在一起。最后，您将了解如何合成和部署这些资源。

创建 Lambda 函数

以下 AWS CDK 代码定义了 Lambda 函数，并提供了其内联源代码。

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)
});
```

您可以在这个简短的示例代码中看到：

- 函数的逻辑名称 `MyLambdaFunction`。
- 函数的源代码，以字符串形式嵌入到 AWS CDK 应用程序的源代码中。
- 其他函数属性，例如要使用的运行时间（节点 18.x）、函数的入口点和超时。

创建状态机

我们的状态机有两种状态：Lambda 函数任务和 Succeed 状态。该函数要求我们创建一个 Step Functions the section called “任务” 来调用我们的函数。此 Task 状态用作状态机的第一个步骤。使用 Task 状态的`next()`方法将 success 状态添加到状态机中。以下代码首先调用名为 `MyLambdaTask` 的函数，然后使用 `next()` 方法定义名为 `GreetedWorld` 的 success 状态。

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")))
});
```

构建并部署 AWS CDK 应用程序

在新创建的 AWS CDK 项目中，编辑包含堆栈定义的文件，使其与下面的示例代码类似。您将从前面的部分中了解 Lambda 函数和 Step Functions 状态机的定义。

1. 如下例所示更新堆栈。

TypeScript

使用以下代码更新 `lib/step-stack.ts`。

```
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

使用以下代码更新 `lib/step-stack.js`。

```
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

使用以下代码更新 step/step_stack.py。

```
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

使用以下代码更新 src/main/java/com.myorg/StepStack.java。

```
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#

使用以下代码更新 scr/Step/StepStack.cs。

```
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. 保存源文件，然后在应用程序的主目录中运行 `cdk synth` 命令。

   AWS CDK 运行应用程序并合成 AWS CloudFormation 模板。然后，AWS CDK 会显示该模板。

   > **Note**
   >
   > 如果您曾经 TypeScript 创建过AWS CDK项目，则运行该`cdk synth`命令可能会返回以下错误。
   >
   > ```
   > TSError: # Unable to compile TypeScript:
   > bin/step.ts:7:33 - error TS2554: Expected 2 arguments, but got 3.
   > ```
   >
   > 修改 `bin/step.ts` 文件，如以下示例所示，可解决此错误。
   >
   > ```
   > #!/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. 要将 Lambda 函数和 Step Functions 状态机部署到您的 AWS 账户，请发出 `cdk deploy`。系统将要求您批准 AWS CDK 已生成的 IAM 策略。

第 3 步：启动状态机执行

在创建您的状态机后，可以开始执行。

启动状态机执行

1. 打开 Step Functions 控制台，然后选择您使用 AWS CDK 创建的状态机的名称。
2. 在状态机页面，选择启动执行。

   随即显示启动执行对话框。
3. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。
4. 选择启动执行。

   状态机的执行将启动，并显示一个说明正在运行的执行的新页面。
5. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

第 4 步：清除

测试状态机完成后，我们建议您删除状态机和相关的 Lambda 函数，以释放 AWS 账户中的资源。在您的应用程序的主目录中运行 `cdk destroy` 命令，以删除状态机。

后续步骤

要详细了解如何使用开发 AWS 基础架构AWS CDK，请参阅《AWS CDK开发人员指南》。

有关使用所选语言编写 AWS CDK 应用程序的信息，请参阅：

TypeScript

AWS CDK在 in 中使用 TypeScript

JavaScript

AWS CDK在 in 中使用 JavaScript

Python

在 Python 中使用 AWS CDK

Java

在 Java 中使用 AWS CDK

C#

在 C# 中使用 AWS CDK

有关本教程中使用的 AWS 构造库模块的更多信息，请参阅以下 AWS CDK API 参考概述：

- aws-lambda
- aws-stepfunctions
- aws-stepfunctions-tasks

使用同步 Express 状态机创建 API Gateway REST API 使用 AWS CDK

本教程将向您展示如何使用 AWS Cloud Development Kit (AWS CDK) 通过同步快速状态机创建一个 API Gateway REST API 作为后端集成。本教程将使用 `StepFunctionsRestApi` 构造将状态机连接到 API Gateway。`StepFunctionsRestApi` 构造将设置默认的输入/输出映射和 API Gateway REST API，具有所需的权限和 HTTP “ANY” 方法。 AWS CDK 是一个基础设施即代码 (IAC) 框架，允许您使用成熟的编程语言定义 AWS 基础架构。您使用 CDK 支持的语言之一编写包含一个或多个堆栈的应用程序，然后将其合成 AWS CloudFormation 模板并将其部署到您的账户。 AWS 我们将使用它来定义 API Gateway REST API，该API与同步快速状态机集成为后端，然后使用 AWS Management Console 来启动执行。

在开始本教程之前，请按照入门必备条件中所述设置您的 AWS CDK 开发环境，然后 AWS CDK 通过发布以下命令进行安装： AWS CDK

```
npm install -g aws-cdk
```

主题

- 第 1 步：设置您的 AWS CDK 项目
- 第 2 步：使用创建具有同步 Express 状态机后端集成的 API Gateway REST API AWS CDK
- 第 3 步：测试 API Gateway Gateway
- 第 4 步：清除

第 1 步：设置您的 AWS CDK 项目

首先，为您的新 AWS CDK 应用程序创建一个目录并初始化项目。

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
```

项目初始化后，激活项目的虚拟环境并安装其 AWS CDK基准依赖关系。

```
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**
>
> 请确保将目录命名为 `stepfunctions-rest-api`。 AWS CDK 应用程序模板使用目录的名称为源文件和类生成名称。如果您使用其他名称，则您的应用将与本教程不匹配。

现在为 AWS Step Functions 和 Amazon API Gateway 安装构造库模块。

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

编辑项目的 `pom.xml` 将以下依赖项添加到现有 `<dependencies>` 容器。

```
        <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 会在下次构建应用程序时自动安装这些依赖关系。要构建，请发出 `mvn compile` 或使用 Java IDE 的构建命令。

C#

```
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.Stepfunctions
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.APIGateway
```

您也可以使用 Visual Studio NuGet GUI 安装指定的软件包，该用户界面可通过 “工具” > “NuGet 包管理器” > “管理解决方案 NuGet 包” 获得。

安装模块后，您可以通过导入以下软件包在 AWS CDK 应用程序中使用它们。

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

将以下内容添加到 `stepfunctions-rest-api.go` 内的 `import` 中

```
"github.com/aws/aws-cdk-go/awscdk/awsapigateway"
"github.com/aws/aws-cdk-go/awscdk/awsstepfunctions"
```

第 2 步：使用创建具有同步 Express 状态机后端集成的 API Gateway REST API AWS CDK

首先，我们将介绍定义同步 Express 状态机和 API Gateway REST API 的各个代码，然后解释如何将它们组合到您的 AWS CDK 应用程序中。然后，您将了解如何合成和部署这些资源。

> **Note**
>
> 我们将在此处展示的状态机将是一个带有 `Pass` 状态的简单状态机。

创建快速状态机

这是定义带有状态的简单`Pass`状态机的 AWS CDK 代码。

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,
})
```

您可以在这个简短的片段中看到：

- 名为 PassState 的计算机定义，它是一个 Pass 状态。
- 状态机的逻辑名称，MyStateMachine。
- 机器定义被用作状态机定义。
- 状态机类型设置为 EXPRESS，因为 StepFunctionsRestApi 只支持同步快速状态机。

使用 **StepFunctionsRestApi** 构造创建 API Gateway REST API

我们将使用 StepFunctionsRestApi 构造来创建具有所需权限和默认输入/输出映射的 API Gateway REST API。

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,
})
```

构建并部署 AWS CDK 应用程序

在您创建的 AWS CDK 项目中，编辑包含堆栈定义的文件，使其看起来像下面的代码。您将从上面的部分中了解 Step Functions 状态机和 API Gateway 的定义。

TypeScript

更新 `lib/stepfunctions-rest-api-stack.ts` 读取如下信息。

```
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

更新 `lib/stepfunctions-rest-api-stack.js` 读取如下信息。

```
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

更新 stepfunctions_rest_api/stepfunctions_rest_api_stack.py 读取如下信息。

```
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

更新 src/main/java/com.myorg/StepfunctionsRestApiStack.java 读取如下信息。

```
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#

更新 `src/StepfunctionsRestApi/StepfunctionsRestApiStack.cs` 读取如下信息。

```
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

更新 `stepfunctions-rest-api.go` 读取如下信息。

```
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
```

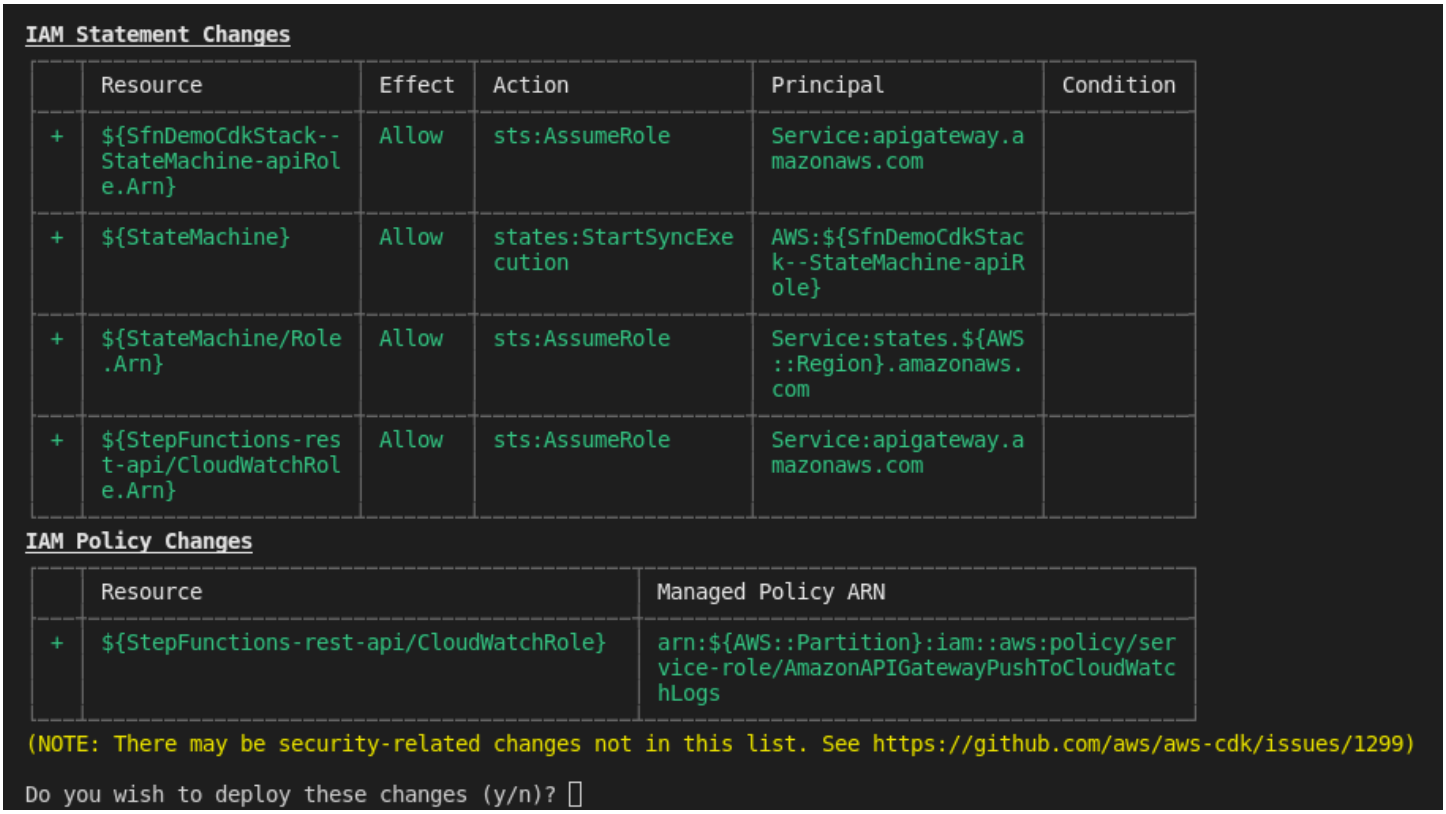
```
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
    // If unspecified, this stack will be "environment-agnostic".
    // Account/Region-dependent features and context lookups will not work, but a
    // single 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")),
    // }
}
```

保存源文件，然后在应用程序的主目录中发出 `cdk synth`。 AWS CDK 运行应用程序并从中合成一个 AWS CloudFormation 模板，然后显示该模板。

要将 Amazon API Gateway 和 AWS Step Functions 状态机实际部署到您的 AWS 账户，请执行以下操作`cdk deploy`。系统将要求您批准 AWS CDK 已生成的 IAM 策略。正在创建的策略与下面类似：

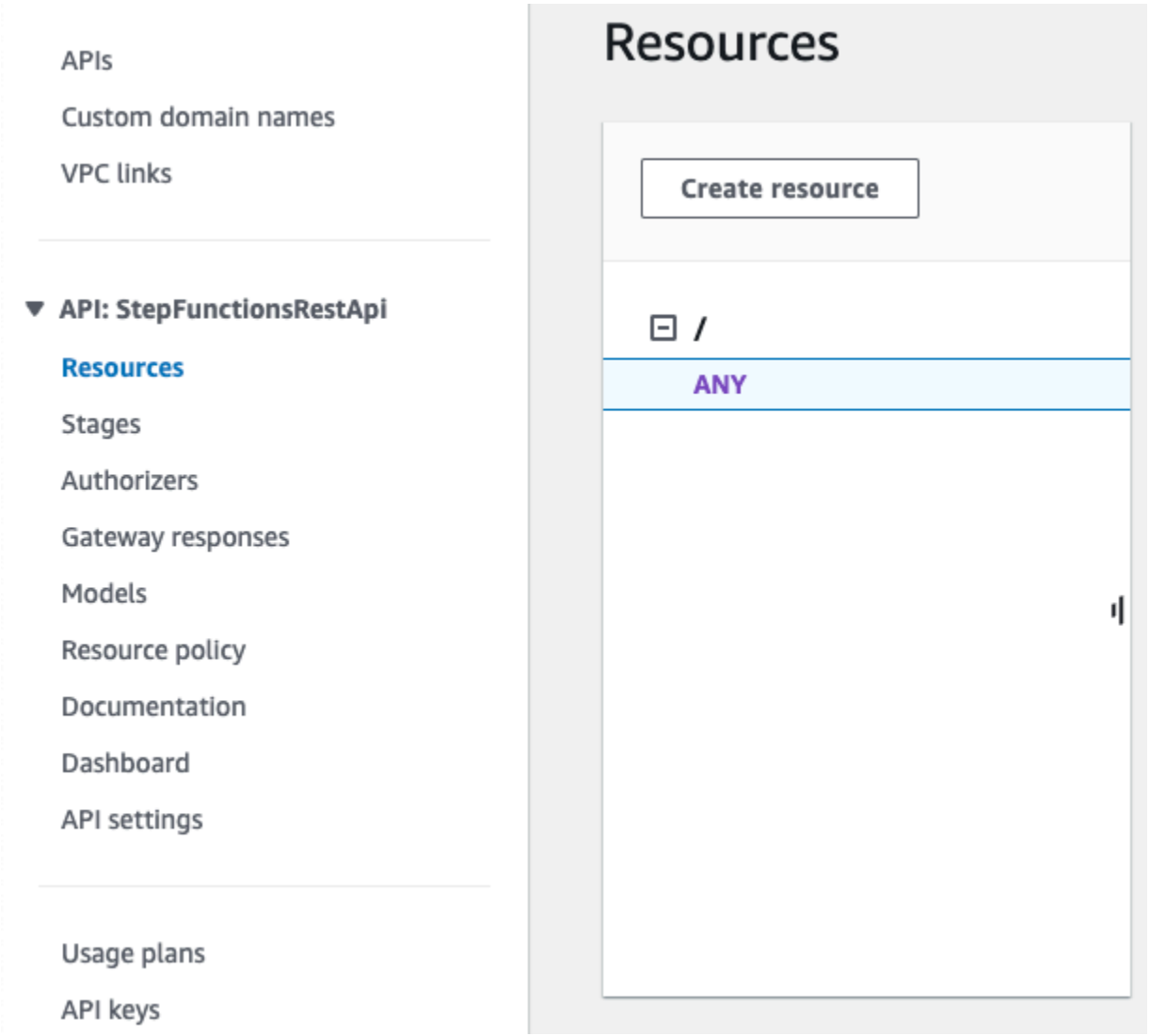


第 3 步：测试 API Gateway Gateway

使用同步快速状态机创建 API Gateway REST API 作为后端集成后，您可以测试 API Gateway。

使用 API Gateway 控制台测试已部署的 API Gateway

1. 打开 Amazon API Gateway 控制台并登录。
2. 选择名为 `StepFunctionsRestApi` 的 REST API。
3. 在资源窗格中，选择 ANY 方法。



4. 选择测试选项卡。您可能需要选择右箭头按钮，以显示该选项卡。
5. 对于方法，选择 POST。
6. 在请求正文中，复制以下请求参数。

```
{
    "key": "Hello"
}
```

7. 选择测试。此时将显示以下信息：

   - 请求是为方法调用的资源路径。
   - 状态是响应的 HTTP 状态代码。
   - 延迟是收到调用方请求和返回响应之间的时间。

- 响应正文是 HTTP 响应正文。
- 响应标头是 HTTP 响应标头。
- 日志显示了在 API Gateway 控制台之外调用此方法时本应写入的模拟 Amazon Lo CloudWatch gs 条目。

> **Note**
>
> 尽管 CloudWatch 日志条目是模拟的，但方法调用的结果是真实的。

响应正文输出应如下所示：

```
"Hello"
```

> **Tip**
>
> 使用不同的方法和无效的输入来尝试 API Gateway，查看错误输出。您可能希望更改状态机来查找特定的键，并在测试期间提供错误的键，以使状态机执行失败，并在响应正文输出中生成错误消息。

使用 cURL 测试已部署的 API

1. 打开终端窗口。
2. 复制以下 cURL 命令将其粘贴到终端窗口中，同时将 `<api-id>` 替换为您的 API 的 API ID 并将 `<region>` 替换为部署您的 API 的区域。

```
curl -X POST\
 'https://<api-id>.execute-api.<region>.amazonaws.com/prod' \
 -d '{"key":"Hello"}' \
 -H 'Content-Type: application/json'
```

响应正文输出应如下所示：

```
"Hello"
```

> **Tip**
>
> 使用不同的方法和无效的输入来尝试 API Gateway，查看错误输出。您可能希望更改状态机来查找特定的键，并在测试期间提供错误的键，以使状态机执行失败，并在响应正文输出中生成错误消息。

第 4 步：清除

当您完成了 API Gateway 测试后，可以使用 AWS CDK 卸载状态机和 API Gateway。在您的应用程序的主目录中发布 `cdk destroy`。

AWS Step Functions 适用于 Python 的数据科学软件开发工具包

AWS Step Functions 数据科学 SDK 是一个面向数据科学家的开源库。使用此 SDK，您可以创建使用和 Step Functions 处理 SageMaker 和发布机器学习模型的工作流程。您还可以在 Python 中创建多步骤机器学习工作流程，以大规模协调 AWS 基础架构，而无需单独配置和集成 AWS 服务。

AWS Step Functions 数据科学开发工具包提供了一个 Python API，可用于创建和调用 Step Functions 工作流。您可以直接在 Python 和 Jupyter 笔记本中管理和执行这些工作流。

除了直接在 Python 中创建可用于生产的工作流程外，D AWS Step Functions ata Science SDK 还允许您复制该工作流程，尝试新选项，然后将经过改进的工作流程投入生产。

有关 AWS Step Functions 数据科学 SDK 的更多信息，请参阅以下内容：

- GitHub 上的项目
- 开发工具包文档
- 以下示例笔记本可在SageMaker 控制台的 Jupyter 笔记本实例和相关GitHub 项目中使用：
  - `hello_world_workflow.ipynb`
  - `machine_learning_workflow_abalone.ipynb`
  - `training_pipeline_pytorch_mnist.ipynb`

使用 Terraform 部署状态机

HashiCorp 的 Terraform 是一个使用基础设施即代码 (IaC) 构建应用程序的框架。借助 Terraform，您可以创建状态机并使用特征，例如预览基础架构部署和创建可重复使用的模板。Terraform 模板通过将代码分解为多个较小的块来帮助您维护和重用代码。

如果您熟悉 Terraform，则可以按照本主题中描述的开发生命周期作为在 Terraform 中创建和部署状态机的模型。如果您不熟悉 Terraform，我们建议您先完成 AWS 上的 Terraform 入门工作坊，以熟悉 Terraform。

> **Tip**
>
> 要在 AWS 账户中部署使用 Terraform 构建的状态机示例，请参阅《AWS Step Functions 研讨会》中的使用基础设施即代码管理状态机模块。

本主题内容

- 先决条件
- 使用 Terraform 的状态机开发生命周期
- 状态机的 IAM 角色和策略

先决条件

在开始之前，您必须满足以下先决条件：

- 在您的系统上安装 Terraform。有关安装 Terraform 的信息，请参阅安装 Terraform。
- 在系统上安装 Step Functions Local。我们建议您安装 Step Functions Local Docker 映像，用于使用 Step Functions Local。有关更多信息，请参阅在本地测试状态机。
- 安装 AWS SAM CLI。有关安装信息，请参阅《AWS Serverless Application Model 开发人员指南》中的安装 AWS SAM CLI。
- 安装 AWS Toolkit for Visual Studio Code，用于查看状态机的工作流图。有关安装信息，请参阅《AWS Toolkit for Visual Studio Code 用户指南》中的安装 AWS Toolkit for Visual Studio Code。

使用 Terraform 的状态机开发生命周期

以下过程说明了如何使用在 Step Functions 控制台中使用 Workflow Studio 构建的状态机原型作为使用 Terraform 和 AWS Toolkit for Visual Studio Code 进行本地开发的起点。

要查看讨论使用 Terraform 开发状态机并详细介绍最佳实操的完整示例，请参阅编写 Step Functions Terraform 项目的最佳实操。

使用 Terraform 启动状态机的开发生命周期

1. 使用以下命令可引导新的 Terraform 项目。

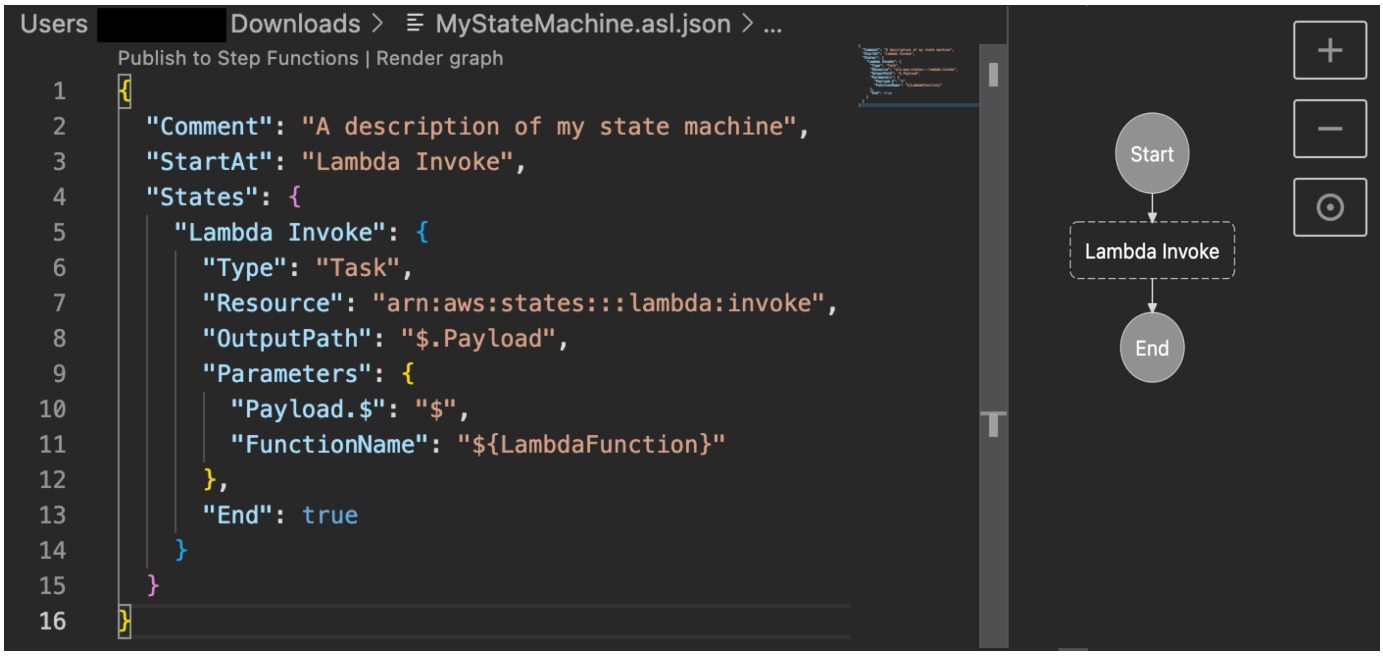
```
terraform init
```

2. 打开 Step Functions 控制台，为状态机创建原型。
3. 在 Works Studio 中，执行以下操作：
   a. 创建您的工作流原型。
   b. 导出工作流的 Amazon States Language (ASL) 定义。为此，请选择导入/导出下拉列表，然后选择导出 JSON 定义。
4. 将导出的 ASL 定义保存在项目目录中。

   您将导出的 ASL 定义作为输入参数传递给使用 `templatefile` 函数的 `aws_sfn_state_machine` Terraform 资源。此函数在定义字段中使用，该字段传递导出的 ASL 定义和任何变量替换。

   > **Tip**
   >
   > 由于 ASL 定义文件可能包含较长的文本块，因此我们建议您避免使用内联 EOF 方法。这可让您更轻松地将参数替换到状态机定义中。

5. （可选）更新 IDE 中的 ASL 定义并使用 AWS Toolkit for Visual Studio Code 可视化您的更改。



为避免持续导出定义并将其重构到项目中，我们建议您在 IDE 中进行本地更新，并使用 Git 跟踪这些更新。

6. 使用 Step Functions Local 测试工作流。

> **Tip**
>
> 您还可以使用 AWS SAM CLI Local 在状态机中本地测试与 Lambda 函数和 API Gateway API 的服务集成。

7. 在部署状态机之前，请预览状态机和其他 AWS 资源。要执行此操作，请运行以下命令。

```
terraform plan
```

8. 使用以下命令从本地环境或通过 CI/CD 管道部署状态机。

```
terraform apply
```

9. （可选）使用以下命令清理资源并删除状态机。

```
terraform destroy
```

状态机的 IAM 角色和策略

使用 Terraform 服务集成策略为您的状态机添加必要的 IAM 权限，例如调用 Lambda 函数的权限。您还可以定义明确的角色和策略，并将它们与状态机相关联。

以下 IAM 策略示例授予您的状态机调用名为 *myFunction* 的 Lambda 函数的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myFunction"
    }
  ]
}
```

我们还建议在 Terraform 中为状态机定义 IAM 策略时使用 aws_iam_policy_document 数据来源。这可以帮助您检查策略是否格式有误，并用变量替换任何资源。

以下 IAM 策略示例使用 aws_iam_policy_document 数据来源，并授予您的状态机调用名为 *myFunction* 的 Lambda 函数的访问权限。

```
data "aws_iam_policy_document" "state_machine_role_policy" {

  statement {
    effect = "Allow"

    actions = [
      "lambda:InvokeFunction"
    ]

    resources = ["${aws_lambda_function.[[myFunction]].arn}:*"]
  }

}
```

> **Tip**
>
> 要查看使用 Terraform 部署的更高级的 AWS 架构模式，请参阅 Terraform examples at Serverless Land Workflows Collection。

测试和调试

Step Functions 提供测试和调试状态机的不同方法。例如，您可以在控制台中测试和调试状态机，使用 TestState API 测试单个状态，或者使用 Step Functions Local 在本地测试状态机。

使用 TestState API，您可以提供单个状态的定义并执行它。您无需创建状态机或更新现有状态机，即可测试单个状态。

Step Functions Local 是 Step Functions 的可下载版本，它允许您使用在自己的开发环境中运行的 Step Functions 版本来开发和测试应用程序。使用 Step Functions Local，您可以在本地开发环境中运行状态机来测试其输入和输出数据流、与支持的服务的集成等。

主题

- 使用 TestState API 测试状态
- 在本地测试状态机

使用 TestState API 测试状态

TestStateAPI 接受单一状态的定义并执行它。您无需创建状态机或更新现有状态机，即可测试状态。

使用 TestState API，您可以测试以下内容：

- 状态的输入和输出处理数据流。
- 与其他 AWS 服务 请求和响应的AWS 服务 集成
- HTTP 任务请求和响应

要测试状态，您还可以使用 Step Functions 控制台、AWS Command Line Interface (AWS CLI) 或 SDK。

`TestState` API 扮演一个 IAM 角色，该角色必须针对状态访问的资源包含必需的 IAM 权限。有关状态可能需要的权限的信息，请参阅IAM使用 TestState API 的权限。

主题

- 使用 TestState API 的注意事项
- 在 TestState API 中使用检查级别

- IAM使用 TestState API 的权限
- 测试状态（控制台）
- 使用 AWS CLI 测试状态
- 测试和调试输入和输出数据流

使用 TestState API 的注意事项

使用 TestStateAPI，您一次只能测试一个状态。您可以测试的状态包括：

- 所有任务类型，活动除外
- Pass
- Wait
- Choice
- Succeed
- Fail

在使用 `TestState` API 时，请记住以下注意事项。

- 该 TestState API 不包括对以下内容的支持：
  - 使用以下资源类型的 任务状态 状态：
    - 活动
    - 类型 `.sync` 或 `.waitForTaskToken` 的服务集成模式
  - Parallel 状态
  - Map 状态
- 测试最多可以运行五分钟。如果测试超过此持续时间，则会因 `States.Timeout` 错误而失败。

在 TestState API 中使用检查级别

要使用 TestStateAPI 测试状态，您需要提供该状态的定义。然后，测试返回输出。对于每种状态，您可以指定要在测试结果中查看的详细信息量。这些详细信息提供有关您要测试的状态的更多信息。例如，如果您使用了任何输入和输出数据处理筛选器，例如状态中的 `InputPath` 或 `ResultPath`，则可以查看中间和最终的数据处理结果。

Step Functions 提供以下级别，供您指定要查看的详细信息：

- 信息
- 调试
- 跟踪

所有这些级别还会返回 `status` 和 `nextState` 字段。`status` 表示状态执行的状态。例如，`SUCCEEDED`、`FAILED`、`RETRIABLE` 和 `CAUGHT_ERROR`。`nextState` 表示要过渡到的下一个状态的名称。如果您尚未在定义中定义下一个状态，则此字段会返回一个空值。

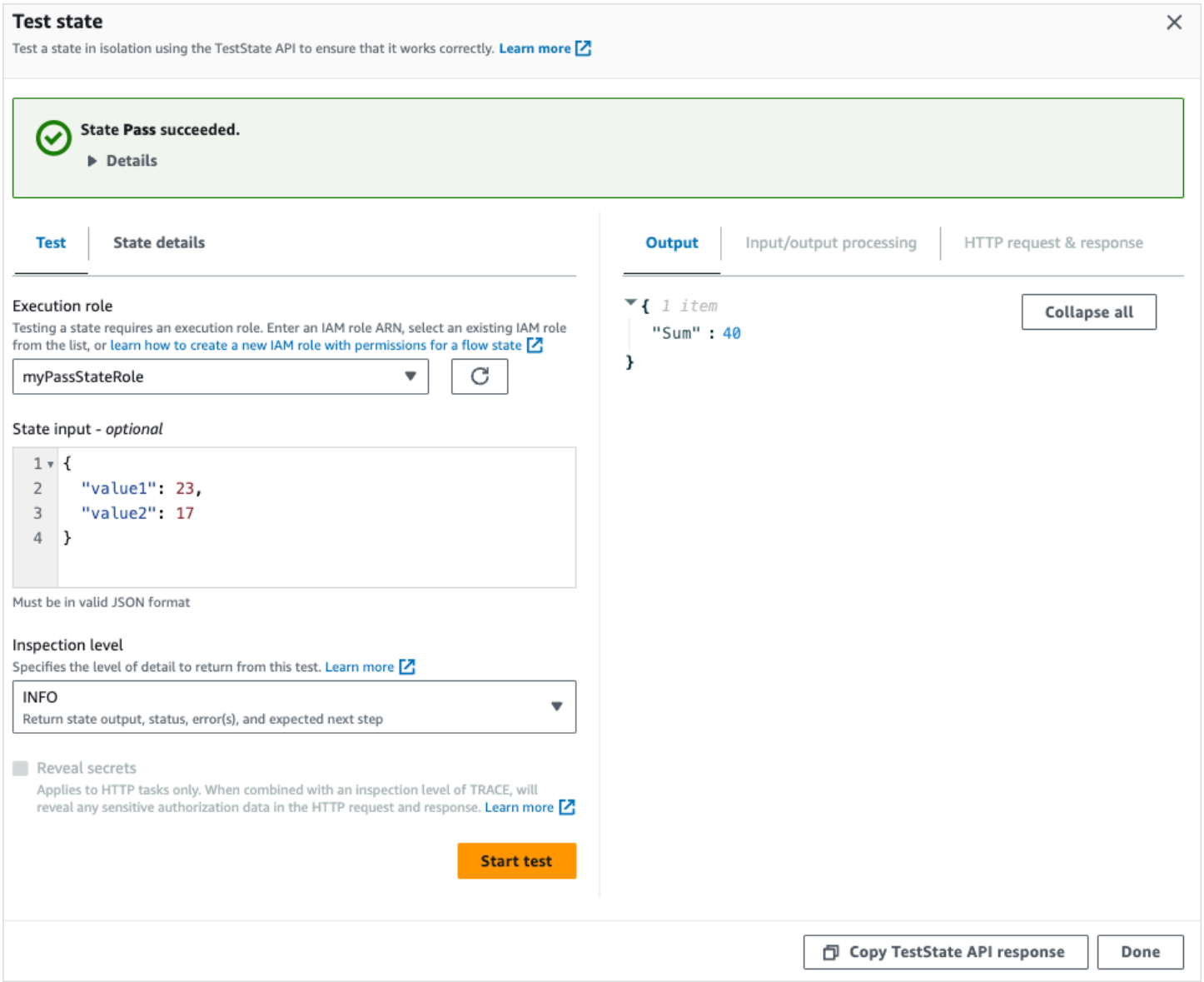
有关在 Step Functions 控制台和 AWS CLI 中使用这些检查级别测试状态的信息，请参阅测试状态（控制台）和使用 AWS CLI 测试状态。

“信息”检查级别

如果测试成功，此级别会显示状态输出。如果测试失败，此级别会显示错误输出。默认情况下，如果您未指定级别，Step Functions 会将检查级别设置为信息。

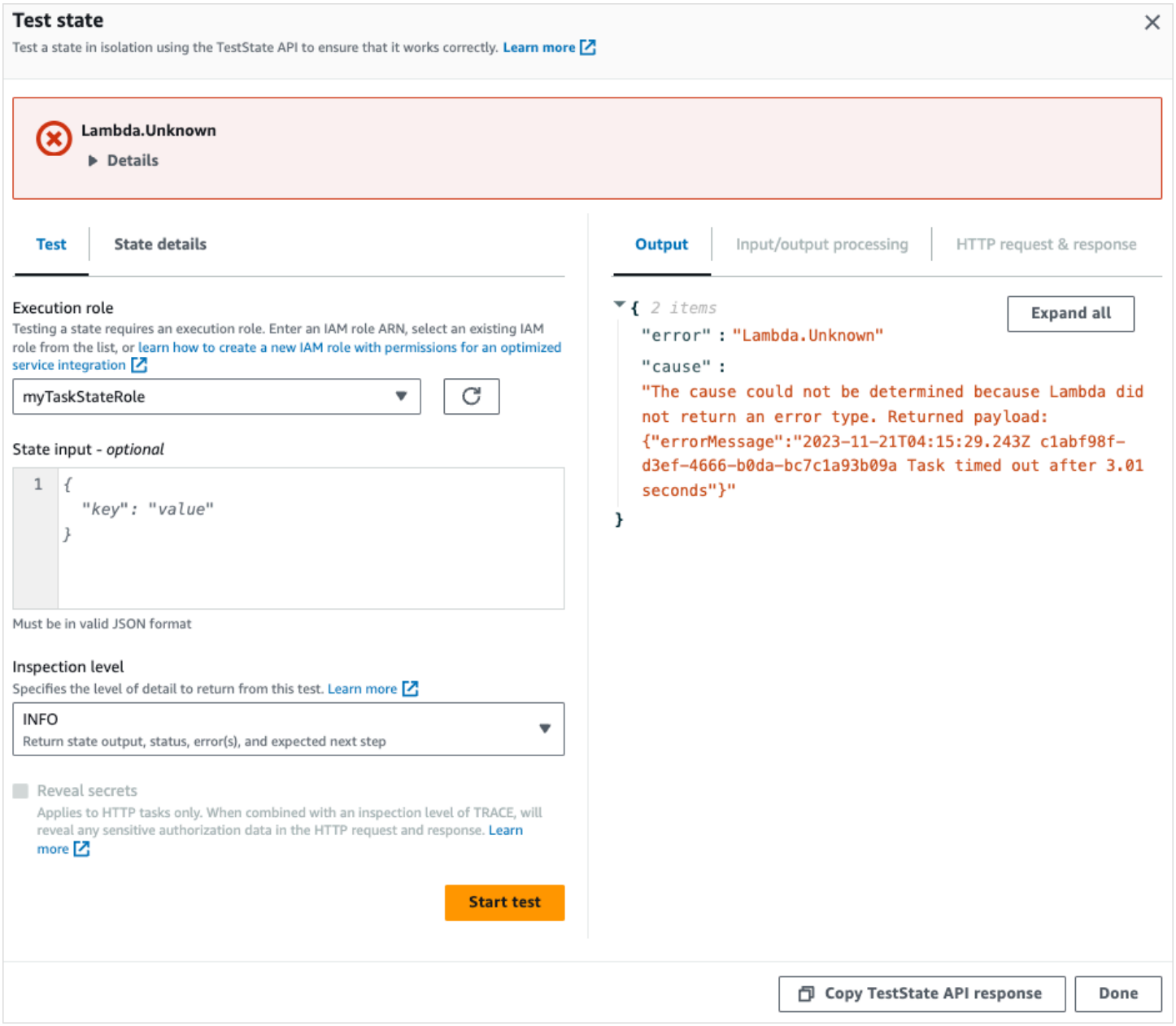
使用信息级别时测试成功的示例

下图显示了成功的 Pass 状态测试。此状态的检查级别设置为信息，该状态的输出显示在输出选项卡中。



使用信息级别时测试失败的示例

下图显示了当检查级别设置为信息时失败的 Task 状态测试。输出选项卡会显示错误输出，其中包含错误名称和错误原因的详细说明。



“调试”检查级别

如果测试成功，此级别会显示状态输出以及输入和输出数据处理的结果。

如果测试失败，此级别会显示错误输出。此级别显示最多到故障点的中间数据处理结果。例如，假设您测试了一个调用 Lambda 函数的 Task 状态。想象一下，您已向 Task 状态应用 InputPath、参数、ResultPath 和 OutputPath 筛选器。假设调用失败。在此例中，DEBUG 级别基于应用的筛选器按以下顺序显示数据处理结果：

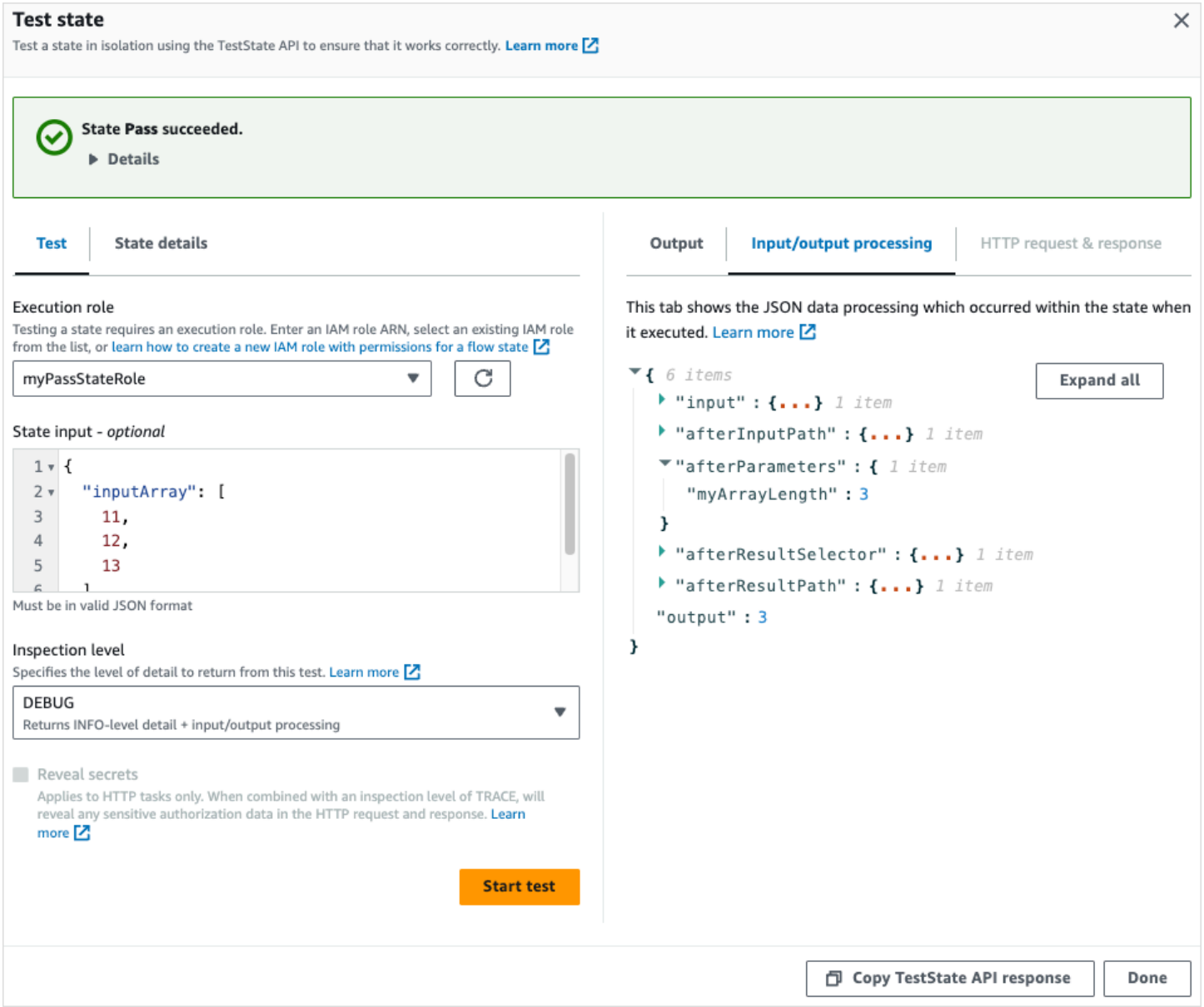
- `input`：原始状态输入

- `afterInputPath`：Step Functions 应用 `InputPath` 筛选器后的输入。
- `afterParameters`：Step Functions 应用 `Parameters` 筛选器后的有效输入。

此级别中提供的诊断信息可以帮助您排查与服务集成或您可能已定义的输入和输出数据处理流相关的问题。

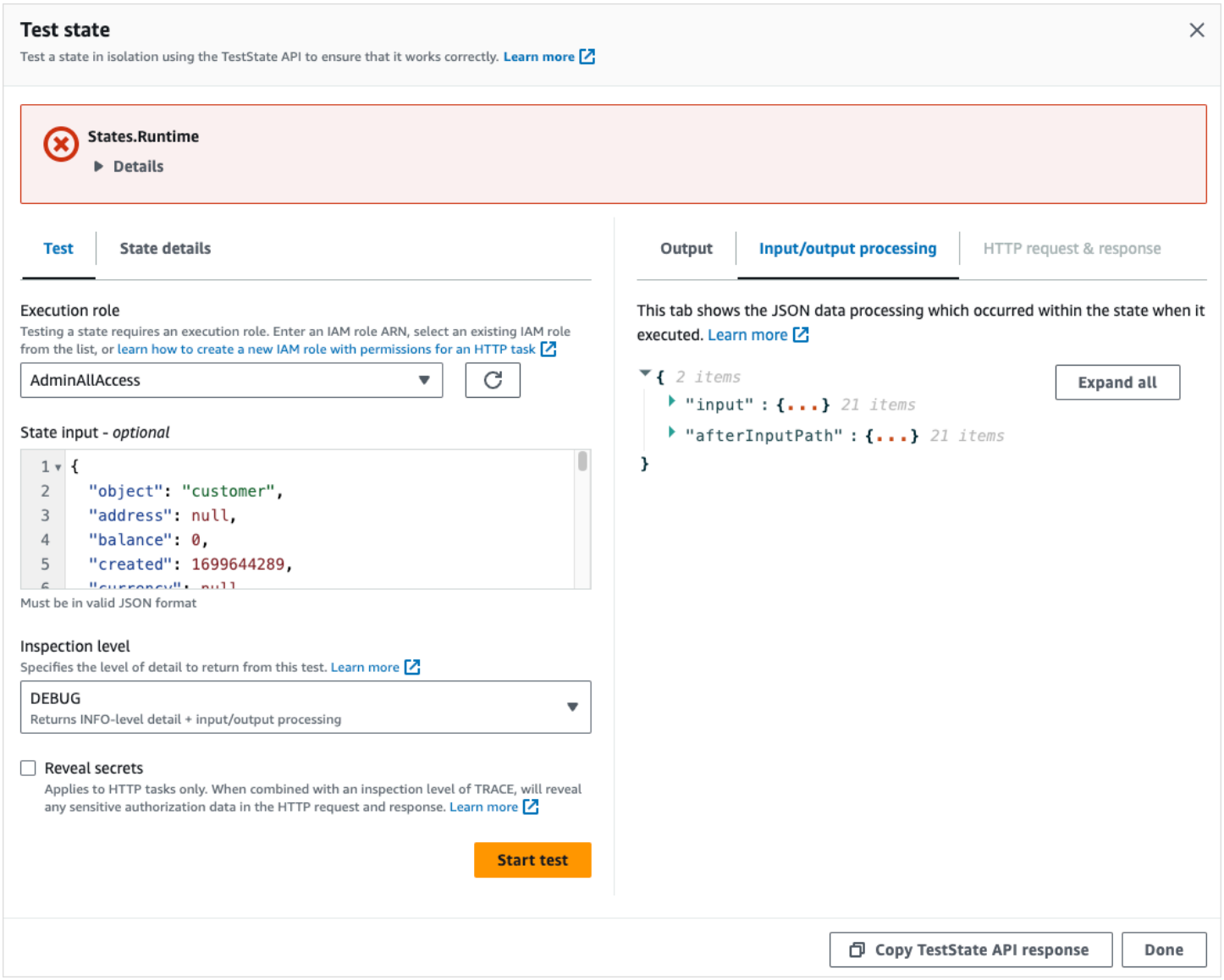
使用调试级别时测试成功的示例

下图显示了成功的 Pass 状态测试。此状态的检查级别设置为调试。下图中的输入/输出处理选项卡显示了在针对提供的输入应用 `Parameters` 后此状态的结果。



使用调试级别时测试失败的示例

下图显示了当检查级别设置为调试时失败的 Task 状态测试。下图中的输入/输出处理选项卡显示了此状态最多到故障点的输入和输出数据处理结果。



“跟踪”检查级别

Step Functions 提供用于测试 HTTP 任务的跟踪级别。此级别会返回有关 Step Functions 发出的 HTTP 请求和第三方 API 返回的响应的信息。响应可能包含标头和请求正文等信息。此外，您还可以在此级别查看状态输出以及输入和输出数据处理的结果。

如果测试失败，此级别会显示错误输出。

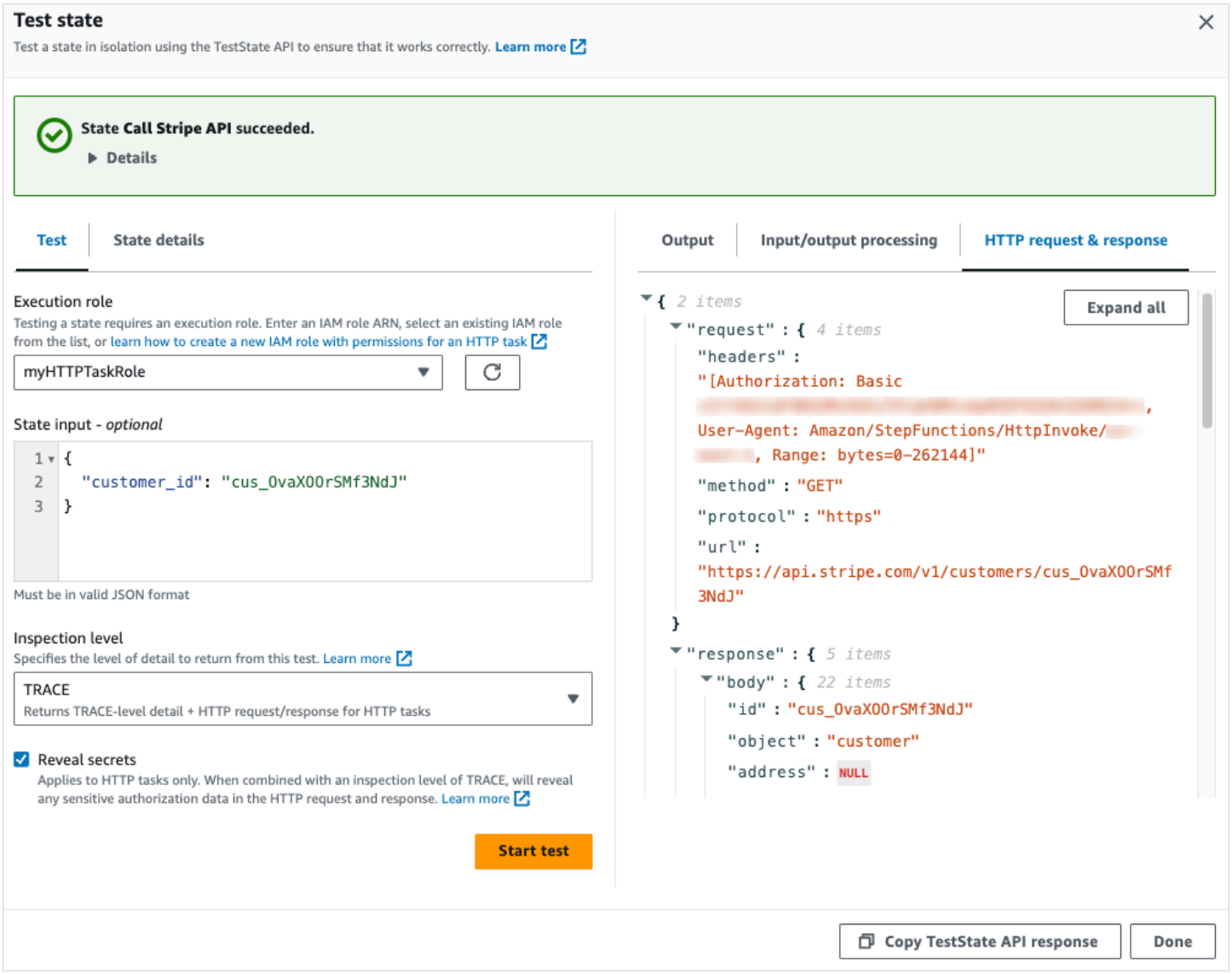
此级别仅适用于 HTTP 任务。如果您将此级别用于其他状态类型，则 Step Functions 会抛出错误。

将检查级别设置为 TRACE 时，您还可以查看EventBridge 连接中包含的密钥。为此，您必须在TestStateAPI `true` 中将`revealSecrets`参数设置为。此外，您必须确保调用 TestState API 的IAM用户拥有该`states:RevealSecrets`操作的权限。有关设置 `states:RevealSecrets` 权限的 IAM策略示例，请参阅IAM使用 TestState API 的权限。如果没有此权限，则 Step Functions 会抛出访问遭拒错误。

如果您将 `revealSecrets` 参数设置为 `false`，则 Step Functions 会省略 HTTP 请求和响应数据中的所有密钥。

使用跟踪级别时测试成功的示例

下图显示了成功的 HTTP 任务测试。此状态的检查级别设置为跟踪。下图中的 HTTP 请求和响应选项卡显示了第三方 API 调用的结果。



IAM使用 TestState API 的权限

调用 `TestState` API 的 IAM 用户必须有权执行 `states:TestState` 和 `iam:PassRole` 操作。此外，如果您将 revealSecrets 参数设置为 `true`，则必须确保 IAM 用户有权执行 `states:RevealSecrets` 操作。如果没有此权限，则 Step Functions 会抛出访问遭拒错误。

此外，对于您的状态访问的资源，您还必须确保执行角色包含必需的 IAM 权限。有关状态可能需要的权限的信息，请参阅管理执行角色。

以下 IAM 策略示例设置了 `states:TestState`、`iam:PassRole` 和 `states:RevealSecrets` 权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:TestState",
        "states:RevealSecrets",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

测试状态（控制台）

您可以在控制台中测试状态，并检查状态输出或输入和输出数据处理流。对于 HTTP 任务，您可以测试原始 HTTP 请求和响应。

测试状态

1. 打开 Step Functions 控制台。
2. 选择创建状态机开始创建状态机，或选择一个现有状态机。
3. 在 Workflow Studio 的设计模式中，选择要测试的状态。
4. 从 Workflow Studio 的 Inspector 面板中选择测试状态。
5. 在测试状态对话框中，执行以下操作：

a. 对于执行角色，选择一个执行角色来测试状态。请确保您对要测试的状态具有必需的 IAM 权限。

b. （可选）针对测试提供所选状态需要的任何 JSON 输入。

c. 对于检查级别，根据要查看的值选择以下选项之一：

- 信息：如果测试成功，此级别会在输出选项卡中显示状态输出。如果测试失败，信息级别会显示错误输出，其中包含错误名称和错误原因的详细说明。默认情况下，如果您未选择级别，Step Functions 会将检查级别设置为信息。
- 调试：如果测试成功，此级别会显示状态输出以及输入和输出数据处理的结果。如果测试失败，调试级别会显示错误输出，其中包含错误名称和错误原因的详细说明。
- 跟踪：此级别会显示原始的 HTTP 请求和响应，对于验证标头、查询参数和其他特定于 API 的详细信息非常有用。此选项仅适用于 HTTP 任务。

  您可以根据需要选择显示密钥。结合跟踪检查级别，此设置可让您查看 EventBridge 连接插入的敏感数据，例如 API 密钥。您用于访问控制台的 IAM 用户身份必须具有执行 `states:RevealSecrets` 操作的权限。如果没有此权限，则在您开始测试时 Step Functions 会抛出访问遭拒错误。有关设置 `states:RevealSecrets` 权限的 IAM 策略示例，请参阅IAM使用 TestState API 的权限。

d. 选择开始测试。

使用 AWS CLI 测试状态

您可以使用中的 TestStateAPI 来测试支持的状态AWS CLI。此 API 会接受状态的定义并执行它。

对于每种状态，您可以指定要在测试结果中查看的详细信息量。这些详细信息提供有关状态执行的额外信息，包括其输入和输出数据处理结果以及 HTTP 请求和响应信息。以下示例展示了您可以为 TestState API 指定的不同检查级别。切记用特定资源信息替换##文本。

本部分包含以下示例，它们说明了如何使用 Step Functions 在 AWS CLI 中提供的不同检查级别：

- 使用 INFO inspectionLevel
- 使用 DEBUG inspectionLevel
- 使用 TRACE inspectionLevel
- 在中AWS CLI使用 jq 实用程序筛选和打印 TestState API 返回的 HTTP 响应

示例 1：使用 INFO inspectionLevel 来测试 Choice 状态

要使用中的 Insp INFO ectionLevel 测试状态 AWS CLI，请按以下示例所示运行test-state命令。

```
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}'
```

此示例使用 Choice 状态根据您提供的数字输入来确定该状态的执行路径。默认情况下，如果您未设置级别，Step Functions 会将 inspectionLevel 设置为 INFO。

Step Functions 返回以下输出：

```
{
    "output": "{\"number\": 2}",
    "nextState": "Equals 2",
    "status": "SUCCEEDED"
}
```

示例 2：使用 DEBUG inspectionLevel 针对 Pass 状态调试输入和输出数据处理

要使用中的 Insp DEBUG ectionLevel 测试状态 AWS CLI，请按以下示例所示运行test-state命令。

```
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
```

此示例使用 Pass 状态来展示 Step Functions 如何使用输入和输出数据处理筛选器来筛选和处理输入 JSON 数据。此示例使用以下筛选器：InputPath、##、ResultPath 和 OutputPath。

Step Functions 返回以下输出：

```
{
    "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"
}
```

示例 3：使用 TRACE inspectionLevel 和 revealSecrets 检查发送到第三方 API 的 HTTP 请求

要使用 Insp TRACE ectionLevel 和中的 revealSecret s 参数测试 HTTP 任务 AWS CLI，请按以下test-state示例所示运行命令。

```
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
```

此示例测试 HTTP 任务是否调用了指定的第三方 API，https://httpbin.org/。它还会显示 API 调用的 HTTP 请求和响应数据。

```
{
    "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"
}
```

示例 4：使用 jq 实用程序筛选并打印 TestState API 返回的响应

TestState API 在其响应中以转义字符串形式返回 JSON 数据。以下AWS CLI示例扩展了示例 3，并使用该jq实用程序筛选和打印 TestState API 以人类可读的格式返回的 HTTP 响应。有关信息jq及其安装说明，请参阅 jq on。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'
```

以下示例显示了以人类可读的格式返回的输出。

```
{
  "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"
}
```

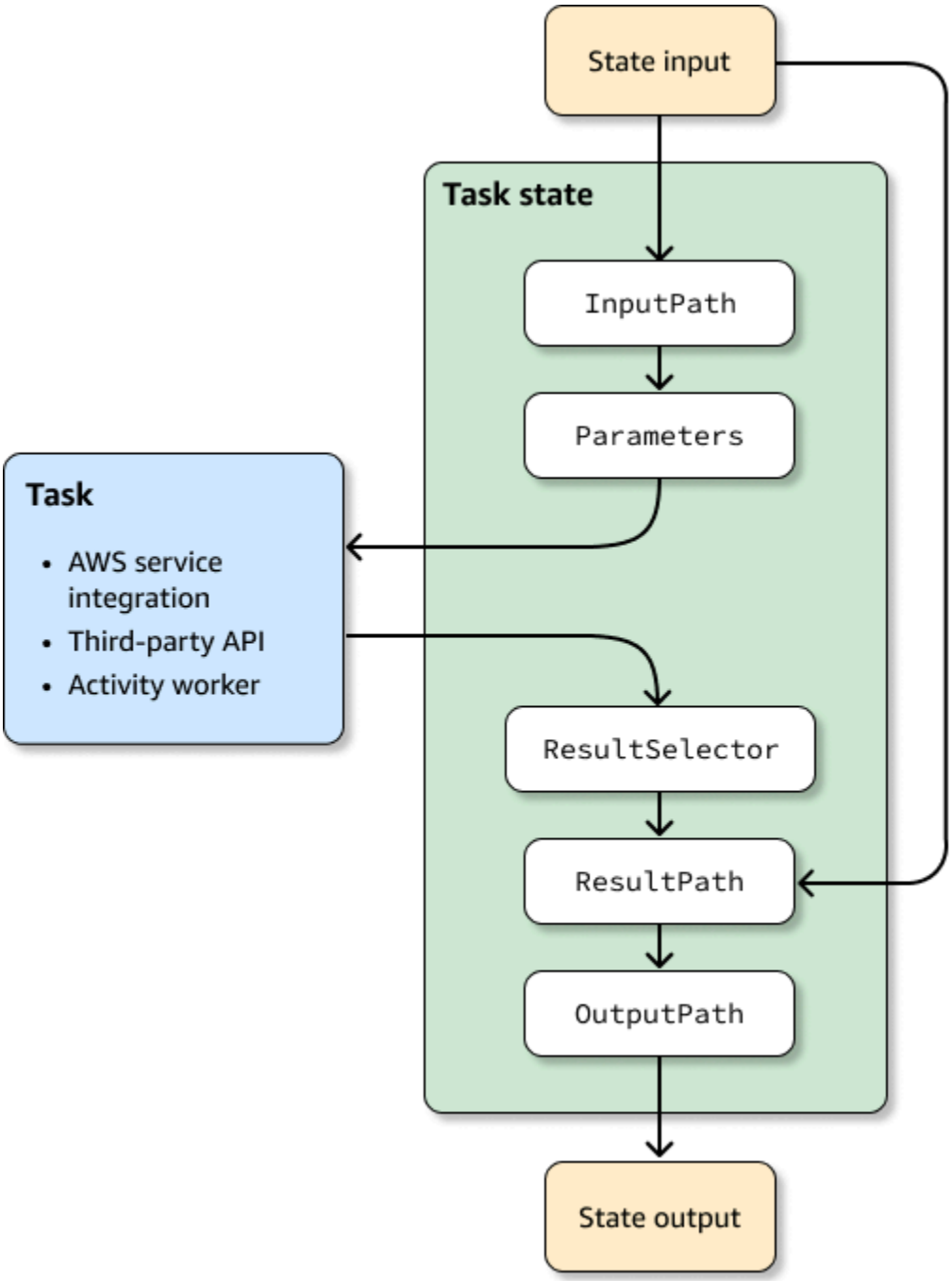
测试和调试输入和输出数据流

TestState API 有助于测试和调试流经您的工作流的数据。本节提供了一些关键概念，并说明了如何 TestState 为此目的使用。

重要概念

在 Step Functions 中，在 JSON 数据经历状态机中的状态时对其进行筛选和处理的过程称为输入和输出处理。有关其工作方式的信息，请参阅Step Functions 中的输入和输出处理。

Amazon States Language (ASL) 中的所有状态类型（Task、Parallel、Map、Pass、Wait、Choice、Succeed 和 Fail）共用一组公共字段，来筛选和处理经历它们的 JSON 数据。这些字段包括：InputPath、参数、ResultSelector、ResultPath 和 OutputPath。对每个字段的支持因状态而异。在运行时，Step Functions 按特定顺序应用每个字段。下图显示了在 Task 状态内这些字段应用于数据的顺序：



以下列表说明了图中显示的输入和输出处理字段的应用顺序。

1. 状态输入是从先前状态传递到当前状态的 JSON 数据。
2. InputPath 筛选部分原始状态输入。
3. 参数 配置要传递给 Task 的一组值。
4. 任务执行工作并返回结果。
5. ResultSelector 从任务结果中选择一组要保留的值。
6. ResultPath 将结果与原始状态输入合并，或者用它替换结果。
7. OutputPath 筛选输出的一部分以传递到下一状态。
8. 状态输出是从当前状态传递到下一状态的 JSON 数据。

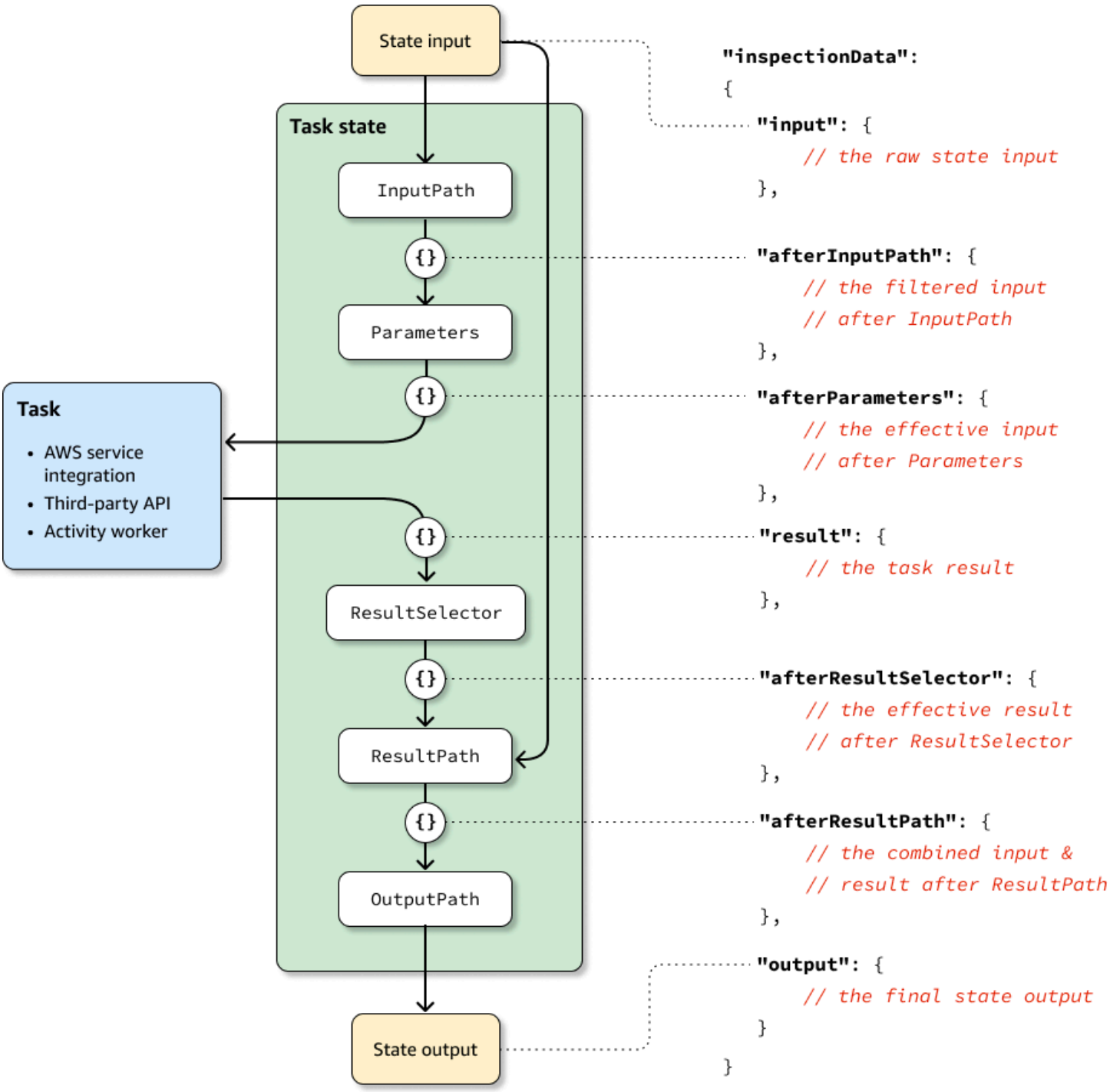
这些输入和输出处理字段是可选的。如果您在状态定义中未使用这些字段中的任何一个，则任务将使用原始状态输入，并将任务结果作为状态输出返回。

TestState 用于检查输入和输出处理

当您调用 `TestState` API 并将 `inspectionLevel` 参数设置为 `DEBUG` 时，API 响应会包含一个名为 `inspectionData` 的对象。此对象包含一些字段，可帮助您检查在相应状态执行时，数据是如何筛选或处理的。以下示例显示了 Task 状态的 `inspectionData` 对象。

```
"inspectionData":   {
  "input": string,
  "afterInputPath": string,
  "afterParameters": string,
  "result": string,
  "afterResultSelector": string,
  "afterResultPath": string,
  "output": string
}
```

在此示例中，每个包含 `after` 前缀的字段都显示特定字段应用后的数据。例如，`afterInputPath` 显示应用 `InputPath` 字段以筛选原始状态输入的效果。下图将每个 ASL 定义字段映射到 `inspectionData` 对象中的相应字段：



有关使用 TestState API 调试输入和输出处理的示例，请参阅以下内容：

- 在 Step Functions 控制台中使用调试检查级别测试状态
- 使用中的 DEBUG 检查级别测试状态 AWS CLI

在本地测试状态机

AWS Step Functions Local 是 Step Functions 的可下载版本，它允许您使用在自己的开发环境中运行的 Step Functions 版本来开发和测试应用程序。Step Functions 的本地版本可以在 AWS 中或本地运行时调用 AWS Lambda 函数。您还可以协调其他支持的 AWS 服务。

> **Note**
>
> Step Functions Local 使用虚拟账户来工作。

在运行 Step Functions Local 时，您可以使用以下方法之一来调用服务集成：

- 为 AWS Lambda 和其他服务配置本地端点。有关支持的端点的信息，请参阅为 Step Functions Local 设置配置选项。
- 从 Step Functions Local 直接调用 AWS 服务。
- 模拟服务集成的响应。有关使用模拟服务集成的信息，请参阅使用模拟服务集成。

AWS Step Functions Local 以 JAR 包或独立的 Docker 映像的形式提供，可在 Microsoft Windows、Linux、macOS 和其他支持 Java 或 Docker 的平台上运行。

> **Warning**
>
> AWS Step Functions 的可下载版本仅用于测试，不得用于处理敏感信息。

> **Tip**
>
> 确保您使用的是 Step Functions Local 1.12.0 或更高版本，以便能够在工作流中包含所有内置函数。

以下主题描述了如何使用 Docker 和 JAR 文件设置 Step Functions Local，以及如何运行 Step Functions Local 与 AWS Lambda、AWS Serverless Application Model (AWS SAM) CLI Local 或其他支持的服务协同工作。

主题

- 设置 Step Functions Local（可下载版本）和 Docker

- 设置 Step Functions Local（可下载版本）- Java 版本
- 为 Step Functions Local 设置配置选项
- 在计算机上运行 Step Functions Local
- 测试 Step Function 和 AWS SAM CLI Local
- 使用模拟服务集成

设置 Step Functions Local（可下载版本）和 Docker

利用 Step Functions Local Docker 映像，您可以通过使用包含所有必要的依赖项的 Docker 映像，来快速开始使用 Step Functions Local。Docker 映像允许您在容器化工作版本中包含 Step Functions Local，将其作为持续集成测试的一部分。

要获取 Step Functions Local 的 Docker 映像，请参阅 https://hub.docker.com/r/amazon/aws-stepfunctions-local，或者输入 Docker `pull` 命令。

```
docker pull amazon/aws-stepfunctions-local
```

要在 Docker 上启动 Step Functions 的可下载版本，请运行以下 Docker `run`命令。

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

要与 AWS Lambda 或支持的其他服务交互，您需要先配置您的凭证和其他配置选项。有关更多信息，请参阅以下主题：

- 为 Step Functions Local 设置配置选项
- Docker 的凭证和配置

设置 Step Functions Local（可下载版本）- Java 版本

AWS Step Functions 的可下载版本作为可执行 JAR 文件或 Docker 映像提供。Java 应用程序将在 Windows、Linux、macOS 和其他支持 Java 的平台上运行。除了 Java 之外，您还需要安装 AWS Command Line Interface (AWS CLI)。有关安装和配置 AWS CLI 的信息，请参阅 AWS Command Line Interface 用户指南。

在您的计算机上设置并运行 Step Functions

1. 使用以下链接下载 Step Functions。

下载链接	校验和
.tar.gz	.tar.gz.md5
.zip	.zip.md5

2. 解压缩 `.zip` 文件。
3. 测试下载并查看版本信息。

```
$ java -jar StepFunctionsLocal.jar -v
Step Function Local
Version: 1.0.0
Build: 2019-01-21
```

4. （可选）查看可用命令的列表。

```
$ java -jar StepFunctionsLocal.jar -h
```

5. 要在计算机上启动 Step Functions，请打开命令提示符，导航到您提取 `StepFunctionsLocal.jar` 的目录，并输入以下命令。

```
java -jar StepFunctionsLocal.jar
```

6. 要访问本地运行的 Step Functions，请使用 `--endpoint-url` 参数。例如，使用 AWS CLI，您需要按以下方式指定 Step Functions 命令：

```
aws stepfunctions --endpoint-url http://localhost:8083 command
```

> **Note**
>
> 默认情况下，Step Functions Local 使用本地测试账户和凭证，AWS 区域设置为美国东部（弗吉尼亚州北部）。要将 Step Functions Local 与 AWS Lambda 或支持的其他服务结合使用，您必须配置您的凭证和区域。
> 如果将快速工作流与 Step Functions Local 结合使用，则执行历史记录将存储在日志文件中。执行历史记录不会记录到 CloudWatch Logs 中。该日志文件的路径将取决于创建本地状态计算机时提供的 CloudWatch Logs 日志组 ARN。该日志文件将存储在 `/aws/states/log-`

group-name/*${execution_arn}*.log 中，此路径相对于运行 Step Functions Local 的位置确定。例如，如果执行 ARN 为：

```
arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI
```

该日志文件将为：

```
aws/states/log-group-name/arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI.log
```

为 Step Functions Local 设置配置选项

使用 JAR 文件启动 AWS Step Functions Local 时，可以使用 AWS Command Line Interface (AWS CLI) 来设置配置选项，也可以通过在系统环境中包含这些选项来设置。对于 Docker，您必须在启动 Step Functions Local 时引用的文件中指定这些选项。

配置选项

将 Step Functions Local 容器配置为使用 Lambda 端点和 Batch 端点等覆盖端点并调用该端点时，Step Functions Local 不会使用您指定的凭证。设置这些端点覆盖是可选的。

选项	命令行	环境
账户	-account, --aws-account	AWS_ACCOUNT_ID
区域	-region, --aws-region	AWS_DEFAULT_REGION
等待时间比例	-waitTimeScale, --wait-time-scale	WAIT_TIME_SCALE
Lambda 端点	-lambdaEndpoint, --lambda-endpoint	LAMBDA_ENDPOINT
Batch 端点	-batchEndpoint, --batch-endpoint	BATCH_ENDPOINT

选项	命令行	环境
DynamoDB 端点	-dynamoDBEndpoint, --dynamodb-endpoint	DYNAMODB_ENDPOINT
ECS 端点	-ecsEndpoint, --ecs-endpoint	ECS_ENDPOINT
Glue 端点	-glueEndpoint,--glue-endpoint	GLUE_ENDPOINT
SageMaker 端点	-sageMakerEndpoint,--sagemaker-endpoint	SAGE_MAKER_ENDPOINT
SQS 端点	-sqsEndpoint,--sqs-endpoint	SQS_ENDPOINT
SNS 端点	-snsEndpoint,--sns-endpoint	SNS_ENDPOINT
Step Functions 端点	-stepFunctionsEndpoint, --step-functions-endpoint	STEP_FUNCTIONS_ENDPOINT

Docker 的凭证和配置

要为 Docker 配置 Step Functions Local，请创建以下文件：`aws-stepfunctions-local-credentials.txt`。

此文件包含您的凭证以及其他配置选项。创建 `aws-stepfunctions-local-credentials.txt` 文件时，以下内容可用作模板。

```
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
```

在 aws-stepfunctions-local-credentials.txt 中配置您的凭证和配置选项后，使用以下命令启动 Step Functions。

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

> **Note**
>
> 建议使用特殊的 DNS 名称 host.docker.internal，该名称可解析为主机使用的内部 IP 地址，例如 http://host.docker.internal:8000。有关更多信息，请分别参阅 Mac 版 Docker 文档和 Windows 版 Docker 文档：Networking features in Docker Desktop for Mac 和 Networking features in Docker Desktop for Windows。

在计算机上运行 Step Functions Local

使用 Step Functions 的本地版本配置、开发和测试计算机上的状态机。

在本地运行 HelloWorld 状态机

在本地使用 AWS Command Line Interface (AWS CLI) 运行 Step Functions 后，您可以启动状态机执行。

1. 在 AWS CLI 中，通过转义状态机定义来创建状态机。

```
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` 不用于 Step Functions Local，但您必须使用正确的语法包含它。您可以使用前面示例中的 Amazon 资源名称 (ARN)。

如果您成功创建状态机，Step Functions 将使用创建日期和状态机 ARN 做出响应。

```
{
    "creationDate": 1548454198.202,
    "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld"
}
```

2. 使用创建的状态机的 ARN 启动执行。

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

Step Functions Local 与 AWS SAM CLI Local

您可以将 Step Functions 的本地版本与 AWS Lambda 的本地版本结合使用。要对此进行配置，您必须安装和配置 AWS SAM。

有关配置和运行 AWS SAM 的信息，请参阅以下内容：

- 设置 AWS SAM
- 启动 AWS SAM CLI Local

在您的本地系统上运行 Lambda 后，您可以启动 Step Functions Local。从提取 Step Functions 本地 JAR 文件的目录中，启动 Step Functions Local，然后使用 `--lambda-endpoint` 参数配置本地 Lambda 端点。

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

有关将 Step Functions Local 与 AWS Lambda 一起运行的更多信息，请参阅测试 Step Function 和 AWS SAM CLI Local。

测试 Step Function 和 AWS SAM CLI Local

通过在本地计算机上将 AWS Step Functions 与 AWS Lambda 一起运行，可以测试您的状态机和 Lambda 函数，而无需将您的代码部署到 AWS。

有关更多信息，请参阅以下主题：

- 在本地测试状态机
- 设置 AWS SAM

主题

- 第 1 步：设置 AWS SAM
- 第 2 步：测试 AWS SAM CLI Local
- 第 3 步：启动 AWS SAM CLI Local
- 第 4 步：启动 Step Functions Local
- 第 5 步：创建引用您的 AWS SAM CLI Local 函数的状态机
- 第 6 步：启动本地状态机执行

第 1 步：设置 AWS SAM

AWS Serverless Application Model (AWS SAM) CLI Local 要求安装 AWS Command Line Interface、AWS SAM 和 Docker。

1. 安装 AWS SAM CLI。

   > **Note**
   >
   > 在安装 AWS SAM CLI 之前，您需要先安装 AWS CLI 和 Docker。参阅安装 AWS SAM CLI 的先决条件。

2. 通读 AWS SAM 快速入门文档。请务必按照以下步骤执行以下操作：

   1. 初始化应用程序
   2. 在本地测试应用程序

这将创建 sam-app 目录，并将生成环境，其中包括基于 Python 语言的 Hello World Lambda 函数。

第 2 步：测试 AWS SAM CLI Local

现在，您已安装了 AWS SAM 并创建了 Hello World Lambda 函数，接下来进行测试。在 sam-app 目录中，输入下面的命令：

```
sam local start-api
```

这将启动 Lambda 函数的本地实例。您应该可以看到类似于如下所示的输出内容：

```
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)
```

打开浏览器并输入以下内容：

```
http://127.0.0.1:3000/hello
```

这将输出以下类似响应：

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

按 CTRL+C 以终止 Lambda API。

第 3 步：启动 AWS SAM CLI Local

现在，您已测试该函数可正常工作，接下来启动 AWS SAM CLI Local。在 sam-app 目录中，输入下面的命令：

```
sam local start-lambda
```

这将启动 AWS SAM CLI Local 并提供要使用的端点，类似于以下输出：

```
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)
```

第 4 步：启动 Step Functions Local

JAR 文件

如果使用 `.jar` 文件版本的 Step Functions Local，请启动 Step Functions 并指定 Lambda 端点。在提取 `.jar` 文件的目录中，输入以下命令：

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

当 Step Functions Local 启动时，它将检查环境，然后检查 `~/.aws/credentials` 文件中配置的凭证。默认情况下，它开始使用虚假用户 ID，并列出为 `region us-east-1`。

```
2019-01-29 15:38:06.324: Failed to load credentials from environment because Unable to
 load AWS credentials from environment variables (AWS_ACCESS_KEY_ID (or AWS_ACCESS_KEY)
 and AWS_SECRET_KEY (or AWS_SECRET_ACCESS_KEY))
2019-01-29 15:38:06.326: Loaded credentials from profile: default
2019-01-29 15:38:06.326: Starting server on port 8083 with account 123456789012, region
 us-east-1
```

Docker

如果您使用 Docker 版本的 Step Functions Local，请使用以下命令启动 Step Functions：

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

有关安装 Docker 版本的 Step Functions 的信息，请参阅设置 Step Functions Local（可下载版本）和 Docker。

> **Note**
>
> 您可以通过命令行指定端点；如果您从 `.jar` 文件启动 Step Functions，则可以通过设置环境变量来指定。对于 Docker 版本，您必须在文本文件中指定端点和凭证。请参阅为 Step Functions Local 设置配置选项。

第 5 步：创建引用您的 AWS SAM CLI Local 函数的状态机

在 Step Functions Local 运行后，创建一个状态机，该状态机引用您在 `HelloWorldFunction` 中初始化的 。

```
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"
```

这将创建一个状态机，并提供一个您可以用来启动执行的 Amazon 资源名称 (ARN)。

```
{
    "creationDate": 1548805711.403,
    "stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"
}
```

第 6 步：启动本地状态机执行

创建状态机后，启动执行。使用以下 **`aws stepfunctions`** 命令时，您需要引用端点和状态机 ARN：

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

这将启动 `HelloWorld` 状态机的名为 `test` 的执行。

```
{
    "startDate": 1548810641.52,
    "executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test"
```

```
}
```

现在 Step Functions 正在本地运行，您可以使用 AWS CLI 与其进行交互。例如，要获取有关此执行的信息，请使用以下命令：

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

请为执行调用 `describe-execution`，这可以提供更完整的详细信息，内容与下面的输出类似：

```
{
    "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": "{}"
}
```

使用模拟服务集成

在 Step Functions Local 中，您可以使用模拟服务集成来测试状态机的执行路径，而无需实际调用集成服务。要将状态机配置为使用模拟服务集成，请创建一个模拟配置文件。在此文件中，您可以将服务集成的所需输出定义为模拟响应，并将使用模拟响应模拟执行路径的执行定义为测试用例。

通过向 Step Functions Local 提供模拟配置文件，您可以通过运行使用测试用例中指定模拟响应的状态机来测试服务集成调用，而无需进行实际的服务集成调用。

> **Note**
>
> 如果您未在模拟配置文件中指定模拟服务集成响应，Step Functions Local 将使用您在设置 Step Functions Local 时配置的端点调用 AWS 服务集成。有关为 Step Functions Local 配置端点的信息，请参阅为 Step Functions Local 设置配置选项。

主题

- 本主题中的关键概念
- 第 1 步：在模拟配置文件中指定模拟服务集成
- 第 2 步：向 Step Functions Local 提供模拟配置文件
- 第 3 步：运行模拟服务集成测试
- 模拟服务集成的配置文件

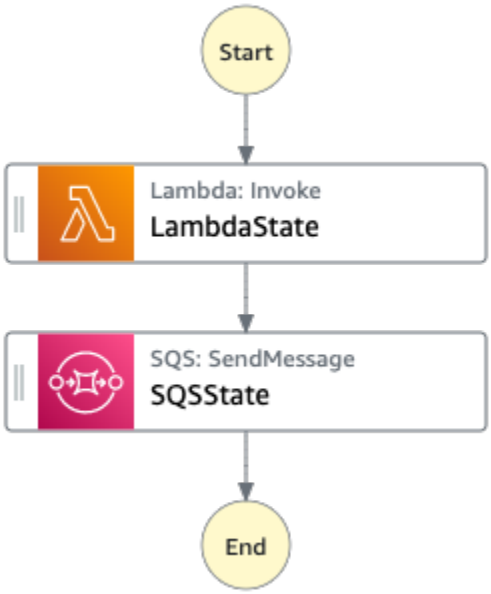
本主题中的关键概念

本主题使用以下列表中定义的几个概念：

- 模拟服务集成 - 指配置为使用模拟响应而非执行实际服务调用的 Task 状态。
- 模拟响应 - 指 Task 状态可以配置为使用的模拟数据。
- 测试用例 - 指配置为使用模拟服务集成的状态机执行。
- 模拟配置文件 - 指包含 JSON 的模拟配置文件，该文件定义了模拟服务集成、模拟响应和测试用例。

第 1 步：在模拟配置文件中指定模拟服务集成

你可以使用 Step Functions Local 测试 Step Functions AWS SDK 和优化的服务集成。下图展示了在状态机定义选项卡中定义的状态机：



为此，您必须创建一个模拟配置文件，其中包含模拟配置结构简介中定义的部分。

1. 创建一个名为 MockConfigFile.json 的文件，以便使用模拟服务集成配置测试。

   以下示例展示了一个引用状态机的模拟配置文件，该状态机有两个已定义的状态，分别名为 LambdaState 和 SQSState。

   Mock configuration file example

   以下是一个模拟配置文件示例，演示了如何模拟调用 Lambda 函数并向 Amazon SQS 发送消息所产生的响应。在此示例中，LambdaSQSIntegration 状态机包含三个测试用例，分别名为 HappyPath、RetryPath 和 HybridPath，它们模拟名为 LambdaState 和 SQSState 的 Task 状态。这些状态使用 MockedLambdaSuccess、MockedSQSSuccess 和 MockedLambdaRetry 模拟服务响应。这些模拟服务响应在文件 MockedResponses 部分中定义。

```
{
  "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

下面是一个名为 LambdaSQSIntegration 的状态机定义示例，它定义了两个名为 LambdaState 和 SQSState 的服务集成任务状态。LambdaState 包含基于 States.ALL 的重试策略。

```
{
  "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
    }
  }
}
```

您可以使用以下测试用例之一运行模拟配置文件中引用的 LambdaSQSIntegration 状态机定义：

- HappyPath - 该测试分别使用 MockedLambdaSuccess 和 MockedSQSSuccess 模拟 LambdaState 和 SQSState 的输出。
  - LambdaState 将返回以下值：

```
"0":{
  "Return":{
    "StatusCode":200,
    "Payload":{
      "StatusCode":200,
      "body":"Hello from Lambda!"
    }
  }
}
```

  - SQSState 将返回以下值：

```
"0":{
  "Return":{
    "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
    "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
  }
}
```

- RetryPath - 该测试分别使用 MockedLambdaRetry 和 MockedSQSSuccess 模拟 LambdaState 和 SQSState 的输出。此外，LambdaState 被配置为执行四次重试尝试。这些尝试的模拟响应在 MockedLambdaRetry 状态中进行了定义和索引。
  - 原始尝试以任务失败结束，其中包含原因和错误消息，如以下示例所示：

```
"0":{
  "Throw": {
    "Error": "Lambda.ResourceNotReadyException",
    "Cause": "Lambda resource is not ready."
  }
}
```

  - 第一次和第二次重试都以任务失败告终，其中包含原因和错误消息，如以下示例所示：

```
"1-2":{
  "Throw": {
    "Error": "Lambda.TimeoutException",
    "Cause": "Lambda timed out."
  }
}
```

- 第三次重试以任务成功结束，其中包含来自模拟 Lambda 响应中的“有效负载”部分的状态结果。

```
"3":{
  "Return": {
    "StatusCode": 200,
    "Payload": {
      "StatusCode": 200,
      "body": "Hello from Lambda!"
    }
  }
}
```

> **Note**
>
> - 对于采用重试策略的状态，Step Functions Local 将用尽策略中设置的重试次数，直到收到成功响应。这意味着必须用连续的尝试次数来表示重试模拟，并且应该在返回成功响应之前涵盖所有的重试尝试。
> - 如果您没有为具体的重试尝试次数指定模拟响应，例如重试 “3”，则状态机执行将失败。

- `HybridPath` - 此测试模拟 `LambdaState` 的输出。`LambdaState` 成功运行并收到模拟数据作为响应后，`SQSState` 对生产中指定的资源执行实际服务调用。

有关如何使用模拟服务集成开始测试执行的信息，请参阅第 3 步：运行模拟服务集成测试。

2. 确保模拟响应的结构符合您在调用集成服务时收到的实际服务响应的结构。有关模拟响应的结构要求信息，请参阅配置模拟服务集成。

   在前面的示例模拟配置文件中，`MockedLambdaSuccess` 和 `MockedLambdaRetry` 中定义的模拟响应与调用 `HelloFromLambda` 时返回的实际响应结构一致。

> **Important**
>
> AWS 服务不同，响应结构可能会不同。Step Functions Local 不会验证模拟响应结构是否符合实际的服务响应结构。在测试之前，您必须确保模拟响应符合实际响应。要查看服务响应的结构，您可以使用 Step Functions 执行实际的服务调用，也可以查看这些服务的文档。

第 2 步：向 Step Functions Local 提供模拟配置文件

您可以通过以下方式将模拟配置文件提供给 Step Functions Local：

Docker

> **Note**
>
> 如果您使用的是 Docker 版本的 Step Functions Local，则只能使用环境变量提供模拟配置文件。此外，在服务器初始启动时，必须将模拟配置文件挂载到 Step Functions Local 容器上。

将模拟配置文件挂载到 Step Functions Local 容器内的任何目录上。然后，设置一个名为 `SFN_MOCK_CONFIG` 的环境变量，其中包含容器中模拟配置文件的路径。此方法允许将模拟配置文件命名为任何名称，只要环境变量包含文件路径和名称即可。

以下命令展示了启动 Docker 映像的格式。

```
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
```

以下示例使用该命令启动了 Docker 映像。

```
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

使用以下任一方式将模拟配置文件提供给 Step Functions Local：

- 将模拟配置文件放在与 `Step FunctionsLocal.jar` 相同的目录中。使用此方法时，必须将模拟配置文件命名为 `MockConfigFile.json`。
- 在运行 Step Functions Local 的会话中，将名为 `SFN_MOCK_CONFIG` 的环境变量设置为模拟配置文件的完整路径。此方法允许将模拟配置文件命名为任何名称，只要环境变量包含其文件路径和名称即可。在以下示例中，`SFN_MOCK_CONFIG` 变量设置为指向名为 `EnvSpecifiedMockConfig.json` 的模拟配置文件，文件位于 `/home/workspace` 目录中。

```
export SFN_MOCK_CONFIG="/home/workspace/EnvSpecifiedMockConfig.json"
```

> **Note**
>
> - 如果您没有向 Step Functions Local 提供环境变量 `SFN_MOCK_CONFIG`，则默认情况下，它将尝试读取在启动 Step Functions Local 的目录中名为 `MockConfigFile.json` 的模拟配置文件。
> - 如果将模拟配置文件放在与 `Step FunctionsLocal.jar` 相同的目录中，并设置环境变量 `SFN_MOCK_CONFIG`，则 Step Functions Local 将读取由环境变量指定的文件。

第 3 步：运行模拟服务集成测试

创建模拟配置文件并将其提供给 Step Functions Local 后，使用模拟服务集成运行模拟配置文件中配置的状态机。然后使用 API 操作检查执行结果。

1. 根据前面提到的模拟配置文件中的定义创建状态机。

```
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. 使用模拟服务集成运行状态机。

   要使用模拟配置文件，请在模拟配置文件中配置的状态机上进行 StartExecution API 调用。为此，请在 StartExecution 使用的状态机 ARN 上添加后缀 *#test_name*。*test_name* 是一个测试用例，在同一个模拟配置文件中为状态机配置。

   下面的命令是一个使用 LambdaSQSIntegration 状态机和模拟配置的示例。在此示例中，使用第 1 步：在模拟配置文件中指定模拟服务集成中定义的 HappyPath 测试来执行 LambdaSQSIntegration 状态机。HappyPath 测试包含执行配置，用于处理 LambdaState 和 SQSState 状态使用 MockedLambdaSuccess 和 MockedSQSSuccess 模拟服务响应进行的模拟服务集成调用。

   ```
   aws stepfunctions start-execution \
       --endpoint http://localhost:8083 \
       --name executionWithHappyPathMockedServices \
       --state-machine arn:aws:states:us-
   east-1:123456789012:stateMachine:LambdaSQSIntegration#HappyPath
   ```

3. 查看状态机执行响应。

   使用模拟服务集成测试调用 StartExecution 的响应与正常调用 StartExecution 的响应相同，都会返回执行 ARN 和开始日期。

   以下是使用模拟服务集成测试调用 StartExecution 的响应示例：

   ```
   {
      "startDate":"2022-01-28T15:03:16.981000-05:00",
      "executionArn":"arn:aws:states:us-
   east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices"
   }
   ```

4. 通过调用 ListExecutions、DescribeExecution 或 GetExecutionHistory API 来检查执行结果。

```
aws stepfunctions get-execution-history \
    --endpoint http://localhost:8083 \
    --execution-arn arn:aws:states:us-
east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices
```

以下示例演示了使用第 2 步所示示例响应中的执行 ARN 调用 GetExecutionHistory 的部分响应。在此示例中，LambdaState 和 SQSState 的输出是模拟配置文件中 MockedLambdaSuccess 和 MockedSQSSuccess 中定义的模拟数据。此外，模拟数据的使用方式与执行实际服务集成调用返回的数据相同。此外，在本示例中，LambdaState 的输出作为输入传递给 SQSState。

```
{
    "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
                }
            }
        },
        ...
    ]
}
```

模拟服务集成的配置文件

要使用模拟服务集成，必须先创建一个名为 `MockConfigFile.json` 的模拟配置文件，其中包含您的模拟配置。然后将模拟配置文件提供给 Step Functions Local。此配置文件定义了测试用例，其中包含使用模拟服务集成响应的模拟状态。下一节将介绍包含模拟状态和模拟响应的模拟配置结构：

主题

- 模拟配置结构简介

- 配置模拟服务集成

模拟配置结构简介

模拟配置是一个 JSON 对象，包含以下顶级字段：

- `StateMachines` - 该对象的字段表示配置为使用模拟服务集成的状态机。
- `MockedResponse` - 该对象的字段表示服务集成调用的模拟响应。

以下是一个模拟配置文件示例，其中包含 `StateMachine` 定义和 `MockedResponse`。

```
{
  "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!"
          }
        }
      }
    }
  }
}
```

模拟配置字段参考

以下部分将说明必须在模拟配置中定义的顶层对象字段。

- StateMachines
- MockedRespons

StateMachines

StateMachines 对象定义哪些状态机将使用模拟服务集成。每个状态机的配置都用 StateMachines 的顶级字段表示。字段名称是状态机的名称，值是包含名为 TestCases 的单个字段的对象，其字段代表该状态机的测试用例。

以下语法显示了带有两个测试用例的状态机：

```
"MyStateMachine": {
  "TestCases": {
    "HappyPath": {
      ...
    },
    "SadPath": {
      ...
    }
  }
```

测试用例

TestCases 字段表示状态机的各个测试用例。每个状态机的测试用例名称必须是唯一的，每个测试用例的值都是一个对象，指定了状态机中 Task 状态使用的模拟响应。

以下 TestCase 示例将两个 Task 状态与两个 MockedResponses 联系起来：

```
"HappyPath": {
  "SomeTaskState": "SomeMockedResponse",
  "AnotherTaskState": "AnotherMockedResponse"
}
```

MockedRespons

MockedResponses 是一个对象，包含多个具有唯一字段名称的模拟响应对象。模拟响应对象定义了每次调用模拟 Task 状态时的成功结果或错误输出。您可以使用单个整数字符串（例如 “0”、“1”、“2” 和 “3”）或包含整数的范围（例如 “0-1”、“2-3”）来指定调用次数。

模拟 Task 时，必须为每次调用指定一个模拟响应。响应必须包含一个名为 Return 或 Throw 的单个字段，其值为模拟 Task 调用的结果或错误输出。如果未指定模拟响应，则状态机执行将失败。

下面是一个包含 Throw 和 Return 对象的 MockedResponse 示例。在此示例中，状态机运行的前三次返回 "0-2" 中指定的响应，状态机运行的第四次返回 "3" 中指定的响应。

```
"SomeMockedResponse": {
  "0-2": {
    "Throw": {
      ...
    }
  },
  "3": {
    "Return": {
      ...
    }
  }
}
```

> **Note**
>
> 如果使用的是 Map 状态，并且希望确保 Map 状态的响应具有可预测性，请将 maxConcurrency 的值设置为 1。如果您设置的值大于 1，Step Functions Local 将同时运行多次迭代，这将导致各迭代状态的整体执行顺序不可预测。这可能会进一步导致 Step Functions Local 在每次执行时对迭代状态使用不同的模拟响应。

Return

Return 表示为 MockedResponse 对象的一个字段。它指定了模拟 Task 状态的成功结果。

下面是一个 Return 对象的示例，其中包含在 Lambda 函数上调用 Invoke 时的模拟响应：

```
"Return": {
  "StatusCode": 200,
  "Payload": {
    "StatusCode": 200,
    "body": "Hello from Lambda!"
  }
}
```

Throw

Throw 表示为 MockedResponse 对象的一个字段。它指定失败 Task 的错误输出。Throw 的值必须是一个对象，其中包含带有字符串值的 Error 和 Cause 字段。此外，您在 MockConfigFile.json 中指定的 Error 字段的字符串值必须与状态机的 Retry 和 Catch 部分处理的错误相匹配。

下面是一个 Throw 对象的示例，其中包含在 Lambda 函数上调用 Invoke 时的模拟响应：

```
"Throw": {
  "Error": "Lambda.TimeoutException",
  "Cause": "Lambda timed out."
}
```

配置模拟服务集成

您可以使用 Step Functions Local 来模拟任何服务集成。但是，Step Functions Local 并不强制要求模拟与真实 API 相同。模拟 Task 永远不会调用服务端点。如果您未指定模拟响应，则 Task 将尝试调用服务端点。此外，使用 .waitForTaskToken 模拟 Task 时，Step Functions Local 将自动生成任务令牌。

Step Functions 的最佳实操

以下实现 AWS Step Functions 工作流程的最佳实操可帮助您优化实现性能。

主题

- 使用超时避免执行卡顿
- 使用 Amazon S3 ARN 而不是传递大量有效负载
- 避免达到历史记录的配额
- 处理 Lambda 服务异常
- 避免轮询活动任务时发生延迟
- 选择标准或快速工作流
- Amazon CloudWatch Logs 资源策略大小限制

使用超时避免执行卡顿

默认情况下，Amazon States Language 不会为状态机定义指定超时。如果没有显式超时，Step Functions 通常仅依靠来自活动工作线程的响应来了解任务是否已完成。如果发生错误并且 `Activity` 或 `Task` 状态未指定 `TimeoutSeconds` 字段，则执行会卡住，等待永远不会出现的响应。

为避免这种情况，请在状态机中创建 `Task` 时指定合理的超时。例如：

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

如果您使用使用任务令牌发出回调 (.waitForTaskToken)，我们建议您使用检测信号并在 `Task` 状态定义中添加 `HeartbeatSeconds` 字段。您可以将 `HeartbeatSeconds` 设置为小于任务超时时间，因此，如果您的工作流因检测信号错误而失败，那么您就知道这是因为任务失败，而不是任务需要很长时间才能完成。

```
{
  "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
    }
  }
}
```

有关更多信息，请参阅 Amazon States Language 文档中的 任务状态。

> **Note**
>
> 您可以使用 Amazon States Language 定义中的 `TimeoutSeconds` 字段为状态机设置超时。有关更多信息，请参阅状态机结构。

使用 Amazon S3 ARN 而不是传递大量有效负载

可以终止在状态间传递大量数据负载的执行。如果您在各状态之间传递的数据可能增长到 256 KB 以上，请使用 Amazon Simple Storage Service (Amazon S3) 存储数据，并在 `Payload` 参数中解析存储桶的 Amazon 资源名称 (ARN)，获取存储桶名称和密钥值。或者，您也可以调整实现，以便在执行中传递较少的负载。

在以下示例中，状态机将输入传递给一个 AWS Lambda 函数，该函数处理 Amazon S3 存储桶中的 JSON 文件。运行此状态机后，Lambda 函数读取 JSON 文件的内容，并将文件内容作为输出返回。

创建 Lambda 函数

以下名为 *`pass-large-payload`* 的 Lambda 函数读取存储在特定 Amazon S3 存储桶中的 JSON 文件的内容。

> **Note**
>
> 创建此 Lambda 函数后，请务必为其 IAM 角色提供相应的权限，使其能够从 Amazon S3 存储桶中读取。例如，将 AmazonS3ReadOnlyAccess 权限附加到 Lambda 函数的角色上。

```
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
```

创建状态机

以下状态机调用您之前创建的 Lambda 函数。

```
{
   "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
        }
    }
}
```

无需传递输入中的大量数据，您可以将数据存储到一个 Amazon S3 存储桶，然后在 Payload 参数中传递该存储桶的 Amazon 资源名称 (ARN)，获取存储桶的名称和密钥值。之后，您的 Lambda 函数就可以使用该 ARN 直接访问数据。以下是状态机执行的输入示例，其中数据以 *large-payload-json* 形式存储在名为 data.json 的 Amazon S3 存储桶中。

```
{
  "key": "data.json",
  "bucket": "arn:aws:s3:::large-payload-json"
}
```

避免达到历史记录的配额

AWS Step Functions 对执行事件历史记录设置了 2.5 万个条目的硬配额。当执行达到 24,999 个事件时，它会等待下一个事件发生。

- 如果事件编号为 25,000 是 ExecutionSucceeded，则执行成功完成。
- 如果事件编号 25,000 不是 ExecutionSucceeded，则会记录 ExecutionFailed 事件，状态机执行因达到历史记录上限而失败

为了避免长时间运行的执行达到此配额，您可以尝试以下一种变通方法：

- 使用分布式模式下的 Map 状态。在此模式下，Map 状态将每次迭代作为子工作流执行运行，从而实现多达 1 万个并行子工作流执行的高并发数。每个子工作流执行都有自己的、独立于父工作流的执行历史记录。
- 直接从正在执行的 Task 状态启动新的状态机执行。要启动此类嵌套工作流执行，请在父状态机中使用 Step Functions 的 StartExecution API 操作并设置必要的参数。有关使用嵌套工作流的更多信息，请参阅从 Task 状态启动工作流执行或使用 Step Functions API 操作继续新的执行教程。

> **Tip**
>
> 要将嵌套工作流的示例部署到您的 AWS 账户，请参阅模块 13 - 嵌套快速工作流。

- 实施一种使用 AWS Lambda 函数的模式为状态机启动新的执行，将持续的工作拆分到多个工作流程执行。有关更多信息，请参阅使用 Lambda 函数继续新的执行教程。

处理 Lambda 服务异常

AWS Lambda 可能会偶尔遇到暂时性服务错误。在这种情况下，调用 Lambda 会导致 500 错误，例如 `ClientExecutionTimeoutException`、`ServiceException`、`AWSLambdaException` 或 `SdkClientException`。作为最佳实操，在状态机中主动处理这些异常，以 `Retry` 调用 Lambda 函数或 `Catch` 错误。

Lambda 错误报告为 `Lambda.`*`ErrorName`*。要重试 Lambda 服务异常错误，可以使用以下 `Retry` 代码。

```
"Retry": [ {
   "ErrorEquals": [ "Lambda.ClientExecutionTimeoutException",
 "Lambda.ServiceException", "Lambda.AWSLambdaException", "Lambda.SdkClientException"],
   "IntervalSeconds": 2,
   "MaxAttempts": 6,
   "BackoffRate": 2
} ]
```

> **Note**
>
> Lambda 中未处理的错误在错误输出中报告为 `Lambda.Unknown`。这些错误包括内存不足错误和函数超时。您可以匹配 `Lambda.Unknown`、`States.ALL` 或 `States.TaskFailed` 来处理这些错误。当 Lambda 达到最大调用次数时，会出现 `Lambda.TooManyRequestsException` 错误。有关 Lambda 函数错误的更多信息，请参阅《AWS Lambda 开发者指南》中的错误处理和自动重试。

有关更多信息，请参阅下列内容：

- 出错后重试
- 使用状态机处理 Step Functions 函数错误情形
- Lambda 调用错误

避免轮询活动任务时发生延迟

GetActivityTask API 旨在仅提供一次 taskToken。如果在通过活动工作线程通信时 taskToken 被丢弃，大量 GetActivityTask 请求可能会被阻止 60 秒以等待响应，直至 GetActivityTask 超时。

如果只有少量轮询等待响应，则可能所有请求都会排在被阻止的请求之后，无法处理。但是，如果每个活动 Amazon 资源名称 (ARN) 都有大量未完成轮询，某个百分比的请求需要等待，不过更多的请求仍然可以获取 taskToken 并开始处理工作。

对于生产系统，建议每个活动 ARN 在每个时间点至少有 100 个轮询。如果一个轮询被阻止，其后有一部分轮询排队，在 GetActivityTask 请求被阻止时，仍然有更多请求将获得 taskToken 可以处理工作。

避免在轮询任务时出现这类延迟问题：

- 在活动工作线程实现的工作之外通过单独的线程实现轮询器。
- 每个活动 ARN 在每个时间点至少有 100 个轮询。

> **Note**
>
> 每个 ARN 都扩展到 100 个轮询会比较昂贵。例如，每个 ARN 100 个 Lambda 函数轮询比具有 100 个轮询线程的单个 Lambda 函数贵 100 倍。要降低延迟并 最大限度减少成本，请使用具有异步 I/O 的语言，并且每个工作线程实施多个轮询线程。有关轮询器线程独立于工作线程的示例活动工作线程，请参阅使用 Ruby 编写的示例活动工作线程。

有关活动和活动工作线程的更多信息，请参阅活动。

选择标准或快速工作流

AWS Step Functions 提供标准工作流作为默认工作流类型，并可选择“Express Workflows”(快速工作流)。

当需要长时间运行、持久且可审计的工作流时，您可以选择“标准工作流”，对于大容量事件处理工作负载，也可以选择“快速工作流”。状态机的执行方式将有所不同，具体取决于您选择的 Type。创建状态机后，无法更改您选择的 Type。

- 有关标准工作流和快速工作流之间差异的详细信息，请参阅标准和快速工作流。

- 有关在使用 Step Functions 构建无服务器工作流时优化成本的信息，请参阅使用快速工作流实现成本优化。

Amazon CloudWatch Logs 资源策略大小限制

CloudWatch Logs 资源策略限制为 5120 个字符。当 CloudWatch Logs 检测到策略接近此大小限制时，它会自动启用以 `/aws/vendedlogs/` 开头的日志组。

创建启用日志记录的状态机时，Step Functions 必须使用您指定的日志组更新 CloudWatch Logs 资源策略。为避免达到 CloudWatch Logs 资源策略大小限制，请在 CloudWatch Logs 日志组名称前加上 `/aws/vendedlogs/` 前缀。在 Step Functions 控制台中创建日志组时，日志组名称的前缀为 `/aws/vendedlogs/states`。有关更多信息，请参阅从某些 AWS 服务启用日志记录。

如果您无法访问 CloudWatch Logs，请参阅Unable to access the CloudWatch Logs 获取信息。

与其他服务 AWS Step Functions 一起使用

了解如何与其他 API 协调 AWS 服务 或调用第三方 API AWS Step Functions。

主题

- 致电其他 AWS 服务
- AWS SDK 服务集成
- Step Functions 的优化集成
- 调用第三方 API
- 服务集成模式
- 将参数传递给服务 API
- 更改支持的 AWS SDK 集成的日志

致电其他 AWS 服务

通过 AWS 服务集成，您可以直接从工作流程中调用 API 操作并协调执行。你可以使用 Step Functions 的 AWS SDK 集成直接从状态机调用 200 多个 AWS 服务中的任何一个，这样你就可以访问九千多个 API 操作。或者，您可以使用 Step Functions 的优化集成，每个集成都经过自定义，可为您的工作流提供特殊功能。有些 API 操作在这两种类型的集成中都可用。如果可能，我们建议使用优化的集成。

您可以直接从 Amazon States Language 中的 `Task` 状态协调这些服务。例如，使用 Step Functions，您可以调用其他服务用于：

- 调用一个 AWS Lambda 函数。
- 运行作 AWS Batch 业，然后根据结果执行不同的操作。
- 从 Amazon DynamoDB 插入或获取一个项目。
- 运行 Amazon Elastic Container Service (Amazon ECS) 任务并等待任务完成。
- 在 Amazon Simple Notification Service (Amazon SNS) 中发布主题。
- 在 Amazon Simple Queue Service (Amazon SQS) 中发送消息。
- 管理我们的 Amazon 的工作 SageMaker。 AWS Glue
- 构建用于执行 Amazon EMR 作业的工作流。
- 启动 AWS Step Functions 工作流程执行。

优化集成

Step Functions 对优化集成进行了自定义，为工作流环境提供特殊功能。例如，Lambda Invoke 将其 API 输出从转义 JSON 转换为 JSON 对象。AWS BatchSubmitJob 允许您暂停执行直到作业完成。第一组优化集成于 2018 年发布，目前已有五十多个 API。

AWS 软件开发工具包集成

AWS SDK 集成的工作方式与使用软件开发工具 AWS 包的标准 API 调用完全相同。它们可以直接从你的状态机定义中调用 200 多个 AWS 服务中的 900 多个 API。

集成模式支持

标准工作流程和快速工作流程支持相同的集成，但不支持相同的集成模式。

- 每种集成的优化集成模式支持都不同。
- Express Workflows 不支持 Run a Job (.sync) 或等待回调 (. waitForTask代币 ）。
- 有关更多信息，请参阅 标准和快速工作流。

Standard Workflows

支持的服务集成

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
优化集成	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS 软件开发工具包集成	两百多种	✓		✓

Express Workflows

支持的服务集成

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
优化集成	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	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS 软件开发工具包集成	两百多种	✓		

跨账户存取

Step Functions 提供对工作流程 AWS 账户 中不同配置的资源的跨账户访问权限。使用 Step Functions 服务集成，您可以调用任何跨账户 AWS 资源，即使该资源 AWS 服务 不支持基于资源的策略或跨账户调用。

有关更多信息，请参阅 在工作流程中访问其他资源 AWS 账户 中的资源。

AWS SDK 服务集成

AWS Step Functions 与集成 AWS 服务，允许您从工作流程中调用每项服务的 API 操作。你可以使用 Step Functions 的 AWS SDK 集成从状态机调用几乎所有 AWS 服务的 API 操作。您还可以使用 Step Functions 的优化集成，每个集成都经过自定义，可为您的工作流提供特殊功能。

某些服务或 SDK 可能暂时或永久无法作为 AWS SDK 集成提供。最近发布的服务可能要等到以后的更新才会提供开发工具包交互。某些服务需要自定义配置，例如指定客户特定的端点，这可能更适合优化集成。其他开发工具包不适合在工作流中使用，例如用于流式传输音频或视频的开发工具包。最后，某些服务在通过 Step Functions 执行的某些内部验证之前可能会被暂停。

> **Tip**
>
> 要将使用 AWS SDK 集成的工作流程示例部署到您 AWS 账户的，请参阅 The Workshop 中的模块 9- AWS SDK 服务集成。 AWS Step Functions

主题

- 使用 AWS SDK 服务集成
- 支持的 AWS SDK 服务集成
- 支持服务的不支持 API 操作
- 已弃用的 AWS SDK 服务集成

使用 AWS SDK 服务集成

要使用 AWS SDK 集成，您需要指定服务名称和 API 调用，也可以指定服务集成模式。

> **Note**
>
> - 中的Step Functions参数以表示 PascalCase，即使本机服务 API 采用驼峰大小写。例如，您可以使用 Step Functions API 操作 `startSyncExecution` 并将其参数指定为 `StateMachineArn`。
> - 对于接受枚举参数的 API 操作，例如适用于 Amazon EC2 的 `DescribeLaunchTemplateVersions` API 操作，请指定参数名称的复数版本。例如，为 `DescribeLaunchTemplateVersions` API 操作的 `Filter.N` 参数指定 `Filters`。

在任务状态`Resource`字段中，您可以直接从 Amazon States 语言调用 AWS SDK 服务。为此，请使用以下语法：

`arn:aws:states:::aws-sdk:`*`serviceName`*`:`*`apiAction`*`.`*`[serviceIntegrationPattern]`*

例如，对于 Amazon EC2，您可以使用 `arn:aws:states:::aws-sdk:ec2:describeInstances`。这将按照 Amazon EC2 describeInstances API 调用的定义返回输出。

如果 S AWS DK 集成遇到错误，则生成的错误字段将由服务名称和错误名称组成，并用句点字符分隔:*`ServiceName`*`.`*`ErrorName`*。服务名称和错误名称均采用 Pascal 拼写法。您还可以在 Task 状态的“资源”字段中看到小写的服务名称。您可以在目标服务的 API 参考文档中找到潜在的错误名称。

例如，您可以使用 `arn:aws:states:::aws-sdk:acmpca:deleteCertificateAuthority` AWS SDK 集成。例如，AWS Private Certificate Authority API 参考指出，`DeleteCertificateAuthority` API 操作可能导致 `ResourceNotFoundException`。因此，要处理此错误，您需要在 Task 状态的“重试器”或“捕获器”中指定错误

`AcmPca.ResourceNotFoundException`。有关 Step Functions 中错误处理的更多信息，请参阅 Step Functions 中的错误处理。

Step Functions 无法为 AWS 软件开发工具包集成自动生成 IAM 策略。创建状态机后，您需要导航到 IAM 控制台并配置您的角色策略。请参阅集成服务的 IAM 策略了解更多信息。

有关如何使用 AWS SDK 集成的示例，请参阅使用 AWS 软件开发工具包服务集成收集 Amazon S3 存储桶信息教程。

支持的 AWS SDK 服务集成

下表列出了 Step Functions 支持的 AWS SDK 服务集成。*Task* 状态资源列列出了在使用左侧服务名称列中指定的服务集成时调用特定 API 操作的语法。支持日期列提供有关支持服务集成的日期信息。此外，右侧的异常前缀列列出了每个服务集成的异常前缀。这些异常前缀存在于错误地与 Step Functions 执行 AWS SDK 服务集成时生成的异常中。

> **Note**
>
> - 标有 *** 的服务具有 Step Functions 目前不支持的 API 操作。有关服务不支持的操作信息，请参阅支持服务的不支持 API 操作表格。
> - 有关每次发布时为扩大对 AWS SDK 集成的支持而进行的更新的信息，请参阅更改支持的 AWS SDK 集成的日志。

> **Important**
>
> API 操作支持按季度发布一次。对已支持的操作（例如新参数）的更新可能不会立即可用。

服务名称	**Task** 状态资源	支持日期	异常前缀
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

服务名称	Task 状态资源	支持日期	异常前缀
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:cloudfrontkeyvaluestore: *[apiAction]*	2024 年 1 月 18 日	CloudFrontKeyValueStore
CodeGuru Security	arn:aws:states:::aws-sdk:codegurusecurity: *[apiAction]*	2024 年 1 月 18 日	CodeGuruSecurity

服务名称	Task 状态资源	支持日期	异常前缀
AWS 成本优化中心	`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 数据匹配服务	`arn:aws:states:::aws-sdk:entityresolution:` *[apiAction]*	2024 年 1 月 18 日	EntityResolution
AWS Free Tier	`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
Amazon Managed Blockchain Query	`arn:aws:states:::aws-sdk:managedblockchainquery:` *[apiAction]*	2024 年 1 月 18 日	ManagedBlockchainQuery

服务名称	Task 状态资源	支持日期	异常前缀
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
Amazon Q Connect	`arn:aws:states:::aws-sdk:qconnect: [apiAction]`	2024 年 1 月 18 日	QConnect

服务名称	Task 状态资源	支持日期	异常前缀
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:cloudtrailddata: [apiAction]`	2023 年 6 月 16 日	CloudTrailData
Amazon CloudWatch Internet Monitor	`arn:aws:states:::aws-sdk:internetmonitor: [apiAction]`	2023 年 6 月 16 日	InternetMonitor

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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:kendrarankingng: [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
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

服务名称	Task 状态资源	支持日期	异常前缀
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 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
AWS Private 5G	`arn:aws:states:::aws-sdk:privatenetworks: [apiAction]`	2023 年 2 月 17 日	PrivateNetworks
AWS 资源探索器	`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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS Billing Conductor	`arn:aws:states:::aws-sdk:billingconductor:` *[apiAction]*	2022 年 7 月 26 日	Billingconductor
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS CloudTrail	`arn:aws:states:::aws-sdk:cloudtrail:` *[apiAction]*	2021 年 9 月 30 日	CloudTrail
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS CodeStar	arn:aws:states:::aws-sdk:codestarconnections: *[apiAction]*	2021 年 9 月 30 日	CodeStarConnections
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 成本和使用情况报告	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

服务名称	Task 状态资源	支持日期	异常前缀
AWS Device Farm	arn:aws:states:::aws-sdk:devicefarm:*[apiAction]*	2021 年 9 月 30 日	DeviceFarm
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS Fault Injection Service	arn:aws:states:::aws-sdk:fis: *[apiAction]*	2021 年 9 月 30 日	Fis
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS IoT Analytics	`arn:aws:states:::aws-sdk:iotanalytics: [apiAction]`	2021 年 9 月 30 日	IoTAnalytics
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 数据	`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 任务数据 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

服务名称	Task 状态资源	支持日期	异常前缀
AWS IoT Wireless	`arn:aws:states:::aws-sdk:iotwireless: [apiAction]`	2021 年 9 月 30 日	IotWireless
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

服务名称	Task 状态资源	支持日期	异常前缀
AWS Elemental MediaTailor	arn:aws:states:::aws-sdk:mediatailor: *[apiAction]*	2021 年 9 月 30 日	MediaTailor
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 策略建议	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

服务名称	Task 状态资源	支持日期	异常前缀
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 价目表	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

服务名称	Task 状态资源	支持日期	异常前缀
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
AWS Service Catalog	arn:aws:states:::aws-sdk:servicecatalog: *[apiAction]*	2021 年 9 月 30 日	ServiceCatalog

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
Amazon Augmented AI	`arn:aws:states:::aws-sdk:sagemakera2iruntime:` *[apiAction]*	2021 年 9 月 30 日	SageMakerA2IRuntime
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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	SageMakerRuntimeFeatureStoreRuntime
Amazon SageMaker Runtime Runtime	arn:aws:states:::aws-sdk:sagemakerruntime: *[apiAction]*	2021 年 9 月 30 日	SageMakerRuntimeRuntime
Amazon SageMaker	arn:aws:states:::aws-sdk:sagemaker: *[apiAction]*	2021 年 9 月 30 日	SageMakerRuntime

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

服务名称	Task 状态资源	支持日期	异常前缀
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
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

支持服务的不支持 API 操作

下表列出了不支持的 AWS SDK 服务集成的 API 操作。右列包含左列中的服务当前不支持的 API 操作。

服务名称	步支持的 API 操作
AWS Application Discovery Service	• DescribeExportConfigurations • ExportConfigurations
Amazon Bedrock	• InvokeModelWithResponseStream
亚马逊 Bedrock Runtime 的代理	• InvokeAgent
AWS CodeDeploy	• BatchGetDeploymentInstances • GetDeploymentInstance • ListDeploymentInstances • 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

服务名称	步支持的 API 操作
AWS IoT 核心设备顾问	• `ListTestCases`
Amazon Kinesis	• `SubscribeToShard`
AWS Lambda	• `InvokeAsync` • `InvokeWithResponseStream`
Amazon Lex Runtime V2	• `StartConversation`
AWS Elemental MediaPackage	• `RotateChannelCredentials`
Amazon Relational Database Service	• `ExecuteSql`
Amazon Simple Storage Service	• `SelectObjectContent`
Amazon S3 控件	• `SelectObjectContent`
AWS Shield	• `DeleteSubscription`
AWS Security Token Service	• `AssumeRole` • `AssumeRoleWithSAML` • `AssumeRoleWithWebIdentity`

已弃用的 AWS SDK 服务集成

以下 AWS SDK 服务集成现已弃用：

- AWS Mobile
- Amazon Macie
- AWS IoT RoboRunner

Step Functions 的优化集成

以下主题包括用于协调其他 AWS 服务的亚马逊州语言中支持的 API、参数和请求/响应语法。这些主题还提供了示例代码。您可以在 `Task` 状态的 `Resource` 字段中直接从 Amazon States Language 调用优化集成服务。

您可以使用三种服务集成模式：

- 请求响应（默认）-等待 HTTP 响应，然后进入下一个状态
- Run a Job (`.sync`)-等待任务完成
- 等待 Callback (`.waitForTaskToken`)-暂停工作流程，直到返回任务令牌

标准工作流程和快速工作流程支持相同的集成，但不支持相同的集成模式。

- 每种集成的优化集成模式支持都不同。
- Express Workflows 不支持 Run a Job (.sync) 或等待回调 (. waitForTask代币）。
- 有关更多信息，请参阅 标准和快速工作流。

Standard Workflows

支持的服务集成

	服务	请求响应	运行作业 (`.sync`)	等待回调 (`.waitForTaskToken`)
优化集成	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS 软件开发工具包集成	两百多种	✓		✓

Express Workflows

支持的服务集成

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
优化集成	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	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		

	服务	请求响应	运行作业 (.sync)	等待回调 (.waitForTaskToken)
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS 软件开发工具包集成	两百多种	✓		

使用 Step Functions 调用 API Gateway

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化后的 API Gateway 集成与 API Gateway AWS SDK 集成有何不同
>
> - `apigateway:invoke`:在 AWS SDK 服务集成中没有等效项。相反，优化的 API Gateway 服务可以直接调用您的 API Gateway 端点。

您可以使用 Amazon API Gateway 创建、发布、维护和监控 HTTP 和 REST API。要与 API Gateway 集成，您可以在步骤功能中定义一个 `Task` 状态，直接调用 API Gateway HTTP 或 API Gateway REST 端点，而无需编写代码或依赖其他基础架构。`Task` 状态定义包括 API 调用的所有必要信息。您也可以选择不同的授权方法。

API Gateway 特征支持

Step Functions API Gateway 集成支持部分 API 特征，而非全部。有关支持特征的详细信息，请参阅以下内容。

- Step Functions API Gateway REST API 和 API Gateway HTTP API 集成都支持：
  - 授权方：IAM（使用签名版本 4）、No Auth、Lambda 授权方（基于请求参数和基于令牌，带有自定义标头）

  - API 类型：区域性
  - API 管理：API Gateway API 域名、API 阶段、路径、查询参数、请求正文
- 由 Step Functions API Gateway HTTP API 集成支持。不支持为边缘优化 API 提供选项的 Step Functions API Gateway REST API 集成。
- Step Functions API Gateway 集成不支持：
  - 授权方：Amazon Cognito、Native Open ID Connect/OAuth 2.0、基于令牌的 Lambda 授权者的授权标头
  - API 类型：私有
  - API 管理：自定义域名

有关 API Gateway 及其 HTTP 和 REST API 的更多信息，请参阅以下内容。

- Amazon API Gateway 概念页面。
- API Gateway 开发人员指南种的在 HTTP API 和 REST API 之间进行选择。

请求格式

在创建 `Task` 状态定义时，Step Functions 会验证参数，构建执行调用所需的 URL，然后调用 API。响应包括 HTTP 状态代码、标头和响应正文。请求格式包括必需和可选参数。

必需请求参数

- `ApiEndpoint`
  - 类型：`String`
  - API Gateway URL 的主机名。格式为 *`<API ID>`*`.execute-api.`*`<region>`*`.amazonaws.com`。

    API ID 只能包含以下字母数字字符的组合：`0123456789abcdefghijklmnopqrstuvwxyz`
- `Method`
  - 类型：`Enum`
  - HTTP 方法，必须是以下方法之一：
    - `GET`
    - `POST`
    - `PUT`

- DELETE
- PATCH
- HEAD
- OPTIONS

可选请求参数

- Headers
  - 类型：JSON
  - HTTP 标头允许列出与同一密钥关联的值。
- Stage
  - 类型：String
  - API 部署到 API Gateway 的阶段名称。对于使用 $default 阶段的任何 HTTP API 来说，它都是可选选项。
- Path
  - 类型：String
  - 附加在 API 端点之后的路径参数。
- QueryParameters
  - 类型：JSON
  - 查询字符串仅允许列出与相同键关联的值。
- RequestBody
  - 类型：JSON 或 String
  - HTTP 请求正文。其类型可以是 JSON 对象或 String。 RequestBody 仅支持PATCH、POST 和 PUT HTTP 方法。
- AllowNullValues
  - 类型：BOOLEAN-默认值：false
  - 在默认设置下，任何处于请求输入状态的空值都不会发送到您的 API。在以下示例中，除非在状态机定义中将其设置AllowNullValues为，否则该category字段不会包含true在请求中。

```
{
    "NewPet": {
        "type": "turtle",
        "price": 123,
```

```
        "category": null
    }
}
```

> **Note**
>
> 默认情况下，请求输入状态为空值的字段不会发送到您的 API。通过`true`在状态机定义中设置为，可以强制将空值发送`AllowNullValues`到您的 API。

- `AuthType`
  - 类型：`JSON`
  - 身份验证方法。默认方法是 `NO_AUTH`。容许值为：
    - `NO_AUTH`
    - `IAM_ROLE`
    - `RESOURCE_POLICY`

    有关更多信息，请参阅身份验证和授权。

> **Note**
>
> 出于安全考虑，目前不允许使用以下 HTTP 标头密钥：
>
> - 任何以 `X-Forwarded`、`X-Amz` 或 `X-Amzn` 为前缀的标头密钥。
> - `Authorization`
> - `Connection`
> - `Content-md5`
> - `Expect`
> - `Host`
> - `Max-Forwards`
> - `Proxy-Authenticate`
> - `Server`
> - `TE`
> - `Transfer-Encoding`
> - `Trailer`

- Upgrade
- Via
- Www-Authenticate

下面的代码示例展示了如何使用 Step Functions 调用 API Gateway。

```
{
    "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"
    }
}
```

身份验证和授权

您可以使用以下验证方法：

- 无授权：直接调用 API，无需授权方法。
- IAM 角色：使用此方法，Step Functions 将扮演状态机的角色，使用签名版本 4 (Sigv4) 对请求进行签名，然后调用 API。
- 资源策略：Step Functions 对请求进行身份验证，然后调用 API。您必须将资源策略附加到 API，其中指定以下内容：

  1. 将调用 API Gateway 的状态机。

> **Important**
>
> 必须指定状态机，用于能限制对它的访问。如果不这样做，那么任何使用资源策略身份验证对 API 进行 API Gateway 请求验证的状态机都将被允许访问。

2. 该 Step Functions 就是调用 API Gateway 的服务："Service": "states.amazonaws.com"。
3. 要访问的资源，包括：
   - *region*。
   - 指定区域内的 *account-id*。
   - *api-id*。
   - *stage-name*。
   - *HTTP-VERB*（方法）。
   - *resource-path-specifier*。

有关资源策略的示例，请参阅 Step Functions 和 API Gateway 的 IAM 策略。

有关资源格式的更多信息，请参阅 API Gateway 开发人员指南中的在 API Gateway 中执行 API 的权限的资源格式。

> **Note**
>
> 只有 REST API 支持资源策略。

服务集成模式

API Gateway 集成支持两种服务集成模式：

- 请求响应，这是默认的集成模式。它允许 Step Functions 在收到 HTTP 响应后立即进入下一步。
- 等待具有任务令牌的回调 (.waitForTaskToken)，该模式会等到任务令牌与有效载荷一起返回。要使用该.waitForTaskToken模式，请在任务定义的 “资源” 字段末尾添加.wai ForTaskToken t，如以下示例所示：

```
{
    "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"
    }
}
```

输出格式

提供以下输出参数：

名称	Type	描述
ResponseBody	JSON 或 String	API 调用的响应正文。
Headers	JSON	响应标头。
StatusCode	Integer	响应的 HTTP 状态代码。
StatusText	String	响应的状态文本。

响应示例：

```
{
    "ResponseBody": {
        "myBills": []
    },
    "Headers": {
        "key": ["value1", "value2"]
    },
```

```
    "StatusCode": 200,
    "StatusText": "OK"
}
```

错误处理

发生错误时，会返回 error 和 cause 如下：

- 如果 HTTP 状态码可用，则错误将以 ApiGateway.*<HTTP Status Code>* 格式返回。
- 如果 HTTP 状态码不可用，则错误将以 ApiGateway.*<Exception>* 格式返回。

在这两种情况下，cause 都以字符串形式返回。

以下示例展示了发生错误时的响应：

```
{
    "error": "ApiGateway.403",
    "cause": "{\"message\":\"Missing Authentication Token\"}"
}
```

> **Note**
>
> 状态码为 2XX 表示成功，不会返回任何错误。所有其他状态代码或抛出的异常都将导致错误。

有关更多信息，请参阅：

- API Gateway 开发人员指南中的 Amazon API Gateway 概念。
- 亚马逊 API Gateway 的 IAM 政策
- 演示如何调用 API Gateway 的示例项目

API Gateway 开发人员指南中的 Amazon API Gateway 概念。

使用 Step Functions 调用 Athena

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

优化后的 Athena 集成与 Athena SDK 集成有何不同 AWS

- 支持运行作业 (.sync) 集成模式。
- 没有针对请求响应集成模式的优化。
- 不支持等待具有任务令牌的回调集成模式。

通过与 Amazon Athena 的 AWS Step Functions 服务集成，您可以使用 Step Functions 启动和停止查询执行并获取查询结果。使用 Step Functions，您可以运行临时或计划数据查询，并检索针对 S3 数据湖的结果。Athena 没有服务器，因此您无需设置或管理任何基础设施，且只需为您运行的查询付费。

要 AWS Step Functions 与亚马逊 Athena 集成，您可以使用提供的雅典娜服务集成 API。

服务集成 API 与相应的 Athena API 相同。并非所有 API 都支持所有集成模式，如下表所示。

API	请求响应	运行作业 (.sync)
`StartQueryExecution`	✓	✓
`StopQueryExecution`	✓	
`GetQueryExecution`	✓	
`GetQueryResults`	✓	

支持的 Amazon Athena API：

Note

在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

- `StartQueryExecution`
  - 请求语法
  - 支持的参数：

        - ClientRequestToken
        - ExecutionParameters
        - QueryExecutionContext
        - QueryString
        - ResultConfiguration
        - WorkGroup
    - Response syntax
- StopQueryExecution
    - 请求语法
    - 支持的参数：
        - QueryExecutionId
- GetQueryExecution
    - 请求语法
    - 支持的参数：
        - QueryExecutionId
    - Response syntax
- GetQueryResults
    - 请求语法
    - 支持的参数：
        - MaxResults
        - NextToken
        - QueryExecutionId
    - Response syntax

下面包含一个启动 Athena 查询作业的 Task 状态。

```
"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"
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

AWS Batch 使用 Step Functions 进行管理

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化后的 AWS Batch 集成与 S AWS BatchAWS DK 集成有何不同
>
> - 运行作业 (.sync) 集成模式可用。
>
> 请注意，没有针对 请求响应 或 等待具有任务令牌的回调 集成模式的优化。

支持 AWS Batch 的 API：

- SubmitJob
  - 请求语法
  - 支持的参数：
    - ArrayProperties
    - ContainerOverrides
    - DependsOn
    - JobDefinition
    - JobName
    - JobQueue
    - Parameters
    - RetryStrategy
    - Timeout

  - Tags
- 响应语法

> **中的Step Functions参数表示为 PascalCase**
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

以下内容包括提交 AWS Batch 作业并等待其完成的`Task`状态。

```
{
  "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
    }
  }
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM策略。

使用 Step Functions 调用 Amazon Bedrock

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

主题

- Amazon Bedrock 服务集成 API
- Amazon Bedrock 集成的 Task 状态定义

Amazon Bedrock 服务集成 API

要将 AWS Step Functions 与 Amazon Bedrock 集成，您可以使用以下 API。这些 API 与相应的 Amazon Bedrock API 相似，但在传递的请求字段方面有所不同。

下表描述了每个服务集成 API 及其相应 Amazon Bedrock API 之间的差异。

Amazon Bedrock 服务集成 API 和相应的 Amazon Bedrock API

Amazon Bedrock 服务集成 API	相应的 Amazon Bedrock API	差异
InvokeModel 使用您在请求正文中提供的输入，调用指定的 Amazon Bedrock 模型来运行推理。您可以使用 `InvokeModel` 对文本模型、图像模型和嵌入模型运行推理。	InvokeModel	Amazon Bedrock 服务集成 API 请求正文包含以下额外参数。 • Body：以 Content-Type 请求标头中指定的格式指定输入数据。Body 包含特定于目标模型的参数。 如果您使用 `InvokeModel` API，则必须指定 Body 参数。Step Functions 不会验证您在 Body 中提供的输入。 当使用 Amazon Bedrock 优化的集成指定 Body 时，您可以指定最大 256 KB 的有

Amazon Bedrock 服务集成 API	相应的 Amazon Bedrock API	差异
		效负载。如果有效负载超过 256 KB，我们建议您使用 `Input`。 • `Input`：指定要从中检索输入数据的源。此可选字段专门用于 Amazon Bedrock 与 Step Functions 的优化集成。在此字段中，您可以指定 `S3Uri`。 您可以在参数中指定 `Body` 或指定 `Input`，但不能同时指定两者。 如果指定 `Input` 但未指定 `ContentType` ，输入数据来源的内容类型将变为 `ContentType` 的值。 • `Output`：指定写入 API 响应的目的地。此可选字段专门用于 Amazon Bedrock 与 Step Functions 的优化集成。在此字段中，您可以指定 `S3Uri`。 如果您指定此字段，API 响应正文将替换为对原始输出的 Amazon S3 位置的引用。 以下示例显示了用于Amazon Bedrock集成 InvokeModel API 的语法。

Amazon Bedrock 服务集成 API	相应的 Amazon Bedrock API	差异

		<pre>{
		    "ModelId": String,
		  // required
		    "Accept": String,
		  // default: applicati
		on/json
		    "ContentType":
		 String, // default:
		 application/json
		    "Input": { // not
		 from Bedrock API
		        "S3Uri": String
		    },
		    "Output": { // not
		 from Bedrock API
		        "S3Uri": String
		    }
		}</pre>

CreateModelCustomizationJob 创建微调作业以自定义基础模型。	CreateModelCustomizationJob	无
CreateModelCustomizationJob.sync 创建微调作业以自定义基础模型。	CreateModelCustomizationJob	无

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

Amazon Bedrock 集成的 Task 状态定义

以下 Task 状态定义显示了如何在状态机中与 Amazon Bedrock 集成。此示例显示了一个 Task 状态，它提取了路径指定的模型调用的完整结果，`result_one`。这基于基础模型的推理参数。此示例使用 Cohere Command 大型语言模型 (LLM)。

```
{
  "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**
>
> 要部署与Amazon Bedrock集成的状态机的示例 AWS 账户，请参阅使用 Amazon Bedrock 执行 AI 提示链接。

AWS CodeBuild 使用 Step Functions 调用

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> **优化后的 CodeBuild 集成与 S CodeBuild AWS DK 集成有何不同**
>
> - 支持运行作业 (.sync) 集成模式。

- 调用StopBuild或后StopBuildBatch，只有在内部完成一些内部工作 CodeBuild 以最终确定一个或多个生成状态后，才能立即删除生成或生成批次。如果您在此期间尝试使用 BatchDeleteBuilds 或 DeleteBuildBatch，则可能无法删除构建或构建批处理。BatchDeleteBuilds 和 DeleteBuildBatch 的优化服务集成包括内部重试，以简化停止后立即删除的用例。

与的 AWS Step Functions 服务集成 AWS CodeBuild 使您能够使用 Step Functions 触发、停止和管理构建，以及共享生成报告。使用 Step Functions，您可以设计和运行持续的集成管道，以验证应用程序的软件更改。

并非所有 API 都支持所有集成模式，如下表所示。

API	请求响应	运行作业 (.sync)
StartBuild	✓	✓
StopBuild	✓	
BatchDeleteBuilds	✓	
BatchGetReports	✓	
StartBuildBatch	✓	✓
StopBuildBatch	✓	
RetryBuildBatch	✓	✓
DeleteBuildBatch	✓	

中的Step Functions参数表示为 PascalCase

即使原生服务 API 在 camelCase 中（例如 API 操作）startSyncExecution，您也可以在中指定参数 PascalCase，例如:。StateMachineArn

支持 CodeBuild 的 API 和语法：

- StartBuild
  - 请求语法
  - 支持的参数：
    - 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
  - 响应语法
- StopBuild
  - 请求语法
  - 支持的参数：
    - Id
  - 响应语法
- BatchDeleteBuilds
  - 请求语法
  - 支持的参数：
    - Ids
  - 响应语法
- BatchGetReports
  - 请求语法
  - 支持的参数：
    - ReportArns
  - 响应语法
- StartBuildBatch
  - 请求语法
  - 支持的参数：
    - 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

- 响应语法

- StopBuildBatch
  - 请求语法
  - 支持的参数：
    - Id
  - 响应语法
- RetryBuildBatch
  - 请求语法

- 支持的参数：
  - Id
  - IdempotencyToken
  - RetryType
- 响应语法

- DeleteBuildBatch
  - 请求语法
  - 支持的参数：
    - Id
  - 响应语法

> **Note**
>
> 您可以对 BatchDeleteBuilds 使用 JSONPath 递归下降 (..) 运算符。这将返回一个数组，并使您可以将 Arn 字段从 StartBuild 转换为复数 Ids 参数，如以下示例所示。

```
"BatchDeleteBuilds": {
    "Type": "Task",
    "Resource": "arn:aws:states:::codebuild:batchDeleteBuilds",
    "Parameters": {
        "Ids.$": "$.Build..Arn"
    },
    "Next": "MyNextState"
},
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 调用 DynamoDB API

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> **Note**
>
> 在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

> **优化后的 DynamoDB 集成与 DynamoDB 软件开发工具包集成有何不同 AWS**
>
> - 请求响应集成模式没有优化。
> - 不支持等待具有任务令牌的回调集成模式。
> - 通过优化集成，只有 `GetItem`、`PutItem`、`UpdateItem` and `DeleteItem` API 操作可用。其他 API 操作（例如 `CreateTable`）可通过 DynamoDB AWS 软件开发工具包集成获得。

支持的 Amazon DynamoDB API 和语法：

- `GetItem`
  - 请求语法
  - 支持的参数：
    - `Key`
    - `TableName`
    - `AttributesToGet`
    - `ConsistentRead`
    - `ExpressionAttributeNames`
    - `ProjectionExpression`
    - `ReturnConsumedCapacity`
  - 响应语法
- `PutItem`
  - 请求语法
  - 支持的参数：
    - `Item`

    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - 响应语法
- DeleteItem
  - 请求语法
  - 支持的参数：
    - Key
    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - 响应语法
- UpdateItem
  - 请求语法
  - 支持的参数：
    - Key
    - TableName
    - AttributeUpdates

- ConditionalOperator
- ConditionExpression
- Expected
- ExpressionAttributeNames
- ExpressionAttributeValues
- ReturnConsumedCapacity
- ReturnItemCollectionMetrics
- ReturnValues
- UpdateExpression

- 响应语法

> **中的Step Functions参数表示为 PascalCase**
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

下面是从 DynamoDB 检索消息的 `Task` 状态。

```
"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"
},
```

要在可正常使用的示例中查看此状态，请参阅传输数据记录（Lambda、DynamoDB、Amazon SQS）示例项目。

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 管理 Amazon ECS 或 Fargate 任务

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 经过优化的亚马逊 ECS/Fargate 集成与亚马逊 ECS 或 Fargate SDK 集成有何不同 AWS
>
> - 支持运行作业 (.sync) 集成模式。
> - `ecs:runTask` 可以返回 HTTP 200 响应，但有一个非空的 `Failures` 字段，如下所示：
>   - 请求响应：返回响应，任务不会失败。这与不进行优化相同。
>   - 运行作业或任务令牌：如果遇到非空 `Failures` 字段，则任务会因 `AmazonECS.Unknown` 错误而失败。

支持的 Amazon ECS/Fargate API 和语法：

> 中的Step Functions参数表示为 PascalCase
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

- `RunTask` 使用指定的任务定义启动新任务。
  - 请求语法
  - 支持的参数：
    - `Cluster`
    - `Group`
    - `LaunchType`
    - `NetworkConfiguration`
    - `Overrides`
    - `PlacementConstraints`
    - `PlacementStrategy`

- PlatformVersion
- PropagateTags
- TaskDefinition
- EnableExecuteCommand

- 响应语法

向 Amazon ECS 任务传递数据

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

您可以使用 overrides 覆盖容器的默认命令，并将输入传递给 Amazon ECS 任务。请参阅 ContainerOverride。在示例中，我们习惯 JsonPath 将值Task从输入传递到Task状态。

下面是一个运行 Amazon ECS 任务并等待任务完成的 Task 状态。

```
{
 "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 中的 "Command.$": "$.commands" 行将命令从状态输入传递到容器。

对于上一个示例，如果执行的输入为以下内容，则每个命令都将作为容器覆盖传递。

```
{
  "commands": [
    "test command 1",
    "test command 2",
    "test command 3"
  ]
}
```

下面包含一个运行 Amazon ECS 任务，然后等待返回任务令牌的 Task 状态。请参阅 等待具有任务令牌的回调。

```
{
   "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
      }
   }
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 调用 Amazon EKS

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 经过优化的亚马逊 EKS 集成与亚马逊 E AWS KS SDK 集成有何不同
>
> - 支持运行作业 (.sync) 集成模式。
> - 没有针对请求响应集成模式的优化。
> - 不支持等待具有任务令牌的回调集成模式。

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

Step Functions 提供了两种服务集成 API，用于与 Amazon Elastic Kubernetes Service 集成。其中一种允许您使用 Amazon EKS API 来创建和管理 Amazon EKS 集群。另一种允许您使用 Kubernetes API 与集群进行交互，并作为应用程序工作流的一部分运行作业。您可以使用 Kubernetes API 与使用 Step Functions 创建的 Amazon EKS 集群、使用 eksctl 工具或 Amazon EKS 控制台创建的 Amazon EKS 集群或类似方法进行集成。有关更多信息，请参阅 Amazon EKS 用户指南中的创建 Amazon EKS 集群。

> Note
>
> Step Functions EKS 集成仅支持具有公共端点访问权限的 Kubernetes API。默认情况下，EKS 集群 API 服务器端点具有公共访问权限。有关更多信息，请参阅《Amazon EKS 用户指南》中的 Amazon EKS 集群端点访问控制。

如果执行停止，Step Functions 不会自动终止 Amazon EKS 集群。如果您的状态机在 Amazon EKS 集群终止之前停止，则您的集群可能会无限期地继续运行，并且可能会产生额外费用。为避免这种情况，请确保您创建的任何 Amazon EKS 集群都已正确终止。有关更多信息，请参阅：

- Amazon EKS 用户指南中删除集群。

- 服务集成模式中的运行作业 (.sync)。

> **Note**
>
> 在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

Kubernetes API 集成

Step Functions 支持以下 Kubernetes API：

RunJob

`eks:runJob` 服务集成允许您在 Amazon EKS 集群上运行作业。您可以是使用 `eks:runJob.sync` 变体等待任务完成，也可以选择检索日志。

您的 Kubernetes API 服务器必须向状态机使用的 IAM 角色授予权限。有关更多信息，请参阅 权限。

对于运行任务 (`.sync`) 模式，作业的状态由轮询确定。Step Functions 最初的轮询速度约为每分钟 1 次。这个速度最终会减慢到大约每 5 分钟进行一次。如果您需要更频繁地进行轮询，或者需要对轮询策略进行更多控制，则可以使用 `eks:call` 集成来查询作业的状态。

`eks:runJob` 集成特定于 `batch/v1` Kubernetes 作业。有关更多信息，请参阅 Kubernetes 文档中的作业。如果您想管理其他 Kubernetes 资源，包括自定义资源，请使用 `eks:call` 服务集成。您可以使用 Step Functions 来构建轮询循环，如the section called “任务状态投票 (Lambda,) AWS Batch”示例项目所示。

支持的参数包括：

- `ClusterName`：要调用的 Amazon EKS 集群的名称。
  - `Type: String`
  - 必需：是
- `CertificateAuthority`：检索与集群通信所需的 Base64 编码证书数据。您可以从亚马逊 EKS 控制台或使用亚马逊 EK S DescribeClusterAPI 获取此值。
  - `Type: String`
  - 必需：是

- Endpoint：Kubernetes API 服务器的端点 URL。您可以从亚马逊 EKS 控制台或使用亚马逊 EK S DescribeClusterAPI 获取此值。
  - Type: String
  - 必需：是
- Namespace：运行作业的命名空间。如果未提供，则使用 default 命名空间。
  - Type: String
  - 必需：否
- Job: Kubernetes 作业的定义。请参阅 Kubernetes 文档中的作业。
  - Type：JSON 或 String
  - 必需：是
- LogOptions：一组用于控制日志可选检索的选项。仅当使用运行作业 (.sync) 服务集成模式来等待作业完成时才适用。
  - Type: JSON
  - 必需：否
  - 日志包含在密钥 logs 下的响应中。任务中可能有多个容器组，每个容器组都有多个容器。

```
{
  ...
  "logs": {
    "pods": {
      "pod1": {
        "containers": {
          "container1": {
            "log": <log>
          },
          ...
        }
      },
      ...
    }
  }
```

  - 将尽最大努力执行日志检索。如果检索日志时出错，则将使用 error 和 cause 字段代替 log 字段。
- LogOptions.RetrieveLogs：在任务完成后启用日志检索。默认情况下，不会检索日志。
  - Type: Boolean

  - 必需：否
- `LogOptions.RawLogs`: 如果 `RawLogs` 设置为 true，则日志将作为原始字符串返回，而无需尝试将其解析为 JSON。默认情况下，如果可能，日志会反序列化为 JSON。在某些情况下，此类解析可能会带来不必要的更改，例如限制包含多位数的数字精确度。
  - Type: `Boolean`
  - 必需：否
- `LogOptions.LogParameters`：Kubernetes API 的读取日志 API 支持查询参数来控制日志检索。例如，您可以使用 `tailLines` 或 `limitBytes` 来限制检索到的日志的大小，并保持在 Step Functions 的数据大小配额之内。有关更多信息，请参阅 Kubernetes API 参考中的读取日志部分。
  - `Type: String` 到 `List of Strings` 的映射
  - 必需：否
  - 例如：

```
"LogParameters": {
  "tailLines": [ "6" ]
}
```

以下示例包括运行作业、等待任务完成然后检索作业日志的 `Task` 状态：

```
{
  "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 服务集成允许您使用 Kubernetes API 通过 Kubernetes API 端点读取和写入 Kubernetes 资源对象。

您的 Kubernetes API 服务器必须向状态机使用的 IAM 角色授予权限。有关更多信息，请参阅 权限。

有关使用可用操作的更多信息，请参阅 Kubernetes API 参考。

Call 支持的参数包括：

- ClusterName：要调用的 Amazon EKS 集群的名称。
  - Type：字符串

  - 必需：是
- `CertificateAuthority`：检索与集群通信所需的 Base64 编码证书数据。您可以从亚马逊 EKS 控制台或使用亚马逊 EK S DescribeClusterAPI 获取此值。
  - Type: `String`
  - 必需：是
- `Endpoint`：Kubernetes API 服务器的端点 URL。您可以在亚马逊 E KS 控制台上或使用亚马逊 EKS 的 DescribeCluster API 找到此值。
  - Type: `String`
  - 必需：是
- `Method`：请求的 HTTP 方法。以下值之一：`GET`、`POST`、`PUT`、`DELETE`、`HEAD` 或 `PATCH`。
  - Type: `String`
  - 必需：是
- `Path`: Kubernetes REST API 操作的 HTTP 路径。
  - Type: `String`
  - 必需：是
- `QueryParameters`: Kubernetes REST API 操作的 HTTP 查询参数。
  - `Type: String 到 List of Strings 的映射`
  - 必需：否
  - 例如：

```
"QueryParameters": {
  "labelSelector": [ "job-name=example-job" ]
}
```

- `RequestBody`: Kubernetes REST API 操作的 HTTP 消息正文。
  - `Type：JSON 或 String`
  - 必需：否

下面包含一个 `Task` 状态，它使用 `eks:call` 列出属于作业 `example-job` 的容器组。

```
{
  "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
    }
  }
}
```

下面包含一个 Task 状态，它使用 eks:call 删除作业 example-job，并设置 propagationPolicy 以确保作业的容器组也被删除。

```
{
  "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
    }
  }
```

```
}
```

支持的 Amazon EKS API

支持的 Amazon EKS API 和语法包括：

- `CreateCluster`
  - 请求语法
  - 响应语法

  使用 `eks:createCluster` 服务集成创建 Amazon EKS 集群时，IAM 角色会作为管理员（具有 system:masters 权限）添加到 Kubernetes RBAC 授权表中。最初，仅该 IAM 实体可以调用 Kubernetes API 服务器。有关更多信息，请参阅：

  - 《Amazon EKS 用户指南》中的管理集群的用户或 IAM 角色
  - 权限部分

  Amazon EKS 使用服务相关角色，其中包含 Amazon EKS 代表您调用其他服务所需的权限。如果您的账户中还不存在这些服务相关角色，您必须将 `iam:CreateServiceLinkedRole` 权限添加到 Step Functions 使用的 IAM 角色中。有关更多信息，请参阅《Amazon EKS 用户指南》中的使用服务相关角色。

  Step Functions 使用的 IAM 角色必须具有 `iam:PassRole` 旋前，才能将集群 IAM 角色传递给 Amazon EKS。有关更多信息，请参阅《Amazon EKS 用户指南》中的 Amazon EKS 集群 IAM 角色。
- `DeleteCluster`
  - 请求语法
  - 响应语法

  在删除集群之前，必须删除任何 Fargate 配置文件或节点组。
- `CreateFargateProfile`
  - 请求语法
  - 响应语法

  Amazon EKS 使用服务相关角色，其中包含 Amazon EKS 代表您调用其他服务所需的权限。如果您的账户中还不存在这些服务相关角色，您必须将 `iam:CreateServiceLinkedRole` 权限添加到 Step Functions 使用的 IAM 角色中。有关更多信息，请参阅《Amazon EKS 用户指南》中的使用服务相关角色。

Fargate 上的 Amazon EKS 可能并未在所有区域提供。有关区域可用性的信息，请参阅《Amazon EKS 用户指南》中的 Fargate 部分。

Step Functions 使用的 IAM 角色必须具有 `iam:PassRole` 旋前，才能将容器组执行 IAM 角色传递给 Amazon EKS。有关更多信息，请参阅《Amazon EKS 用户指南》中的容器组执行角色。

- DeleteFargateProfile
  - 请求语法
  - 响应语法
- CreateNodegroup
  - 请求语法
  - 响应语法

  Amazon EKS 使用服务相关角色，其中包含 Amazon EKS 代表您调用其他服务所需的权限。如果您的账户中还不存在这些服务相关角色，您必须将 `iam:CreateServiceLinkedRole` 权限添加到 Step Functions 使用的 IAM 角色中。有关更多信息，请参阅《Amazon EKS 用户指南》中的使用服务相关角色。

  Step Functions 使用的 IAM 角色必须具有 `iam:PassRole` 旋前，才能将节点 IAM 角色传递给 Amazon EKS。有关更多信息，请参阅《Amazon EKS 用户指南》中的使用服务相关角色。
- DeleteNodegroup
  - 请求语法
  - 响应语法

以下内容包含一个创建 Amazon EKS 集群的 `Task` 状态。

```
{
  "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
    }
  }
}
```

以下内容包括一个删除 Amazon EKS 集群的 Task 状态。

```
{
  "StartAt": "DeleteCluster.sync",
  "States": {
    "DeleteCluster.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteCluster.sync",
      "Parameters": {
        "Name": "MyCluster"
      },
      "End": true
    }
  }
}
```

以下内容包含一个创建 Fargate 配置文件的 Task 状态。

```
{
  "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
    }
  }
}
```

以下内容包含一个删除 Fargate 配置文件的 Task 状态。

```
{
  "StartAt": "DeleteFargateProfile.sync",
  "States": {
    "DeleteFargateProfile.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteFargateProfile.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "FargateProfileName": "MyFargateProfile"
      },
      "End": true
    }
  }
}
```

以下内容包含一个创建节点组的 Task 状态。

```
{
  "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
    }
  }
}
```

以下内容包含一个删除节点组的 Task 状态。

```
{
  "StartAt": "DeleteNodegroup.sync",
  "States": {
    "DeleteNodegroup.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteNodegroup.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "NodegroupName": "MyNodegroup"
      },
      "End": true
    }
  }
}
```

权限

使用 `eks:createCluster` 服务集成创建 Amazon EKS 集群时，IAM 角色会作为管理员添加到 Kubernetes RBAC 授权表中，并拥有 `system:masters` 权限。最初，仅该 IAM 实体可以调用 Kubernetes API 服务器。例如，您将无法使用 kubectl 与您的 Kubernetes API 服务器交互，除非您承担与 Step Functions 状态机相同的角色，或者您配置 Kubernetes 向其他 IAM 实体授予权限。有关更多信息，请参阅《Amazon EKS 用户指南》中的管理集群的用户或 IAM 角色。

您可以将其他 IAM 实体（如用户或角色）添加到 kube-system 名称空间中的 `aws-auth ConfigMap`，从而为它们添加权限。如果您通过 Step Functions 创建集群，请使用 `eks:call` 服务集成。

下面包含一个 `Task` 状态，该状态创建了 `aws-auth ConfigMap` ，并向用户 `arn:aws:iam::123456789012:user/my-user` 和 IAM 角色 `arn:aws:iam::123456789012:role/my-role` 授予 `system:masters` 权限。

```
{
  "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**
>
> 您可能会看到 IAM 角色的 ARN 以包含 /service-role/ 路径的格式显示，例如 `arn:aws:iam::123456789012:role/service-role/my-role`。在 `aws-auth` 中列出角色时，不应包含此服务角色路径令牌。

首次创建集群时，`aws-auth ConfigMap` 并不存在，但如果创建了 Fargate 配置文件，则会自动添加。您可以检索 `aws-auth` 的当前值、添加其他权限和 PUT 一个新版本。通常，在 Fargate 配置文件之前创建 `aws-auth` 会更容易。

如果您的集群是在 Step Functions 之外创建的，则可以配置 kubectl 与 Kubernetes API 服务器通信。然后，使用 `kubectl apply -f aws-auth.yaml` 创建新的 `aws-auth ConfigMap` 或使用 `kubectl edit -n kube-system configmap/aws-auth` 编辑已存在的 aws-auth ConfigMap。有关更多信息，请参阅：

- 《Amazon EKS 用户指南》中的为 Amazon EKS 创建 kubeconfig。
- 《Amazon EKS 用户指南》中的管理集群的用户或 IAM 角色。

如果您的 IAM 角色在 Kubernetes 中没有足够的权限，`eks:call` 或 `eks:runJob` 服务集成将失败，并出现以下错误：

```
Error:
EKS.401

Cause:
{
  "ResponseBody": {
    "kind": "Status",
    "apiVersion": "v1",
    "metadata": {},
    "status": "Failure",
    "message": "Unauthorized",
    "reason": "Unauthorized",
    "code": 401
  },
  "StatusCode": 401,
  "StatusText": "Unauthorized"
}
```

使用 Step Functions 调用 Amazon EMR

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> **优化的亚马逊 EMR 集成与亚马逊 EMR SDK 集成有何不同 AWS**
>
> 优化的 Amazon EMR 服务集成有一组自定义的 API，用于封装底层的 Amazon EMR API，如下所述。因此，它与 Amazon EMR AWS SDK 服务集成有很大不同。此外，它还支持运行作业 (.sync) 集成模式。

要 AWS Step Functions 与亚马逊 EMR 集成，您可以使用提供的亚马逊 EMR 服务集成 API。服务集成 API 与相应的 Amazon EMR API 相似，但在传递的字段和返回的响应方面有所不同。

如果执行停止，Step Functions 不会自动终止 Amazon EMR 集群。如果您的状态机在 Amazon EMR 集群终止之前停止，则您的集群可能会无限期地继续运行，并且可能会产生额外费用。为避免这种情况，请确保您创建的任何 Amazon EMR 集群都已正确终止。有关更多信息，请参阅：

- 在 Amazon EMR 用户指南中控制集群终止。
- 服务集成模式运行作业 (.sync) 部分。

> **Note**
>
> 自 `emr-5.28.0` 起，您可以在创建集群时指定参数 `StepConcurrencyLevel`，以允许在单个集群上并行运行多个步骤。您可以使用 Step Functions `Map` 和 `Parallel` 状态将工作并行提交到集群。

Amazon EMR 服务集成的可用性取决于 Amazon EMR API 的可用性。请查看 Amazon EMR 文档，了解特殊区域的限制。

> **Note**
>
> 为了与 Amazon EMR 集成，Step Functions 在前 10 分钟具有硬编码的 60 秒作业轮询频率，10 分钟后为 300 秒作业轮询频率。

下表描述了每个服务集成 API 及其相应 Amazon EMR API 之间的差异。

Amazon EMR 服务集成 API 和相应的 Amazon EMR API

Amazon EMR 服务集成 API	相应的 EMR API	差异
CreateCluster 创建并开始运行集群（作业流程）。 Amazon EMR 与一种独特类型的 IAM 角色（称为服务相关角色）直接关联。要使 `createCluster` 和 `createCluster.sync` 起作用，您必须配置必要的权限以创建与服务关联的角色 `AWSServiceRoleForE`	runJobFlow	`createCluster` 使用与相同的请求语法 runJobFlow，但以下语法除外： • 必须填写 `Instances.KeepJobFlowAliveWhenNoSteps` 字段，且该字段必须具有 Boolean 值 `TRUE`。 • 不允许填写字段 `Steps`。 • 应提供 `Instances.InstanceFleets[in`

Amazon EMR 服务集成 API	相应的 EMR API	差异
MRCleanup 。有关此问题的更多信息，包括您可以添加到 IAM 权限策略的语句，请参阅使用 Amazon EMR 的服务关联角色。		dex].Name 字段的值，并且如果使用可选的 modifyInstanceFleetByName 连接器 API，该字段必须是唯一的。 • 应提供 Instances.InstanceGroups[index].Name 字段的值，并且如果使用可选的 modifyInstanceGroupByName API，该字段必须是唯一的。 响应如下： { "ClusterId": "string" } Amazon EMR 使用以下信息： { "JobFlowId": "string" }
createCluster.sync 创建并开始运行集群（作业流程）。	runJobFlow	与 createCluster 相同，但等待集群达到 WAITING 状态。

Amazon EMR 服务集成 API	相应的 EMR API	差异

setClusterTermination保护
 	
锁定集群（作业流程），以使集群中的 EC2 实例无法通过用户干预、API 调用或作业流程错误终止。
	setTerminationProtection	请求使用：
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR 使用以下信息：
		
		`{ "JobFlowIds": ["string"] }`

terminateCluster
 	
关闭集群（作业流程）。
	terminateJobFlows	请求使用：
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR 使用以下信息：
		
		`{ "JobFlowIds": ["string"] }`

terminateCluster.sync 关闭集群（作业流程）。	terminateJobFlows	与 `terminateCluster` 相同，但等待集群终止。

Amazon EMR 服务集成 API	相应的 EMR API	差异

addStep
 	
向正在运行的集群添加新步骤。
 	
另外，使用此 API 时，还能指定 `ExecutionRoleArn` 参数。
	addJobFlow步骤	请求使用密钥 `"ClusterId"` 。Amazon EMR 使用 `"JobFlowId"` 。请求使用单一步骤。
		
		`{ "Step": <"StepConfig object"> }`
		
		Amazon EMR 使用以下信息：
		
		`{ "Steps": [<StepConfig objects>] }`
		
		响应如下：
		
		`{ "StepId": "string" }`
		
		Amazon EMR 返回以下内容：
		
		`{ "StepIds": [<strings>] }`

Amazon EMR 服务集成 API	相应的 EMR API	差异
addStep.sync 向正在运行的集群添加新步骤。 另外，使用此 API 时，还能指定 `ExecutionRoleArn` 参数。	addJobFlow步骤	与 addStep 相同，但等待步骤完成。

cancelStep
 	
取消正在运行的集群中的一个待处理步骤。
	cancelSteps	请求使用：
		
		`{ "StepId": "string" }`
		
		Amazon EMR 使用以下信息：
		
		`{ "StepIds": [<strings>] }`
		
		响应如下：
		
		`{ "CancelStepsInfo": <CancelStepsInfo object> }`
		
		Amazon EMR 使用以下信息：
		
		`{ "CancelStepsInfoList": [<CancelStepsInfo objects>] }`

Amazon EMR 服务集成 API	相应的 EMR API	差异
modifyInstanceFleetByName 使用指定的 `InstanceFleetName` 修改实例队列的目标按需容量和目标 Spot 容量。	modifyInstanceFleet	请求与 `modifyInstanceFleet` 相同，但以下情况除外： • 不允许填写字段 `InstanceFleetId` 。 • 在运行时，通过调用 `ListInstanceFleets` 并解析结果，服务集成会自动确定 `InstanceFleetId` 。

Amazon EMR 服务集成 API	相应的 EMR API	差异

modifyInstanceGroupByName
 	
修改实例组的节点数和配置设置。
	modifyInstanceGroups	请求如下：
		
		`{ "ClusterId": "string", "InstanceGroup": <InstanceGroupModifyConfig object> }`
		
		Amazon EMR 使用以下列表：
		
		`{ "ClusterId": ["string"], "InstanceGroups": [<InstanceGroupModifyConfig objects>] }`
		
		在 InstanceGroupModifyConfig 对象中，不允许填写 InstanceGroupId 字段。
		
		已添加一个新字段 InstanceGroupName 。在运行时，通过调用 ListInstanceGroups 并解析结果，服务集成会自动确定 InstanceGroupId 。

以下内容包含一个创建集群的 Task 状态。

```
"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
}
```

以下内容包括启用终止保护的 Task 状态。

```
"Enable_Termination_Protection": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:setClusterTerminationProtection",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "TerminationProtected": true
    },
    "End": true
}
```

以下内容包括向集群提交步骤的 Task 状态。

```
"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
}
```

以下内容包括取消步骤的 Task 状态。

```
"Cancel_Step_One": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:cancelStep",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "StepId.$": "$.AddStepsResult.StepId"
    },
    "End": true
}
```

以下内容包括终止集群的 Task 状态。

```
"Terminate_Cluster": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:terminateCluster.sync",
    "Parameters": {
        "ClusterId.$": "$.ClusterId"
    },
    "End": true
}
```

以下内容包括为实例组向上或向下扩展集群的 Task 状态。

```
"ModifyInstanceGroupByName": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceGroupByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceGroupName": "MyCoreGroup",
        "InstanceGroup": {
            "InstanceCount": 8
        }
    },
    "End": true
}
```

以下内容包括为实例队列向上或向下扩展集群的 Task 状态。

```
"ModifyInstanceFleetByName": {
    "Type": "Task",
```

```
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceFleetByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceFleetName": "MyCoreFleet",
        "InstanceFleet": {
            "TargetOnDemandCapacity": 8,
            "TargetSpotCapacity": 0
        }
    },
    "End": true
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用以下方式在 EKS 上致电 Amazon EMR AWS Step Functions

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

EKS 上优化的亚马逊 EMR 集成与 EKS 上的 Amazon EMR SDK 集成有何不同 AWS

- 支持运行作业 (.sync) 集成模式。
- 没有针对请求响应集成模式的优化。
- 不支持等待具有任务令牌的回调集成模式。

Note

为了与 Amazon EMR 集成，Step Functions 在前 10 分钟具有硬编码的 60 秒作业轮询频率，10 分钟后为 300 秒作业轮询频率。

要在 EKS 上 AWS Step Functions 与 Amazon EMR 集成，请使用 EKS 上的 Amazon EMR 服务集成 API。服务集成 API 与相应的 Amazon EMR on EKS API 相同，但并非所有 API 都支持所有集成模式，如下表所示。

API	请求响应	运行作业 (.sync)
CreateVirtualCluster	✓	
DeleteVirtualCluster	✓	✓
StartJobRun	✓	✓

支持的 Amazon EMR on EKS API

> **Note**
>
> 在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

- CreateVirtualCluster
  - 请求语法
  - 支持的参数
  - 响应语法
- DeleteVirtualCluster
  - 请求语法
  - 支持的参数
  - 响应语法
- StartJobRun
  - 请求语法
  - 支持的参数
  - 响应语法

以下内容包含一个创建虚拟集群的 Task 状态。

```
"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
}
```

下面包括一个将作业提交到虚拟集群并等待完成的 Task 状态。

```
"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
}
```

下面包括一个删除虚拟集群并等待删除完成的 Task 状态。

```
"Delete_Virtual_Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-containers:deleteVirtualCluster.sync",
  "Parameters": {
    "Id.$": "$.VirtualClusterId"
  },
  "End": true
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 调用 Amazon EMR Serverless

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

优化的 EMR Serverless 集成与 EMR Serverless AWS 开发工具包集成有何不同

- 优化的 EMR Serverless 服务集成有一组自定义的 API，用于封装底层的 EMR Serverless API。由于这种自定义，优化的EMR Serverless集成与 EMR Serverless AWS SDK 服务集成有很大不同。此外，优化的 EMR Serverless 集成支持运行作业 (.sync) 集成模式。
- 不支持等待具有任务令牌的回调集成模式。

本主题内容

- EMR Serverless 服务集成 API
- EMR 无服务器集成用例

EMR Serverless 服务集成 API

要将 AWS Step Functions 与 EMR Serverless 集成，您可以以下六个服务集成 API。这些服务集成 API 与相应的 EMR Serverless API 相似，但在传递的字段和返回的响应方面有所不同。

下表描述了每个服务集成 API 及其相应 EMR Serverless API 之间的差异。

EMR Serverless 服务集成 API 和相应的 EMR Serverless API

EMR Serverless 服务集成 API	相应的 EMR Serverless API	差异
createApplication 创建应用程序。 EMR Serverless 与一种独特类型的 IAM 角色（称为服务相关角色）关联。要使 `createApplication` 和 `createApplication.sync` 起作用，您必须配置必要的权限以创建与服务关联的角色 `AWSServiceRoleForAmazonEMRServerless`。有关此问题的更多信息，包括您可以添加到 IAM 权限策略	CreateApplication	无

EMR Serverless 服务集成 API	相应的 EMR Serverless API	差异
的语句，请参阅使用 EMR Serverless 的服务关联角色。		
createApplication.sync 创建应用程序。	CreateApplication	EMR Serverless API 和 EMR Serverless 服务集成 API 的请求和响应之间没有区别。但是，createApplication.sync 会等待应用程序进入 CREATED 状态。
startApplication 启动指定的应用程序并初始化该应用程序的初始容量（如果已配置）。	StartApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }`
startApplication.sync 启动指定的应用程序并初始化初始容量（如果已配置）。	StartApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }` 此外，startApplication.sync 会等待应用程序进入 STARTED 状态。

EMR Serverless 服务集成 API	相应的 EMR Serverless API	差异
stopApplication 停止指定的应用程序并释放初始容量（如果已配置）。在停止应用程序之前，必须完成或取消所有已计划和正在运行的作业。	StopApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }`
stopApplication.sync 停止指定的应用程序并释放初始容量（如果已配置）。在停止应用程序之前，必须完成或取消所有已计划和正在运行的作业。	StopApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }` 此外，stopApplication.sync 会等待应用程序进入 STOPPED 状态。
deleteApplication 删除应用程序 应用程序必须处于 STOPPED 或 CREATED 状态才能被删除。	DeleteApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }`

EMR Serverless 服务集成 API	相应的 EMR Serverless API	差异
deleteApplication.sync 删除应用程序 应用程序必须处于 STOPPED 或 CREATED 状态才能被删除。	DeleteApplication	EMR Serverless API 响应不包含任何数据，但 EMR Serverless 服务集成 API 响应包含以下数据。 `{ "ApplicationId": "string" }` 此外，stopApplication.sync 会等待应用程序进入 TERMINATED 状态。
startJobRun 启动作业运行。	StartJobRun	无
startJobRun.sync 启动作业运行。	StartJobRun	EMR Serverless API 和 EMR Serverless 服务集成 API 的请求和响应之间没有区别。但是，startJobRun.sync 会等待应用程序进入状态。SUCCESS
cancelJobRun 取消作业运行。	CancelJobRun	无
cancelJobRun.sync 取消作业运行。	CancelJobRun	EMR Serverless API 和 EMR Serverless 服务集成 API 的请求和响应之间没有区别。但是，cancelJobRun.sync 会等待应用程序进入状态。CANCELLED

EMR 无服务器集成用例

对于优化的 EMR Serverless 服务集成，我们建议您创建单个应用程序，然后使用该应用程序运行多个作业。例如，在单个状态机中，您可以包含多个startJobRun请求，所有这些请求都使用同一个应用程序。以下 任务状态 状态示例显示了将 EMR Serverless API 与 Step Functions 集成的用例。有关 EMR Serverless 其他用例的信息，请参阅什么是 Amazon EMR Serverless。

> **Tip**
>
> 要将与集成EMR Serverless用于运行多个作业的状态机示例部署到您的中 AWS 账户，请参阅运行 EMR Serverless 作业。

- 创建应用程序
- 启动应用程序
- 停止应用程序
- 删除应用程序
- 启动应用程序中的作业
- 取消应用程序中的作业

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

在以下用例所示的示例中，请将`####` 替换为特定于资源的信息。例如，*`yourApplicationId`*替换为EMR Serverless应用程序的 ID，例如`00yv7iv71inak893`。

创建应用程序

以下 Task 状态示例使用 createApplication.sync 服务集成 API 创建了一个应用程序。

```
"Create_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:createApplication.sync",
    "Parameters": {
        "Name": "MyApplication",
        "ReleaseLabel": "emr-6.9.0",
        "Type": "SPARK"
    },
```

```
    "End": true
}
```

启动应用程序

以下 Task 状态示例使用 startApplication.sync 服务集成 API 启动应用程序。

```
"Start_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:startApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

停止应用程序

以下 Task 状态示例使用 stopApplication.sync 服务集成 API 停止应用程序。

```
"Stop_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:stopApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

删除应用程序

以下 Task 状态示例使用 deleteApplication.sync 服务集成 API 删除应用程序。

```
"Delete_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:deleteApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

启动应用程序中的作业

以下任务状态示例使用 startJobRun.sync 服务集成 API 在应用程序中启动作业。

```
"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
}
```

取消应用程序中的作业

以下任务状态示例使用 cancelJobRun.sync 服务集成 API 取消应用程序中的作业。

```
"Cancel_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:cancelJobRun.sync",
    "Parameters": {
        "ApplicationId.$": "$.ApplicationId",
        "JobRunId.$": "$.JobRunId"
    },
    "End": true
}
```

EventBridge 使用 Step Functions 调用

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

优化 EventBridge 集成与 EventBridge AWS SDK 集成有何不同

- 执行 ARN 和状态机 ARN 会自动附加到每个 `PutEventsRequestEntry` 的 `Resources` 字段。
- 如果来自 `PutEvents` 的响应包含非零的 `FailedEntryCount`，则 `Task` 状态会失败，并出现错误 `EventBridge.FailedEntry`。

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

Step Functions 提供了一个用于与亚马逊集成的服务集成 API EventBridge。这样，您就可以通过直接从 Step Functions 工作流发送自定义事件来构建事件驱动型应用程序。

要使用该 `PutEvents` API，你需要在账户中创建一个与你要发送的事件的特定模式相匹配的 EventBridge 规则。例如，您可以：

- 在您的账户中创建 Lambda 函数，用于接收和打印与规则匹配的事件。 EventBridge
- 在您的账户中使用默认事件总线创建一条 EventBridge 规则，该规则与特定事件模式相匹配并以 Lambda 函数为目标。

有关更多信息，请参阅：

- 在 EventBridge 用户指南 PutEvents中@@ 使用添加亚马逊 EventBridge 活动。
- 服务集成模式中的等待具有任务令牌的回调。

Note

在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

支持的 EventBridge API

支持 EventBridge 的 API 和语法包括：

- PutEvents
  - 请求语法
  - 支持的参数：
    - Entries
  - 响应语法

以下内容包含一个发送自定义事件的 Task：

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::events:putEvents",
  "Parameters": {
    "Entries": [
      {
        "Detail": {
          "Message": "MyMessage"
        },
        "DetailType": "MyDetailType",
        "EventBusName": "MyEventBus",
        "Source": "my.source"
      }
    ]
  },
  "End": true
}
```

错误处理

PutEvents API 接受一个条目数组作为输入，然后返回一个结果条目数组。只要 PutEvents 操作成功，即使有一个或多个条目失败，PutEvents 也会返回 HTTP 200 响应。PutEvents 会在 FailedEntryCount 字段中返回失败条目的数量。

Step Functions 会检查 FailedEntryCount 是否大于零。如果大于零，Step Functions 将以 EventBridge.FailedEntry 错误结束状态。这样，当出现失败条目时，就可以使用 Step Functions 在任务状态上的内置错误处理功能来捕捉或重试，而不需要使用额外的状态来分析响应中的 FailedEntryCount。

> **Note**
>
> 如果已实现幂等性并能安全地重试所有条目，则可以使用 Step Functions 的重试逻辑。Step Functions 不会在重试前删除 `PutEvents` 输入数组中的成功条目。相反，它会使用原始的条目数组进行重试。

使用 Step Functions 管理 AWS Glue 作业

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> **优化后的 AWS Glue 集成与 S AWS GlueAWS DK 集成有何不同**
>
> - 运行作业 (.sync) 集成模式可用。
> - `JobName` 字段从请求中提取出来并插入到响应中，而响应通常只包含 `JobRunID`。

支持 AWS Glue 的 API：

- `StartJobRun`

> **中的Step Functions参数表示为 PascalCase**
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

以下内容包括启动 AWS Glue 作业的`Task`状态。

```
"Glue StartJobRun": {
      "Type": "Task",
      "Resource": "arn:aws:states:::glue:startJobRun.sync",
      "Parameters": {
        "JobName": "GlueJob-JTrRO5l98qMG"
      },
      "Next": "ValidateOutput"
    },
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM策略。

使用 Step Functions 管理 AWS Glue DataBrew 作业

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

您可以使用该 DataBrew 集成将数据清理和数据标准化步骤添加到分析和机器学习工作流程中。

支持 DataBrew 的 API：

- StartJobRun

以下内容包括启动请求-响应 DataBrew任务的Task状态。

```
"DataBrew StartJobRun": {
            "Type": "Task",
            "Resource": "arn:aws:states:::databrew:startJobRun",
            "Parameters": {
               "Name": "sample-proj-job-1"
            },
            "Next": "NEXT_STATE"
          },
```

以下内容包括启动同步 DataBrew 作业的Task状态。

```
"DataBrew StartJobRun": {
           "Type": "Task",
           "Resource": "arn:aws:states:::databrew:startJobRun.sync",
           "Parameters": {
              "Name": "sample-proj-job-1"
           },
           "Next": "NEXT_STATE"
          },
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM策略。

使用 Step Functions 调用 Lambda

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化后的 Lambda 集成与 Lambda 软件开发工具包集成有何不同 AWS
>
> - 响应的 `Payload` 字段从转义的 Json 解析为 Json。
> - 如果响应中包含字段 `FunctionError`，或者在 Lambda 函数中引发了异常，则任务将失败。

有关管理状态输入、输出和结果的更多信息，请参阅Step Functions 中的输入和输出处理。

支持 AWS Lambda 的 API：

- `Invoke`
  - 请求语法
  - 支持的参数
    - `ClientContext`
    - `FunctionName`
    - `InvocationType`
    - `Qualifier`
    - `Payload`
  - 响应语法

> 中的Step Functions参数表示为 PascalCase
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

下面是调用 Lambda 函数的 `Task` 状态。

```
{
```

```
   "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
      }
   }
}
```

以下内容包括实施回调服务集成模式的 Task 状态。

```
{
   "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
      }
   }
}
```

当您调用 Lambda 函数时，执行将等待函数完成。如果使用回调任务调用 Lambda 函数，则在 Lambda 函数完成执行并返回结果之后，信号检测超时才会开始计数。只要 Lambda 函数在执行，信号检测超时就不会强制执行。

您也可以使用 InvocationType 参数异步调用 Lambda，如以下示例所示：

> **Note**
>
> 对于 Lambda 函数的异步调用，信号检测超时时间会立即开始。

```
{

  "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
    }
  }
}
```

Task 结果返回时，函数输出嵌套在元数据目录中。例如：

```
{

   "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
}
```

或者，您可以通过直接在“资源”字段中指定函数 ARN 来调用 Lambda 函数。以这种方式调用 Lambda 函数时，您无法指定 `.waitForTaskToken`，并且任务结果仅包含函数输出。

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
         "Resource":"arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
         "End": true
      }
   }
}
```

通过在 `Resource` 字段的 ARN 中指定相应选项，您可以调用特定的 Lambda 函数版本或别名。请参阅 Lambda 文档中的以下内容：

- AWS Lambda 版本控制
- AWS Lambda 别名

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

AWS Elemental MediaConvert 使用 Step Functions 进行管理

> **试用 Step Functions 和 MediaConvert**
>
> 了解如何在工作流程中使用 MediaConvert 优化的集成，从视频片段的开头检测和移除长度未知的 SMTPE 色条。阅读 2024 年 4 月 12 日的博客文章：低代码工作流程 AWS Elemental MediaConvert

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

优化的集成与标准 AWS SDK 集成有何不同

- 运行作业 (.sync) 集成模式可用。
- 没有针对等待具有任务令牌的回调集成模式请求响应的优化。

支持 MediaConvert 的 API：

- CreateJob
  - 请求语法
  - 支持的参数：
    - Role（必填）
    - Settings（必填）
    - CreateJobRequest（可选）
  - 响应语法 — 参见CreateJobResponse 架构

以下内容包括提交 MediaConvert 作业并等待其完成的Task状态。

```
{
    "StartAt": "MediaConvert_CreateJob",
    "States": {
        "MediaConvert_CreateJob": {
        "Type": "Task",
        "Resource": "arn:aws:states:::mediaconvert:createJob.sync",
        "Parameters": {
            "Role": "arn:aws:iam::111122223333:role/Admin",
            "Settings": {
            "OutputGroups": [
                {
                "Outputs": [
                    {
                    "ContainerSettings": {
                        "Container": "MP4"
                    },
                    "VideoDescription": {
                        "CodecSettings": {
                        "Codec": "H_264",
                        "H264Settings": {
```

```
                        "MaxBitrate": 1000,
                        "RateControlMode": "QVBR",
                        "SceneChangeDetect": "TRANSITION_DETECTION"
                    }
                    }
                },
                "AudioDescriptions": [
                    {
                    "CodecSettings": {
                        "Codec": "AAC",
                        "AacSettings": {
                        "Bitrate": 96000,
                        "CodingMode": "CODING_MODE_2_0",
                        "SampleRate": 48000
                        }
                    }
                    }
                ]
                }
            ],
            "OutputGroupSettings": {
                "Type": "FILE_GROUP_SETTINGS",
                "FileGroupSettings": {
                "Destination": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/"
                }
            }
            }
        ],
        "Inputs": [
            {
            "AudioSelectors": {
                "Audio Selector 1": {
                "DefaultSelection": "DEFAULT"
                }
            },
            "FileInput": "s3://DOC-EXAMPLE-SOURCE-BUCKET/DOC-EXAMPLE-SOURCE_FILE"
            }
        ]
        }
    },
    "End": true
    }
}
```

```
}
```

有关在Step Functions与一起使用时如何配置IAM权限的信息 MediaConvert，请参阅适用的 IAM 政策 AWS Elemental MediaConvert。

> 中的Step Functions参数表示为 PascalCase
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

SageMaker 使用 Step Functions 进行管理

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化后的 SageMaker 集成与 S SageMaker AWS DK 集成有何不同
>
> - 支持运行作业 (.sync) 集成模式。
> - 没有针对请求响应集成模式的优化。
> - 不支持等待具有任务令牌的回调集成模式。

支持 SageMaker 的 API 和语法：

- `CreateEndpoint`
  - 请求语法
  - 支持的参数：
    - `EndpointConfigName`
    - `EndpointName`
    - `Tags`
  - 响应语法
- `CreateEndpointConfig`
  - 请求语法
  - 支持的参数：

    - EndpointConfigName
    - KmsKeyId
    - ProductionVariants
    - Tags
  - 响应语法
- CreateHyperParameterTuningJob

> **Note**
>
> 此 API 操作支持 .sync 集成模式。

  - 请求语法
  - 支持的参数：
    - HyperParameterTuningJobConfig
    - HyperParameterTuningJobName
    - Tags
    - TrainingJobDefinition
    - WarmStartConfig
  - 响应语法
- CreateLabelingJob

> **Note**
>
> 此 API 操作支持 .sync 集成模式。

  - 请求语法
  - 支持的参数：
    - HumanTaskConfig
    - InputConfig
    - LabelAttributeName
    - LabelCategoryConfigS3Uri

    - LabelingJobAlgorithmsConfig
    - LabelingJobName
    - OutputConfig
    - RoleArn
    - StoppingConditions
    - Tags
  - 响应语法
- CreateModel
  - 请求语法
  - 支持的参数：
    - Containers
    - EnableNetworkIsolation
    - ExecutionRoleArn
    - ModelName
    - PrimaryContainer
    - Tags
    - VpcConfig
- CreateProcessingJob

> **Note**
>
> 此 API 操作支持 .sync 集成模式。

  - 请求语法
  - 支持的参数：
    - AppSpecification
    - Environment
    - ExperimentConfig
    - NetworkConfig
    - ProcessingInputs
    - ProcessingJobName

    - ProcessingOutputConfig
    - ProcessingResources
    - RoleArn
    - StoppingCondition
    - Tags
  - 响应语法
- CreateTrainingJob

> **Note**
>
> 此 API 操作支持 .sync 集成模式。

  - 请求语法
  - 支持的参数：
    - AlgorithmSpecification
    - HyperParameters
    - InputDataConfig
    - OutputDataConfig
    - ResourceConfig
    - RoleArn
    - StoppingCondition
    - Tags
    - TrainingJobName
    - VpcConfig
  - 响应语法
- CreateTransformJob

> **Note**
>
> 此 API 操作支持 .sync 集成模式。

> **Note**
>
> AWS Step Functions 不会自动为创建策略`CreateTransformJob`。您必须将内联策略附加到创建的角色。有关更多信息，请参阅此示例 IAM 策略：`CreateTrainingJob`。

- 请求语法
- 支持的参数：
  - `BatchStrategy`
  - `Environment`
  - `MaxConcurrentTransforms`
  - `MaxPayloadInMB`
  - `ModelName`
  - `Tags`
  - `TransformInput`
  - `TransformJobName`
  - `TransformOutput`
  - `TransformResources`
- 响应语法
- `UpdateEndpoint`
  - 请求语法
  - 支持的参数：
    - `EndpointConfigName`
    - `EndpointName`
  - 响应语法

SageMaker 转换 Job 示例

以下内容包括创建 Amazon SageMaker 转换任务的`Task`状态，其中指定了`DataSource`和的 Amazon S3 位置`TransformOutput`。

```
{
```

```
"SageMaker CreateTransformJob": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
  "Parameters": {
    "ModelName": "SageMakerCreateTransformJobModel-9iFBKsYti9vr",
    "TransformInput": {
      "CompressionType": "None",
      "ContentType": "text/csv",
      "DataSource": {
        "S3DataSource": {
          "S3DataType": "S3Prefix",
          "S3Uri": "s3://my-s3bucket-example-1/TransformJobDataInput.txt"
        }
      }
    },
    "TransformOutput": {
      "S3OutputPath": "s3://my-s3bucket-example-1/TransformJobOutputPath"
    },
    "TransformResources": {
      "InstanceCount": 1,
      "InstanceType": "ml.m4.xlarge"
    },
    "TransformJobName": "sfn-binary-classification-prediction"
  },
  "Next": "ValidateOutput"
},
```

SageMaker 训练 Job 示例

以下内容包括创建 Amazon SageMaker 训练作业的Task状态。

```
{
   "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 标注 Job 示例

以下内容包括创建 Amazon SageMaker 贴标任务的Task状态。

```
{
  "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 处理 Job 示例

以下内容包括创建 Amazon SageMaker 处理任务的Task状态。

```
{
  "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."
    }
  }
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 调用 Amazon SNS

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化的亚马逊 SNS 集成与亚马 AWS 逊 SNS 软件开发工具包集成有何不同
>
> 没有针对请求响应或等待具有任务令牌的回调集成模式的优化。

支持的 Amazon SNS API：

> Note
>
> 在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

- Publish
  - 请求语法
  - 支持的参数
    - Message
    - MessageAttributes
    - MessageStructure
    - PhoneNumber
    - Subject
    - TargetArn
    - TopicArn
  - 响应语法

> 中的Step Functions参数表示为 PascalCase
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

下面包括一个向 Amazon Simple Notification Service (Amazon SNS) 主题发布的任务状态。

```
{
 "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
    }
  }
}
```

传递动态值。您可以修改上面的示例，以动态传递来自此 JSON 有效负载的属性：

```
{
"input": {
  "message": "Hello world"
},
"SNSDetails": {
  "attribute1": "some value",
  "attribute2": "some other value",
}
}
```

将 `.$` 附加到 `StringValue` 字段：

```
"MessageAttributes": {
```

```
  "my_attribute_no_1": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute1"
  },
  "my_attribute_no_2": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute2"
  }
```

以下内容包括发布 Amazon SNS 主题，然后等待返回任务令牌的 Task 状态。请参阅 等待具有任务令牌的回调。

```
{
   "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
      }
   }
}
```

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 调用 Amazon SQS

Step Functions 可以直接从 Amazon States Language (ASL) 控制某些 AWS 服务。要了解更多信息，请参阅使用其他服务和将参数传递给服务 API。

> 优化的亚马逊 SQS 集成与亚马 AWS 逊 SQS 软件开发工具包集成有何不同
>
> 没有针对请求响应或等待具有任务令牌的回调集成模式的优化。

支持的 Amazon SQS API：

> **Note**
>
> 在 Step Functions 中，任务的最大输入或结果数据大小有一个配额。在向另一个服务发送数据或从另一个服务接收数据时，数据大小不得超过 256 KB（UTF-8 编码字符串）。请参阅 与状态机执行相关的配额。

- SendMessage

  支持的参数：

  - DelaySeconds
  - MessageAttribute
  - MessageBody
  - MessageDeduplicationId
  - MessageGroupId
  - QueueUrl
- Response syntax

> **中的Step Functions参数表示为 PascalCase**
>
> 即使原生服务 API 在 camelCase 中（例如 API 操作）`startSyncExecution`，您也可以在中指定参数 PascalCase，例如:。`StateMachineArn`

以下内容包括发送Amazon Simple Queue Service (Amazon SQS) 消息的 `Task` 状态。

```
{
 "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
    }
  }
}
```

下面包含一个发布到 Amazon SQS 队列，然后等待返回任务令牌的 Task 状态。请参阅 等待具有任务令牌的回调。

```
{
   "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
      }
   }
}
```

要了解有关在 Amazon SQS 中接收消息的更多信息，请参阅《Amazon Simple Queue Service 开发商指南》中的接收和删除消息。

有关在与其他 AWS 服务Step Functions一起使用时如何配置IAM权限的信息，请参阅集成服务的 IAM 策略。

将 AWS Step Functions 执行作为集成服务进行管理

Step Functions 与其自己的 API 集成以作为服务集成。这允许 Step Functions 直接从正在运行的执行的任务状态开始新的状态机执行。构建新的工作流时，请使用嵌套工作流执行降低主要工作流的复杂性并重用常用流程。

> 优化后的 Step Functions 集成与 Step Functions AWS SDK 集成有何不同
>
> - 运行作业 (.sync) 集成模式可用。
>
> 请注意，没有针对 请求响应 或 等待具有任务令牌的回调 集成模式的优化。

有关更多信息，请参阅下列内容：

- 从任务开始执行
- 使用其他服务
- 将参数传递给服务 API

支持的 Step Functions API 和语法：

- `StartExecution`
  - 请求语法
  - 支持的参数
    - `Input`
    - `Name`
    - `StateMachineArn`
  - 响应语法

下面包括一个 `Task` 状态，此状态开始执行另一个状态机并等待其完成。

```
{
   "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
}
```

下面包括一个 Task 状态，此状态开始执行另一个状态机。

```
{
   "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
}
```

以下内容包括实施回调服务集成模式的 Task 状态。

```
{
   "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
}
```

要将嵌套的工作流执行与启动它的父执行相关联，请传递一个特殊命名的参数，该参数包括从上下文对象中提取的执行 ID。启动嵌套执行时，使用名为 AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID 的参数。通过将 .$ 附加到参数名称并在上下文对象中通过 $$.Execution.Id 引用 ID 来传递执行 ID。有关更多信息，请参阅 访问上下文对象。

```
{
   "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
}
```

嵌套状态机返回以下内容：

资源	输出
startExecution.sync	字符串
startExecution.sync:2	JSON

两者都将等待嵌套状态机完成，但它们返回不同的 Output 格式。例如，如果您创建一个返回对象 { "MyKey": "MyValue" } 的 Lambda 函数，则会得到以下响应：

对于 startExecution.sync：

```
{
   <other fields>
   "Output": "{ \"MyKey\": \"MyValue\" }"
}
```

对于 startExecution.sync:2：

```
{
   <other fields>
   "Output": {
      "MyKey": "MyValue"
   }
}
```

为嵌套状态机配置 IAM 权限

父状态机使用轮询和事件来确定子状态机是否已完成执行。轮询需要权限，`states:DescribeExecution`而发送 EventBridge 到 Step Functions 的事件则需要`events:PutTargets`、`events:PutRule`和的权限`events:DescribeRule`。如果您的 IAM 角色中缺少这些权限，则在父状态机意识到子状态机的执行已完成之前，可能会有一段延迟。

对于为单个嵌套工作流执行调用 `StartExecution` 的状态机，应使用 IAM 策略限制该状态机的权限。

有关更多信息，请参阅 Step Functions 的 IAM 权限。

调用第三方 API

HTTP 任务是一种 任务状态 状态，可让您在工作流中调用任何公共第三方 API，例如 Salesforce 和 Stripe。要调用第三方 API，请配合使用 Task 状态与 `arn:aws:states:::http:invoke` 资源。然后，提供 API 端点配置详细信息，例如 API URL、您要使用的方法和身份验证详细信息。

如果您使用 Workflow Studio 构建包含 HTTP 任务的状态机，则 Workflow Studio 会自动为 HTTP 任务生成一个带 IAM 策略的执行角色。有关更多信息，请参阅 在 Workflow Studio 中测试 HTTP 任务的角色。

主题

- HTTP 任务定义
- HTTP 任务字段
- HTTP 任务的身份验证
- 合并 EventBridge 连接和 HTTP 任务定义数据
- 在请求正文上应用 URL 编码
- 运行 HTTP 任务的 IAM 权限
- HTTP 任务示例

- 测试 HTTP 任务
- 不支持的 HTTP 任务响应

HTTP 任务定义

ASL 定义表示具有 http:invoke 资源的 HTTP 任务。以下 HTTP 任务定义调用了一个会返回所有客户列表的 Stripe API。

```
"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 任务字段

HTTP 任务的定义中包括以下字段。

Resource（必填）

要指定任务类型，请在 Resource 字段中提供其 ARN。对于 HTTP 任务，您可以按如下方式指定 Resource 字段。

```
"Resource": "arn:aws:states:::http:invoke"
```

Parameters（必填）

包含 ApiEndpoint、Method 和 ConnectionArn 字段，这些字段提供有关您要调用的第三方 API 的信息。Parameters 还包含可选字段，例如 Headers 和 QueryParameters。

您可以像在Parameters字段Parameters中一样指定静态 JSON 和JsonPath语法的组合。有关更多信息，请参阅 将参数传递给服务 API。

ApiEndpoint（必填）

指定您要调用的第三方 API 的 URL。要将查询参数附加到 URL，请使用 QueryParameters 字段。以下示例展示了如何调用 Stripe API 来获取所有客户的列表。

```
"ApiEndpoint":"https://api.stripe.com/v1/customers"
```

您也可以使用JsonPath语法指定参考路径，以选择包含第三方 API 网址的 JSON 节点。例如，假设您想使用特定客户 ID 调用 Stripe 的某个 API。想象一下，您已提供以下状态输入。

```
{
    "customer_id": "1234567890",
    "name": "John Doe"
}
```

要使用 Stripe API 检索该客户 ID 的详细信息，请指定 ApiEndpoint，如以下示例中所示。此示例使用了内置函数和参考路径。

```
"ApiEndpoint.$":"States.Format('https://api.stripe.com/v1/customers/{}',
 $.customer_id)"
```

在运行时，Step Functions 解析 ApiEndpoint 的值，如下所示。

```
https://api.stripe.com/v1/customers/1234567890
```

Method（必填）

指定要用于调用第三方 API 的 HTTP 方法。您可以在 HTTP 任务中指定以下方法之一：GET、POST、PUT、DELETE、PATCH、OPTIONS 或 HEAD。

例如，要使用 GET 方法，请按如下方式指定 Method 字段。

```
"Method": "GET"
```

您也可以在运行时使用参考路径来指定方法。例如，**"Method.$": "$.myHTTPMethod"**。

Authentication（必填）

包含指定如何对第三方 API 调用进行身份验证的 ConnectionArn 字段。Step Functions 支持使用 Amazon EventBridge 的连接资源对指定 ApiEndpoint 进行身份验证。

`ConnectionArn`（必填）

指定 EventBridge 连接 ARN。

HTTP 任务需要EventBridge 连接，该连接可以安全地管理 API 提供商的身份验证凭证。连接会指定用于授权第三方 API 的授权类型和凭证。使用连接可以帮助您避免将密钥（例如 API 密钥）硬编码到状态机定义中。在连接中，您还可以指定 `Headers`、`QueryParameters` 和 `RequestBody` 参数。

创建 EventBridge 连接时，您需要提供身份验证详细信息。有关如何对 HTTP 任务进行身份验证的更多信息，请参阅HTTP 任务的身份验证。

以下示例说明了如何在 HTTP 任务定义中指定 `Authentication` 字段。

```
"Authentication": {
  "ConnectionArn": "arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
}
```

`Headers`（可选）

为 API 端点提供额外的上下文和元数据。您可以将标头指定为字符串或 JSON 数组。

您可以在 EventBridge 连接以及 HTTP 任务的 `Headers` 字段中指定标头。建议您不要在 `Headers` 字段中包含 API 提供程序的身份验证详细信息。建议您将这些详细信息包含在 EventBridge 连接中。

Step Functions 会将您在 EventBridge 连接中指定的标头添加到您在 HTTP 任务定义中指定的标头。如果定义和连接中存在相同的标头键，则 Step Functions 会使用 EventBridge 连接中为这些标头指定的相应值。有关 Step Functions 如何执行数据合并的更多信息，请参阅合并 EventBridge 连接和 HTTP 任务定义数据。

以下示例指定了将包含在第三方 API 调用中的标头：`content-type`。

```
"Headers": {
  "content-type": "application/json"
}
```

您也可以在运行时使用参考路径来指定标头。例如，**`"Headers.$": "$.myHTTPHeaders"`**。

Step Functions 可设置 `User-Agent`、`Range` 和 `Host` 标头。Step Functions 会根据您要调用的 API 设置 `Host` 标头的值。以下是这些标头的一个示例。

```
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1,
Range: bytes=0-262144,
Host: api.stripe.com
```

您不能在 HTTP 任务定义中使用以下标头。如果您使用这些标头，HTTP 任务将失败并显示 `States.Runtime` 错误。

- A-IM
- Accept-Charset
- Accept-Datetime
- Accept-Encoding
- Cache-Control
- Connection
- Content-Encoding
- Content-MD5
- Date
- Expect
- Forwarded
- From
- Host
- HTTP2-Settings
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Origin
- Pragma
- Proxy-Authorization
- Referer
- Server

- TE
- Trailer
- Transfer-Encoding
- Upgrade
- Via
- Warning
- x-forwarded-*
- x-amz-*
- x-amzn-*

`QueryParameters`（可选）

在 API URL 的末尾插入键值对。您可以将查询参数指定为字符串、JSON 数组或 JSON 对象。当调用第三方 API 时，Step Functions 会自动对查询参数进行 URL 编码。

例如，假设您想调用 Stripe API 来搜索使用美元 (USD) 进行交易的客户。想象一下，您已提供以下 `QueryParameters` 作为状态输入。

```
"QueryParameters": {
  "currency": "usd"
}
```

在运行时，Step Functions 会将 `QueryParameters` 附加到 API URL，如下所示。

```
https://api.stripe.com/v1/customers/search?currency=usd
```

您也可以在运行时使用参考路径来指定查询参数。例如，**`"QueryParameters.$": "$.myQueryParameters"`**。

如果您在 EventBridge 连接中指定了查询参数，则 Step Functions 会将这些查询参数添加到您在 HTTP 任务定义中指定的查询参数。如果定义和连接中存在相同的查询参数键，则 Step Functions 会使用 EventBridge 连接中为这些标头指定的相应值。有关 Step Functions 如何执行数据合并的更多信息，请参阅合并 EventBridge 连接和 HTTP 任务定义数据。

`Transform`（可选）

包含 `RequestBodyEncoding` 和 `RequestEncodingOptions` 字段。默认情况下，Step Functions 会将请求正文作为 JSON 数据发送到 API 端点。

如果您的 API 提供程序接受 form-urlencoded 请求正文，请使用 Transform 字段为请求正文指定 URL 编码。您还必须将 content-type 标头指定为 application/x-www-form-urlencoded。Step Functions 然后会自动对请求正文进行 URL 编码。

RequestBodyEncoding

指定请求正文的 URL 编码。您可以指定以下值之一：NONE 或 URL_ENCODED。

- NONE：HTTP 请求正文将是 RequestBody 字段的序列化 JSON。这是默认值。
- URL_ENCODED：HTTP 请求正文将是 RequestBody 字段的 URL 编码表单数据。

RequestEncodingOptions

用于确定在您将 RequestBodyEncoding 设置为 URL_ENCODED 的情况下，要为请求正文中的数组使用的编码选项。

Step Functions 支持以下数组编码选项。有关这些选项的更多信息及其示例，请参阅在请求正文上应用 URL 编码。

- INDICES：使用数组元素的索引值对数组进行编码。Step Functions 默认使用此编码选项。
- REPEAT：针对数组中的每个项重复使用一个键。
- COMMAS：将键中的所有值编码为以逗号分隔的值列表。
- BRACKETS：针对数组中的每个项重复使用一个键，并为该键附加一个方括号 []，以表示它是一个数组。

以下示例为请求正文数据设置了 URL 编码。它还指定在请求正文中为数组使用 COMMAS 编码选项。

```
"Transform": {
  "RequestBodyEncoding": "URL_ENCODED",
  "RequestEncodingOptions": {
    "ArrayFormat": "COMMAS"
  }
}
```

RequestBody（可选）

接受您在状态输入中提供的 JSON 数据。在中RequestBody，您可以指定静态 JSON 和JsonPath语法的组合。例如，假设您提供以下状态输入：

```
{
    "CardNumber": "1234567890",
    "ExpiryDate": "09/25"
}
```

要在运行时在请求正文中使用 `CardNumber` 和 `ExpiryDate` 的这些值，您可以在请求正文中指定以下 JSON 数据。

```
"RequestBody": {
  "Card": {
    "Number.$": "$.CardNumber",
    "Expiry.$": "$.ExpiryDate",
    "Name": "John Doe",
    "Address": "123 Any Street, Any Town, USA"
  }
}
```

如果您要调用的第三方 API 需要 `form-urlencoded` 请求正文，则您必须为请求正文数据指定 URL 编码。有关更多信息，请参阅 在请求正文上应用 URL 编码。

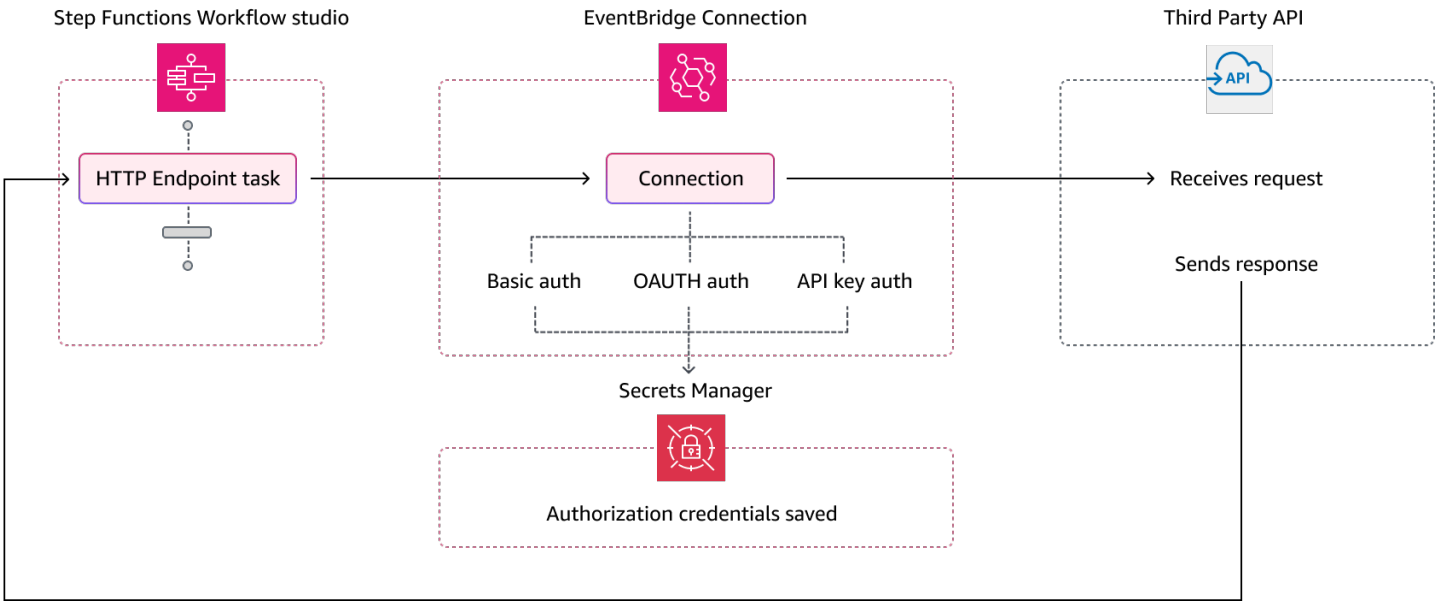
HTTP 任务的身份验证

HTTP 任务需要EventBridge 连接，该连接可以安全地管理 API 提供商的身份验证凭证。连接会指定用于授权第三方 API 的授权类型和凭证。使用连接可以帮助您避免将密钥（例如 API 密钥）硬编码到状态机定义中。 EventBridge 连接支持基本、OAuth 和 API 密钥授权方案。

创建 EventBridge 连接时，您需要提供身份验证详细信息。您还可以包含授权 API 所需的标头、正文和查询参数。您必须在调用第三方 API 的任何 HTTP 任务中包含连接 ARN。

当您创建连接并添加授权参数时， EventBridge 会在中创建密钥 AWS Secrets Manager。在此密钥中，以加密形式 EventBridge 存储连接和授权参数。要成功创建或更新连接，必须使用有权使用 S AWS 账户 ecrets Manager 的。有关状态机访问 EventBridge 连接所需的IAM权限的更多信息，请参阅运行 HTTP 任务的 IAM 权限。

下图显示了 Step Functions 如何使用 EventBridge 连接处理第三方 API 调用的身份验证。



合并 EventBridge 连接和 HTTP 任务定义数据

调用 HTTP 任务时，您可以在 EventBridge 连接和 HTTP 任务定义中指定数据。此数据包括 `Headers`、`QueryParameters` 和 `RequestBody` 参数。在调用第三方 API 之前，Step Functions 会将请求正文与连接正文参数合并，除非您的请求正文为字符串且连接正文参数非空。在这种情况下，HTTP 任务将失败并显示 `States.Runtime` 错误。

如果 HTTP 任务定义和 EventBridge 连接中指定了任何重复的键，则 Step Functions 会用连接中的值覆盖 HTTP 任务中的值。

以下列表说明了在调用第三方 API 之前 Step Functions 如何合并数据：

- 标头：Step Functions 会将您在连接中指定的任何标头添加到 HTTP 任务 `Headers` 字段中的标头。如果标头键之间存在冲突，Step Functions 会使用连接中为这些标头指定的值。例如，如果您在 HTTP 任务定义和 EventBridge 连接中都指定了 `content-type` 标头，则 Step Functions 会使用连接中指定的 `content-type` 标头值。
- 查询参数：Step Functions 会将您在连接中指定的任何查询参数添加到 HTTP 任务 `QueryParameters` 字段中的查询参数。如果查询参数键之间存在冲突，Step Functions 会使用连接中为这些查询参数指定的值。例如，如果您在 HTTP 任务定义和 EventBridge 连接中都指定了 `maxItems` 查询参数，则 Step Functions 会使用连接中指定的 `maxItems` 查询参数值。
- 主体参数
  - Step Functions 会将连接中指定的任何请求正文值添加到 HTTP 任务 `RequestBody` 字段中的请求正文。如果请求正文键之间存在冲突，Step Functions 会使用连接中为请求正文指定的值。

例如，假设您在 HTTP 任务定义和 EventBridge 连接的 RequestBody 中都指定了 Mode 字段。Step Functions 会使用您在连接中指定的 Mode 字段值。

- 如果您将请求正文指定为字符串而不是 JSON 对象，并且 EventBridge 连接还包含请求正文，则 Step Functions 无法合并在这两个位置指定的请求正文。它会使 HTTP 任务失败，并显示 States.Runtime 错误。

在请求正文完成合并后，Step Functions 会应用所有转换并将请求正文序列化。

以下示例在 HTTP 任务和 EventBridge 连接中设置了 Headers、QueryParameters 和 RequestBody 字段。

HTTP 任务定义

```
{
  "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 连接

```
{
  "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"
        }
      ]
    }
  }
}
```

在此示例中，HTTP 任务和 EventBridge 连接中指定了重复的键。因此，Step Functions 用连接中的值覆盖 HTTP 任务中的值。以下代码段显示了 Step Functions 发送到第三方 API 的 HTTP 请求。

```
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"}
```

在请求正文上应用 URL 编码

默认情况下，Step Functions 会将请求正文作为 JSON 数据发送到 API 端点。如果您的第三方 API 提供程序需要 `form-urlencoded` 请求正文，则您必须为请求正文指定 URL 编码。Step Functions 然后会根据您选择的 URL 编码选项自动对请求正文进行 URL 编码。

您可以使用 `Transform` 字段指定 URL 编码。此字段包含 `RequestBodyEncoding` 字段，后者用于指定您是否要为请求正文应用 URL 编码。当您指定 `RequestBodyEncoding` 字段时，Step Functions 会在调用第三方 API 之前，将您的 JSON 请求正文转换为 `form-urlencoded` 请求正文。您还必须将 `content-type` 标头指定为 `application/x-www-form-urlencoded`，因为接受 URL 编码数据的 API 需要 `content-type` 标头。

为对请求正文中的数组进行编码，Step Functions 提供以下数组编码选项。

- `INDICES`：针对数组中的每个项重复使用一个键，并为该键附加一个方括号 []，以表示它是一个数组。此方括号包含数组元素的索引。添加索引可帮助您指定数组元素的顺序。Step Functions 默认使用此编码选项。

  例如，如果您的请求正文包含以下数组。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions 会将此数组编码为以下字符串。

  ```
  array[0]=a&array[1]=b&array[2]=c&array[3]=d
  ```

- `REPEAT`：针对数组中的每个项重复使用一个键。

  例如，如果您的请求正文包含以下数组。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions 会将此数组编码为以下字符串。

```
array=a&array=b&array=c&array=d
```

- COMMAS：将键中的所有值编码为以逗号分隔的值列表。

  例如，如果您的请求正文包含以下数组。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions 会将此数组编码为以下字符串。

  ```
  array=a,b,c,d
  ```

- BRACKETS：针对数组中的每个项重复使用一个键，并为该键附加一个方括号 []，以表示它是一个数组。

  例如，如果您的请求正文包含以下数组。

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions 会将此数组编码为以下字符串。

  ```
  array[]=a&array[]=b&array[]=c&array[]=d
  ```

运行 HTTP 任务的 IAM 权限

您的状态机执行角色必须具有 `states:InvokeHTTPEndpoint`、`events:RetrieveConnectionCredentials`、`secretsmanager:G` 和 `secretsmanager:DescribeSecret` 权限，HTTP 任务才能调用第三方 API。下面的 IAM 策略示例授予状态机角色调用 Stripe API 所需的最低权限。此 IAM 策略还向状态机角色授予访问特定 EventBridge 连接的权限，包括存储在 Secrets Manager 中的此连接的密钥。

```
{
    "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 任务示例

以下状态机定义显示了包含 Headers、QueryParameters Transform 和 RequestBody 参数的 HTTP 任务。该 HTTP 任务调用 Stripe API https://api.stripe.com/v1/invoices 来生成发票。该 HTTP 任务还使用 INDICES 编码选项为请求正文指定 URL 编码。

确保您已创建 EventBridge 连接。以下示例显示了使用基本授权类型创建的连接。

```
{
    "Type": "BASIC",
    "AuthParameters": {
```

```
        "BasicAuthParameters": {
            "Password": "myPassword",
            "Username": "myUsername"
         },
    }
}
```

切记用特定资源信息替换##文本。

```
{
  "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
    }
  }
}
```

要运行此状态机，请提供客户 ID 作为输入，如以下示例所示：

```
{
    "customer_id": "1234567890"
}
```

以下示例显示了 Step Functions 发送到 Stripe API 的 HTTP 请求。

```
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
```

测试 HTTP 任务

您可以通过控制台、软件开发工具包或使用TestState该 API AWS CLI 来测试 HTTP 任务。以下过程介绍如何在Step Functions控制台中使用 TestState API。您可以迭代测试 API 请求、响应和身份验证详细信息，直到 HTTP 任务按预期运行。

在 Step Functions 控制台中测试 HTTP 任务状态

1. 打开 Step Functions 控制台。
2. 选择创建状态机开始创建状态机，或选择包含 HTTP 任务的现有状态机。

   如果您要在现有状态机中测试任务，请参阅步骤 4。
3. 在 Workflow Studio 的设计模式中，直观地配置 HTTP 任务。或者选择代码模式，从本地开发环境中复制粘贴状态机定义。
4. 在设计模式下，从 Workflow Studio 的 Inspector 面板中选择测试状态。
5. 在测试状态对话框中，执行以下操作：

   a. 对于执行角色，选择一个执行角色来测试状态。如果您没有权限足够的角色来配置 HTTP 任务，请参阅在 Workflow Studio 中测试 HTTP 任务的角色，创建一个角色。

   b. （可选）针对测试提供所选状态需要的任何 JSON 输入。

   c. 对于检查级别，保留默认选择信息。此级别会显示 API 调用的状态和状态输出。这对于快速检查 API 响应很有用。

   d. 选择开始测试。

   e. 如果测试成功，则状态输出会显示在测试状态对话框的右侧。如果测试失败，系统会显示错误。

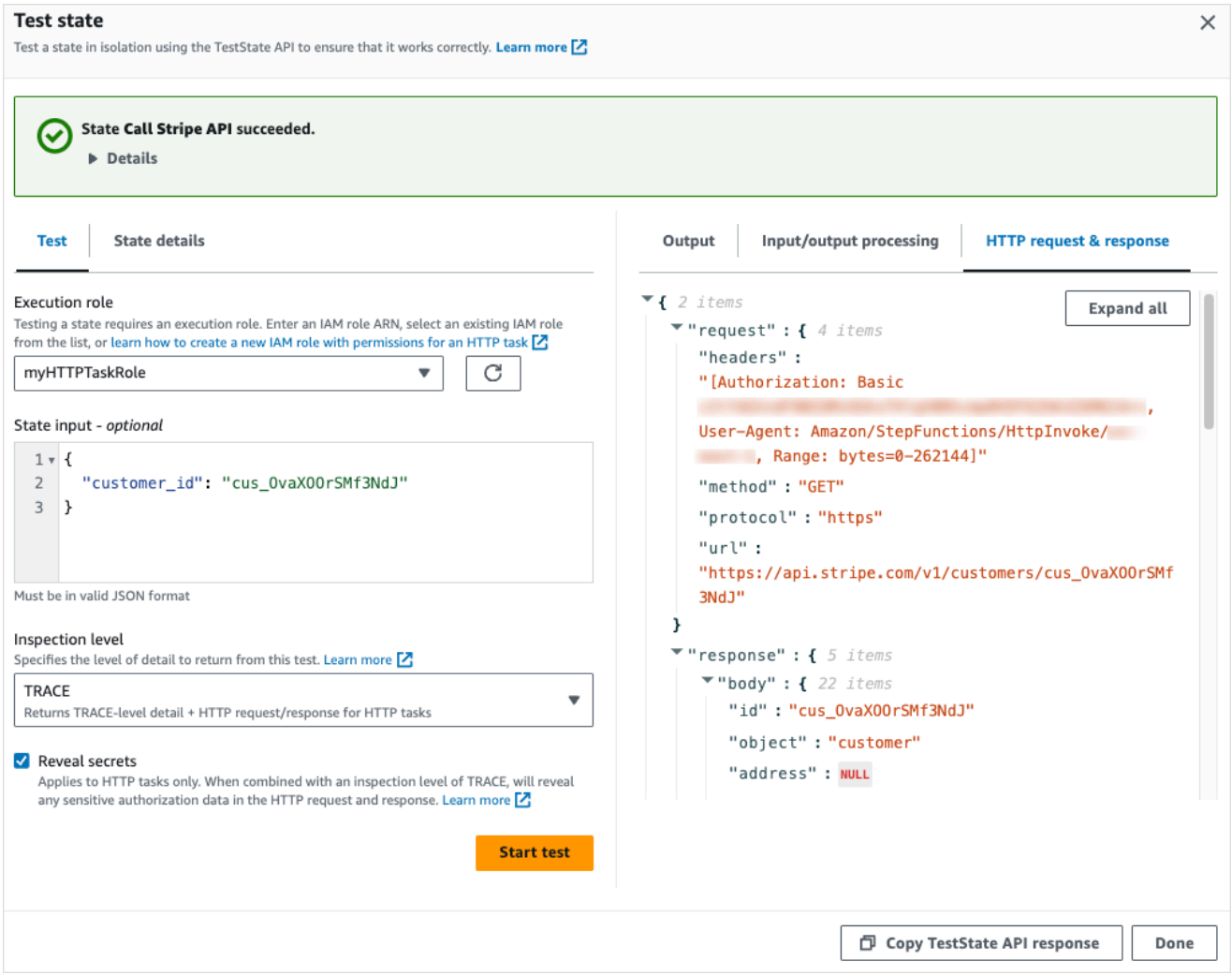
      在对话框的状态详细信息选项卡中，您可以看到状态定义和指向 EventBridge 连接的链接。

   f. 将检查级别更改为跟踪。此级别会显示原始的 HTTP 请求和响应，对于验证标头、查询参数和其他特定于 API 的详细信息非常有用。

   g. 选中显示密钥复选框。结合跟踪检查级别，此设置可让您查看 EventBridge 连接插入的敏感数据，例如 API 密钥。您用于访问控制台的 IAM 用户身份必须具有执行 `states:RevealSecrets` 操作的权限。如果没有此权限，则在您开始测试时 Step

Functions 会抛出访问遭拒错误。有关设置 `states:RevealSecrets` 权限的 IAM 策略示例，请参阅IAM使用 TestState API 的权限。

下图显示了成功的 HTTP 任务测试。此状态的检查级别设置为跟踪。下图中的 HTTP 请求和响应选项卡显示了第三方 API 调用的结果。



h. 选择开始测试。

i. 如果测试成功，您可以在 HTTP 请求和响应选项卡下看到 HTTP 详细信息。

不支持的 HTTP 任务响应

如果返回的响应满足以下任意条件，则 HTTP 任务失败并显示 `States.Runtime` 错误：

- 响应包含 `application/octet-stream`、`image/*`、`video/*` 或 `audio/*` Content-Type 标头。

- 响应无法读取为有效字符串。例如，二进制数据或图像数据。

服务集成模式

AWS Step Functions 直接与 Amazon 各州语言的服务集成。您可以使用三种服务集成模式来控制这些 AWS 服务：

- 调用服务，并让 Step Functions 在获得 HTTP 响应后立即进入下一个状态。
- 调用服务，并让 Step Functions 等待作业完成。
- 使用任务令牌调用服务并让 Step Functions 等待，直到该令牌与有效负载一起返回。

上述每种服务集成模式都受控于在任务定义的 "Resource" 字段中创建 URI 的方式。

调用集成服务的方式

- 请求响应
- 运行作业 (.sync)
- 等待具有任务令牌的回调

有关为集成服务配置 AWS Identity and Access Management (IAM) 的信息，请参阅集成服务的 IAM 策略。

请求响应

在任务状态的 "Resource" 字符串中指定服务，并且只提供资源时，Step Functions 将等待 HTTP 响应，然后进入下一个状态。Step Functions 不会等待作业完成。

以下示例显示了如何发布 Amazon SNS 主题。

```
"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"
}
```

此示例引用了 Amazon SNS 的 Publish API。调用 Publish API 后，工作流进入下一状态

> **Tip**
>
> 要将使用请求响应服务集成模式的示例工作流部署到您的工作流程 AWS 账户，请参阅模块 2- AWS Step Functions 研讨会的请求响应。

运行作业 (.sync)

对于诸如 AWS Batch 和 Amazon ECS 之类的集成服务，Step Functions 可以等待请求完成后再进入下一个状态。要让 Step Functions 等待，请使用在资源 URI 之后附加的 .sync 后缀，在任务状态定义中指定 "Resource" 字段。

例如，提交 AWS Batch 作业时，使用状态机定义中的"Resource"字段，如本示例所示。

```
"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"
}
```

将 .sync 部分附加到资源 Amazon 资源名称 (ARN) 上意味着 Step Functions 会等待作业完成。在调用 AWS Batch submitJob 后，工作流会暂停。作业完成后，Step Functions 将进入下一状态。欲了解更多信息，请参阅 AWS Batch 示例项目：管理批处理作业（AWS Batch、Amazon SNS）。

如果使用此 (.sync) 服务集成模式的任务中止，并且 Step Functions 无法取消该任务，则集成服务可能会向您收取额外费用。在以下情况下，任务可能被中止：

- 状态机执行被停止。
- 并行状态的另一个分支因未捕获的错误而失败。
- Map 状态的一次迭代失败并出现未捕获的错误。

Step Functions 将尽力尝试取消任务。例如，如果 Step Functions `states:startExecution.sync` 任务中止，它将调用 Step Functions `StopExecution` API 操作。但是，Step Functions 可能无法取消该任务。原因包括但不限于：

- 您的 IAM 执行角色没有权限进行相应的 API 调用。
- 发生临时服务中断。

当您使用 `.sync` 服务集成模式时，Step Functions 会使用轮询来监控任务的状态，并且会消耗分配的配额和事件。对于同一账户内的`.sync`调用，Step Functions 使用 EventBridge 事件并轮询您在状态中指定的 API。`Task`对于跨账户的 `.sync` 调用，Step Functions 仅使用轮询。例如，对于`states:StartExecution.sync`，Step Functions 对 DescribeExecutionAPI 执行轮询并使用您分配的配额。

> **Tip**
>
> 要将使用 Run a Job (.sync) 服务集成模式的示例工作流部署到您的 AWS 账户，请参阅 The Workshop 的模块 3-运行作业 (.sync)。 AWS Step Functions

要查看等待作业完成的集成服务支持的列表 (`.sync`)，请参阅 Step Functions 的优化集成。

> **Note**
>
> 使用 `.sync` 模式的服务集成需要额外的 IAM 权限。有关更多信息，请参阅 集成服务的 IAM 策略。

在某些情况下，您可能希望 Step Functions 在任务完全完成之前继续工作流。实现的方法与等待具有任务令牌的回调服务集成模式相同。为此，请将任务令牌传递给作业，然后使用 `SendTaskSuccess` 或 `SendTaskFailure` API 调用将其返回。Step Functions 会使用您在该调用中提供的数据来完成任务、停止监控作业并继续工作流。

等待具有任务令牌的回调

回调任务提供了一种暂停工作流程，直到返回任务令牌的方法。任务可能需要等待人员批准、与第三方集成或调用旧式系统。对于此类任务，您可以暂停 Step Functions，直到工作流执行达到一年的服务限额（参阅与状态限制相关的配额），然后等待外部流程或工作流完成。对于这些情况，Step Functions

允许您将任务令牌传递给 AWS SDK 服务集成以及某些经过优化的服务集成。任务将会暂停，直到它通过 SendTaskSuccess 或 SendTaskFailure 调用接收到该任务令牌。

如果使用回调任务令牌的 Task 状态超时，则会生成一个新的随机令牌。您可以从上下文对象访问任务令牌。

> **Note**
>
> 任务令牌必须包含至少 1 个字符，并且不能超过 1024 个字符。

要.waitForTaskToken与 S AWS DK 集成一起使用，您使用的 API 必须具有用于放置任务令牌的参数字段。

> **Note**
>
> 您必须从同一 AWS 账户中的委托人那里传递任务代币。如果您使用其他 AWS 账户的委托人发送代币，则代币将无法使用。

> **Tip**
>
> 要将使用回调任务令牌服务集成模式的示例工作流程部署到您 AWS 账户，请参阅模块 4-使用 AWS Step Functions 研讨会的任务令牌等待回调。

要查看支持等待任务令牌的集成服务的列表 (.waitForTaskToken)，请参阅 Step Functions 的优化集成。

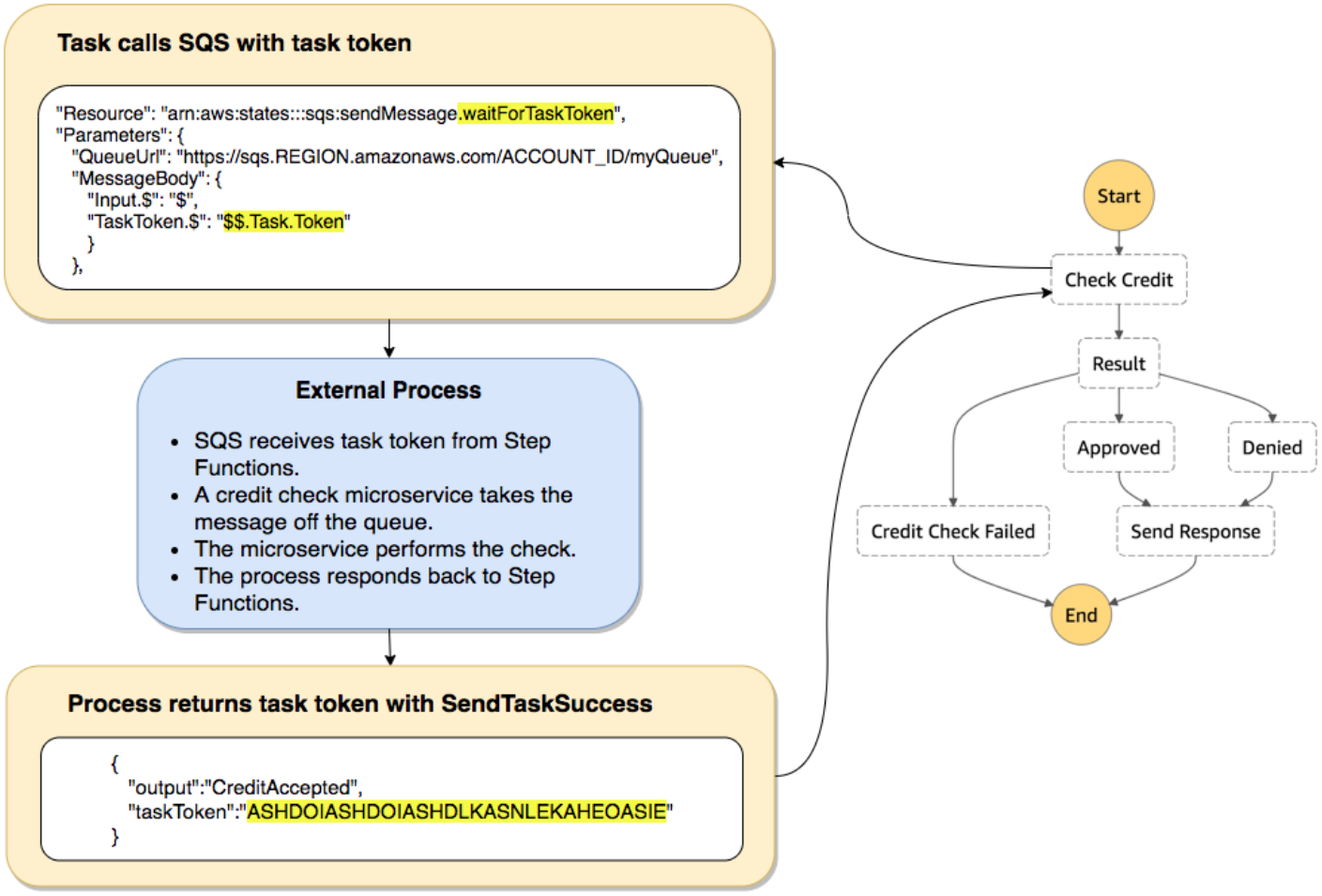
主题

- 任务令牌示例
- 从上下文对象获取令牌
- 为等待任务配置检测信号超时

任务令牌示例

在此示例中，Step Functions 工作流需要与外部微服务集成，才能在批准工作流中执行信用检查。Step Functions 会发布一条 Amazon SQS 消息，该消息中包含任务令牌。外部系统与 Amazon SQS 集成，

并将消息从队列中拉出。完成后，它将返回结果和原始任务令牌。然后 Step Functions 继续其工作流。



引用 Amazon SQS 的任务定义的 "Resource" 字段包括附加到末尾的 .waitForTaskToken。

```
"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"
}
```

这会指示 Step Functions 暂停并等待任务令牌。使用 .waitForTaskToken 指定资源时，可以使用特殊路径名称 ($$.Task.Token) 在状态定义的 "Parameters" 字段中访问任务令牌。初始 $$. 指定访问上下文对象的路径，并在正在运行的执行中获取当前任务的任务令牌。

完成后，外部服务会调用 SendTaskSuccess 或 SendTaskFailure，其中包含 taskToken。只有这样，工作流程才会继续进入下一个状态。

> **Note**
>
> 要避免在流程无法与 SendTaskSuccess 或 SendTaskFailure 一起发送任务令牌时无限期地等待，请参阅为等待任务配置检测信号超时。

从上下文对象获取令牌

上下文对象是一个内部 JSON 对象，其中包含有关执行的信息。与状态输入一样，可以在执行期间使用来自 "Parameters" 字段的路径访问它。从任务定义中访问时，它包含有关特定执行的信息（包括任务令牌）。

```
{
    "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"
```

```
    }
}
```

您可以使用任务定义的 "Parameters" 字段内的特殊路径来访问任务令牌。要访问输入或上下文对象，首先通过将 .$ 附加到参数名称来指定参数将是路径。以下内容指定 "Parameters" 规范中输入和上下文对象的节点。

```
"Parameters": {
   "Input.$": "$",
   "TaskToken.$": "$$.Task.Token"
 },
```

在这两种情况下，将 .$ 添加到参数名称会告知 Step Functions 期望一个路径。在第一种情况下，"$" 是一个路径，其中包含整个输入。在第二种情况下，$$. 指定路径将访问上下文对象，$$.Task.Token 将参数设置为正在运行的执行的上下文对象中的任务令牌的值。

在 Amazon SQS 示例中，"Resource" 字段中的 .waitForTaskToken 告诉 Step Functions 等待任务令牌返回。"TaskToken.$": "**$$.Task.Token**" 参数将该令牌作为 Amazon SQS 消息的一部分传递。

```
"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"
}
```

有关上下文对象的更多信息，请参阅本指南中 输入和输出处理 部分中的 Context 对象。

为等待任务配置检测信号超时

等待任务令牌的任务将等到执行达到一年服务配额（请参阅 与状态限制相关的配额）。要避免执行卡顿，您可以在状态机定义中配置检测信号超时间隔。使用 HeartbeatSeconds 字段指定超时间隔。

```
{
```

```
  "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
    }
  }
}
```

在此状态机定义中，任务将消息推送到 Amazon SQS 并等待外部进程使用提供的任务令牌进行回调。`"HeartbeatSeconds": 600` 字段将检测信号超时间隔为 10 分钟。该任务将等待任务令牌通过以下 API 操作之一返回：

- `SendTaskSuccess`
- `SendTaskFailure`
- `SendTaskHeartbeat`

如果等待任务在该 10 分钟内未收到有效的任务令牌，则任务失败并显示 `States.Timeout` 错误名称。

有关更多信息，请参阅回调任务示例项目：回调模式示例（Amazon SQS、Amazon SNS、Lambda）。

将参数传递给服务 API

使用 `Task` 状态中的 `Parameters` 字段来控制传递给服务 API 的参数。

在 `Parameters` 字段中，必须在 API 操作中使用数组参数的复数形式。例如，如果您使用 `DescribeSnapshots` API 操作的筛选字段与 Amazon EC2 集成，则必须将该字段定义为 `Filters`。如果您没有使用复数形式，Step Functions 会返回以下错误：

```
The field Filter is not supported by Step Functions.
```

将静态 JSON 作为参数传递

您可以直接在状态机定义中包含 JSON 对象，以作为参数传递给资源。

例如，要为 AWS Batch 的 SubmitJob API 设置 RetryStrategy 参数，您可以在参数中包含以下内容。

```
"RetryStrategy": {
  "attempts": 5
}
```

也可以使用静态 JSON 传递多个参数。作为一个更完整的示例，下面是向名为 *myTopic* 的 Amazon SNS 主题发布任务规范中的 Resource 和 Parameters 字段。

```
"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"
       }
     }
  },
```

使用路径将状态输入作为参数传递

您可以使用路径将部分状态输入作为参数传递。路径是一个以 $ 开头的字符串，用于标识 JSON 文本中的组件。Step Functions 路径使用 JsonPath 语法。

要指定参数使用路径，请在参数名称后使用 .$ 结尾。例如，如果状态输入包含名为 message 的节点中的文本，就可以使用路径将该文本作为参数传递。

考虑以下状态输入：

```
{
```

```
  "comment": "A message in the state input",
  "input": {
    "message": "foo",
    "otherInfo": "bar"
  },
  "data": "example"
}
```

要将名为 message 的节点值作为参数传递，请指定以下语法：

```
"Parameters": {"myMessage.$": "$.input.message"},
```

然后，Step Functions 将值 foo 作为参数传递。

有关在 Step Functions 中使用参数的更多信息，请参阅以下内容：

- 输入和输出处理
- InputPath、参数和 ResultSelector

将上下文对象节点作为参数传递

除了静态内容和状态输入中的节点之外，您还可以将上下文对象中的节点作为参数传递。上下文对象是状态机执行期间存在的动态 JSON 数据。它包含状态机和当前执行的相关信息。您可以使用状态定义的 "Parameters" 字段中的路径访问上下文对象。

有关上下文对象以及如何从 "Parameters" 字段访问该数据的更多信息，请参阅以下内容：

- Context 对象
- 访问上下文对象
- 从上下文对象获取令牌

更改支持的 AWS SDK 集成的日志

下表汇总了服务最初与 Step Functions 集成的时间以及其集成 API 的最新更新时间。有关使用集成的详细信息，请参阅AWS SDK 服务集成。

> **Important**
>
> API 操作支持按季度发布一次。对已支持的操作（例如新参数）的更新可能不会立即可用。

服务	初始支持	已更新	
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		
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 成本优化中心	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 数据匹配服务	2024 年 1 月 18 日		
AWS Free Tier	2024 年 1 月 18 日		

服务	初始支持	已更新	
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		
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		

服务	初始支持	已更新	
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		
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		

服务	初始支持	已更新	
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		
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 资源探索器	2023 年 2 月 17 日		
AWS SimSpace Weaver	2023 年 2 月 17 日		
AWS Support App	2023 年 2 月 17 日		

服务	初始支持	已更新	
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		
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		

服务	初始支持	已更新	
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		
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		

服务	初始支持	已更新	
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		
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 成本和使用情况报告	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		

服务	初始支持	已更新	
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		
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	

服务	初始支持	已更新	
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		
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 数据	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	

服务	初始支持	已更新	
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	
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 策略建议	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	

服务	初始支持	已更新	
AWS Outposts	2021 年 9 月 30 日		
AWS Panorama	2022 年 4 月 19 日	2023 年 2 月 17 日	
Amazon Relationa l Database Service Performance Insights	2021 年 9 月 30 日		
AWS 价目表	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		

服务	初始支持	已更新	
AWS Server Migration Service	2021 年 9 月 30 日		
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	

服务	初始支持	已更新	
AWS Marketplace Metering Service	2021 年 9 月 30 日		
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		

服务	初始支持	已更新	
Amazon CloudFront	2021 年 9 月 30 日	2023 年 2 月 17 日	
Amazon CloudSearch	2021 年 9 月 30 日		
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		

服务	初始支持	已更新	
Amazon Connect Contact Lens	2021 年 9 月 30 日		
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	

服务	初始支持	已更新	
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	
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	

服务	初始支持	已更新	
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	
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		

服务	初始支持	已更新	
Amazon Kinesis Analytics V2	2021 年 9 月 30 日		
Amazon Kinesis Firehose	2021 年 9 月 30 日		
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		

服务	初始支持	已更新	
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		
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 Kafkax	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		

服务	初始支持	已更新	
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	
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		

服务	初始支持	已更新	
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	
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		

服务	初始支持	已更新	
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	
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		

服务	初始支持	已更新	
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		
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 的示例项目

在 AWS Step Functions 控制台中，您可以选择以下入门模板之一，将状态机部署到您的 AWS 账户。这些入门模板是随时可以运行的示例项目，可自动创建工作流原型和定义，以及项目的所有相关 AWS 资源。

您可以使用这些示例项目按原样部署和运行，也可以使用工作流原型在其基础上进行构建。如果在这些项目的基础上进行构建，Step Functions 会创建工作流原型，但不会部署工作流定义中列出的资源。

部署示例项目时，它们会提供一个功能完备的状态机，并为状态机的运行创建相关资源。创建示例项目后，Step Functions 将使用 AWS CloudFormation 创建由状态机引用的相关资源。

主题

- 管理批处理作业（AWS Batch、Amazon SNS）
- 管理容器任务（Amazon ECS、Amazon SNS）
- 传输数据记录（Lambda、DynamoDB、Amazon SQS）
- 任务状态投票 (Lambda,) AWS Batch
- 任务计时器（Lambda、Amazon SNS）
- 回调模式示例（Amazon SQS、Amazon SNS、Lambda）
- 管理 Amazon EMR 任务
- 运行 EMR Serverless 作业
- 在工作流内启动工作流（Step Functions、Lambda）
- 使用 Map 状态动态处理数据
- 使用分布式 Map 处理 CSV 文件
- 使用分布式 Map 处理 Amazon S3 存储桶中的数据
- 训练机器学习模型
- 调整机器学习模型
- 处理来自 Amazon SQS（快速工作流）的大批量消息
- 选择性检查点示例（快速工作流）
- 构建 AWS CodeBuild 项目（CodeBuild，亚马逊 SNS）
- 预处理数据并训练机器学习模型

- Lambda 编排示例
- 启动 Athena 查询
- 执行多个查询（Amazon Athena、Amazon SNS）
- 查询大型数据集（亚马逊 Athena、亚马逊 S3、亚马逊 SN AWS Glue S）
- 保持数据最新（亚马逊 Athena、Amazon S3、）AWS Glue
- 管理 Amazon EKS 集群
- 调用 API Gateway
- 使用 API Gateway 集成调用在 Fargate 上运行的微服务
- 将自定义事件发送到 EventBridge
- 调用同步快速工作流
- 使用 Amazon Redshift 运行 ETL/ELT 工作流（Lambda、Amazon Redshift 数据 API）
- 使用 Step Functions 和 AWS Batch 进行错误处理
- 分散一份 AWS Batch 工作
- AWS Batch 用 Lambda
- 使用 Amazon Bedrock 执行 AI 提示链接

管理批处理作业（AWS Batch、Amazon SNS）

此示例项目演示了如何提交 AWS Batch 作业，然后根据该作业是成功还是失败发送 Amazon SNS 通知。部署此示例项目将创建 AWS Step Functions 状态机、AWS Batch 作业和 Amazon SNS 主题。

在此项目中，Step Functions 使用状态机同步调用 AWS Batch 作业。然后它等待作业成功或失败，并发送一个 Amazon SNS 主题，其中包含有关作业是成功还是失败的消息。

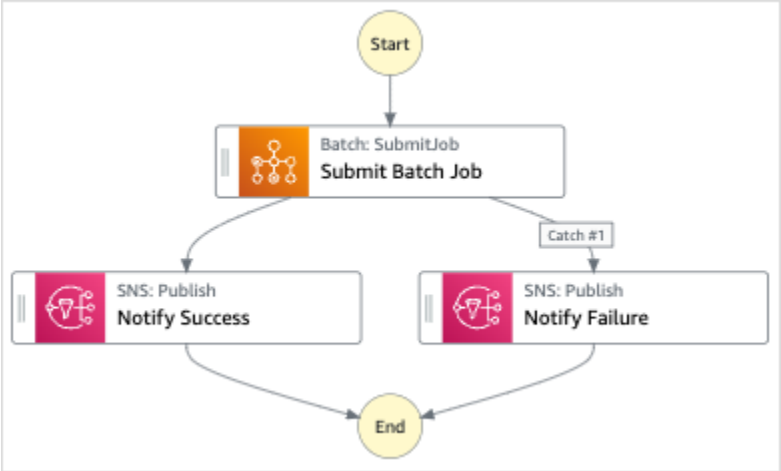
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Manage a batch job`**，然后从返回的搜索结果中选择管理批处理任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

该示例项目部署了以下资源：

- 一份 AWS Batch 工作
- 一个 Amazon SNS 主题
- AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了管理批处理任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon SNS 资源来与 AWS Batch 这些资源集成。

浏览此示例状态机，通过连接到`Resource`字段中的亚马逊资源名称 (ARN) AWS Batch 并传递`Parameters`到服务 API，了解 Step Functions 如何控制和亚马逊 SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

管理容器任务（Amazon ECS、Amazon SNS）

此示例项目演示了如何运行 AWS Fargate 任务，然后根据该任务是成功还是失败发送 Amazon SNS 通知。部署此示例项目将创建一个 AWS Step Functions 状态机、一个 Fargate 集群和一个 Amazon SNS 主题。

在此项目中，Step Functions 使用状态机同步调用 Fargate 任务。然后它等待任务成功或失败，并发送一个 Amazon SNS 主题，其中包含有关任务是成功还是失败的消息。

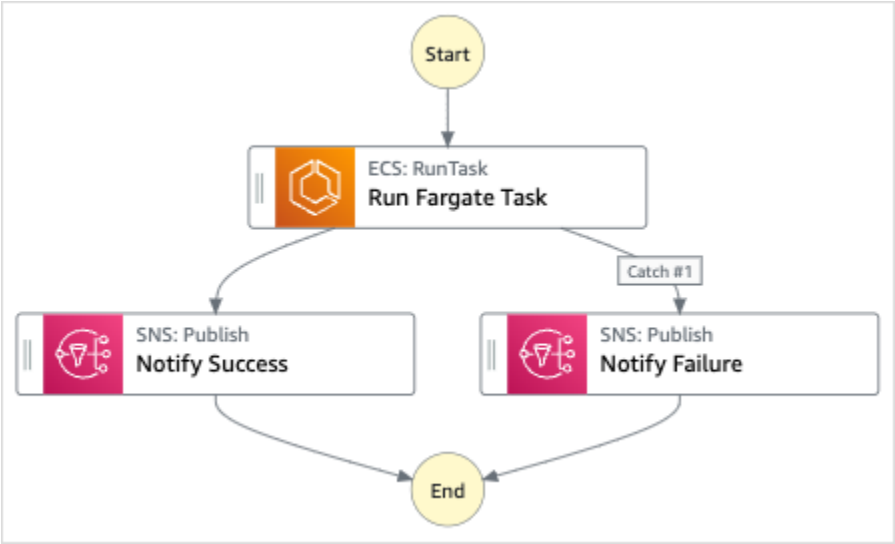
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Manage a container task`**，然后从返回的搜索结果中选择管理容器任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 AWS Fargate 集群
   - 一个 Amazon SNS 主题
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了管理容器任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon SNS 资源来与 AWS Fargate 这些资源集成。浏览此状态机示例，了解 Step Functions 如何使用状态机同步调用 Fargate 任务，等待任务成功或失败，并发送一个包含有关作业是成功还是失败的消息的 Amazon SNS 主题。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。最佳实操是在您的 IAM 策略仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

传输数据记录（Lambda、DynamoDB、Amazon SQS）

此示例项目演示了如何以迭代方式从 Amazon DynamoDB 表中读取项目并使用 Step Functions 状态机将这些项目发送到 Amazon SQS 队列。部署此示例项目将创建一个 Step Functions 状态机、一个 DynamoDB 表、一个 AWS Lambda 函数和一个 Amazon SQS 队列。

在此项目中，Step Functions 使用 Lambda 函数填充 DynamoDB 表格。状态机还使用 `for` 循环读取每个条目，然后将每个条目发送到 Amazon SQS 队列。

第 1 步：创建状态机并预置资源

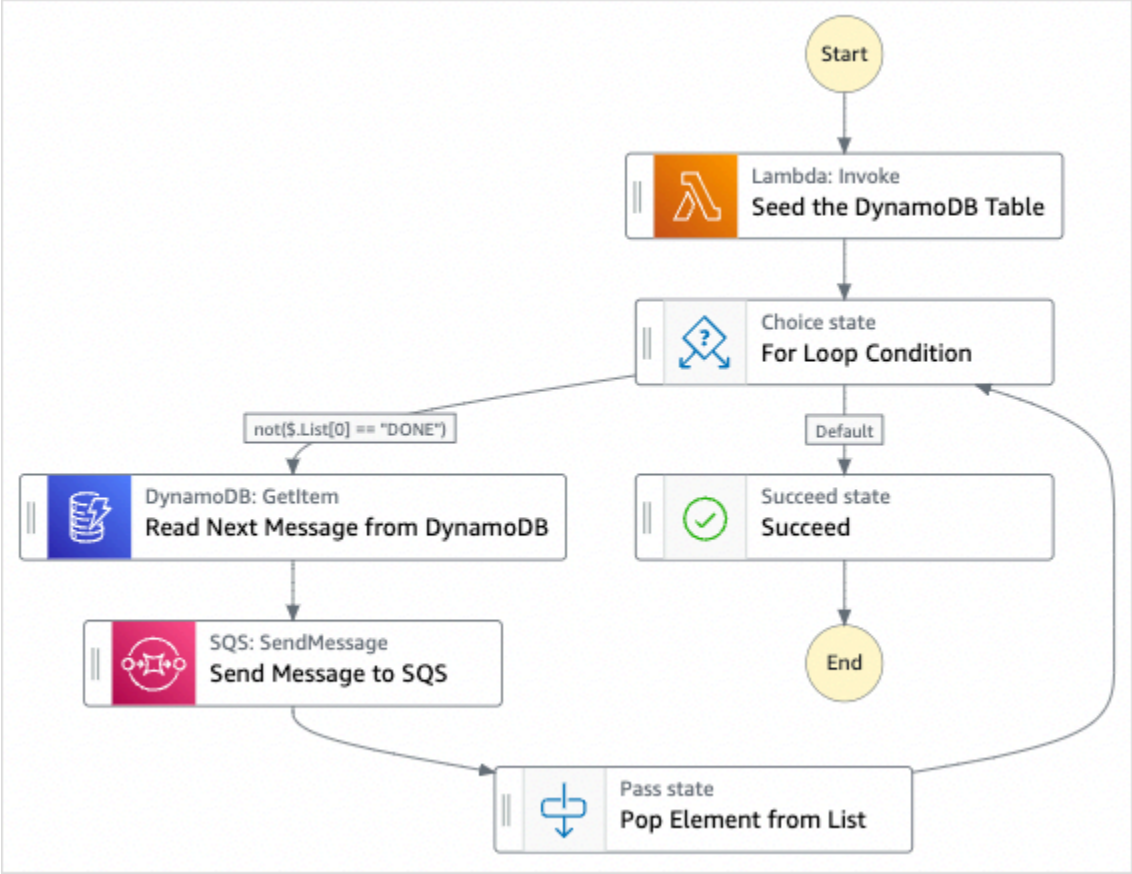
1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Transfer data records`**，然后从返回的搜索结果中选择传输数据记录。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Lambda 函数，用于为 DynamoDB 表制作种子
   - 一个 Amazon SQS 队列
   - 一个 DynamoDB 表

- AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了传输数据记录示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源来与 DynamoDB 和 Amazon SQS 集成。

浏览此示例状态机，了解 Step Functions 如何通过连接到 `Resource` 字段中的 Amazon 资源名称 (ARN)，以及向服务 API 传递 `Parameters` 来控制 DynamoDB 和 Amazon SQS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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"
    }
  }
}
```

有关传递参数和管理结果的更多信息，请参阅以下内容：

- 将参数传递给服务 API
- ResultPath

IAM 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。最佳实操是在您的 IAM 策略仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

任务状态投票 (Lambda,) AWS Batch

此示例项目创建了一个 AWS Batch 职位轮询器。它实现了一个 AWS Step Functions 状态机， AWS Lambda 用于创建检查 AWS Batch 任务的`Wait`状态循环。

此示例项目创建并配置所有资源，以便您的 Step Functions 工作流程将提交 AWS Batch 作业，并等待该任务完成后才成功结束。

> **Note**
>
> 您也可以在不使用 Lambda 函数的情况下实现此模式。有关 AWS Batch 直接控制的信息，请参见与其他服务 AWS Step Functions 一起使用。

此示例项目创建状态机、两个 Lambda 函数和一个 AWS Batch 队列，并配置相关的 IAM 权限。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

第 1 步：创建状态机并预置资源

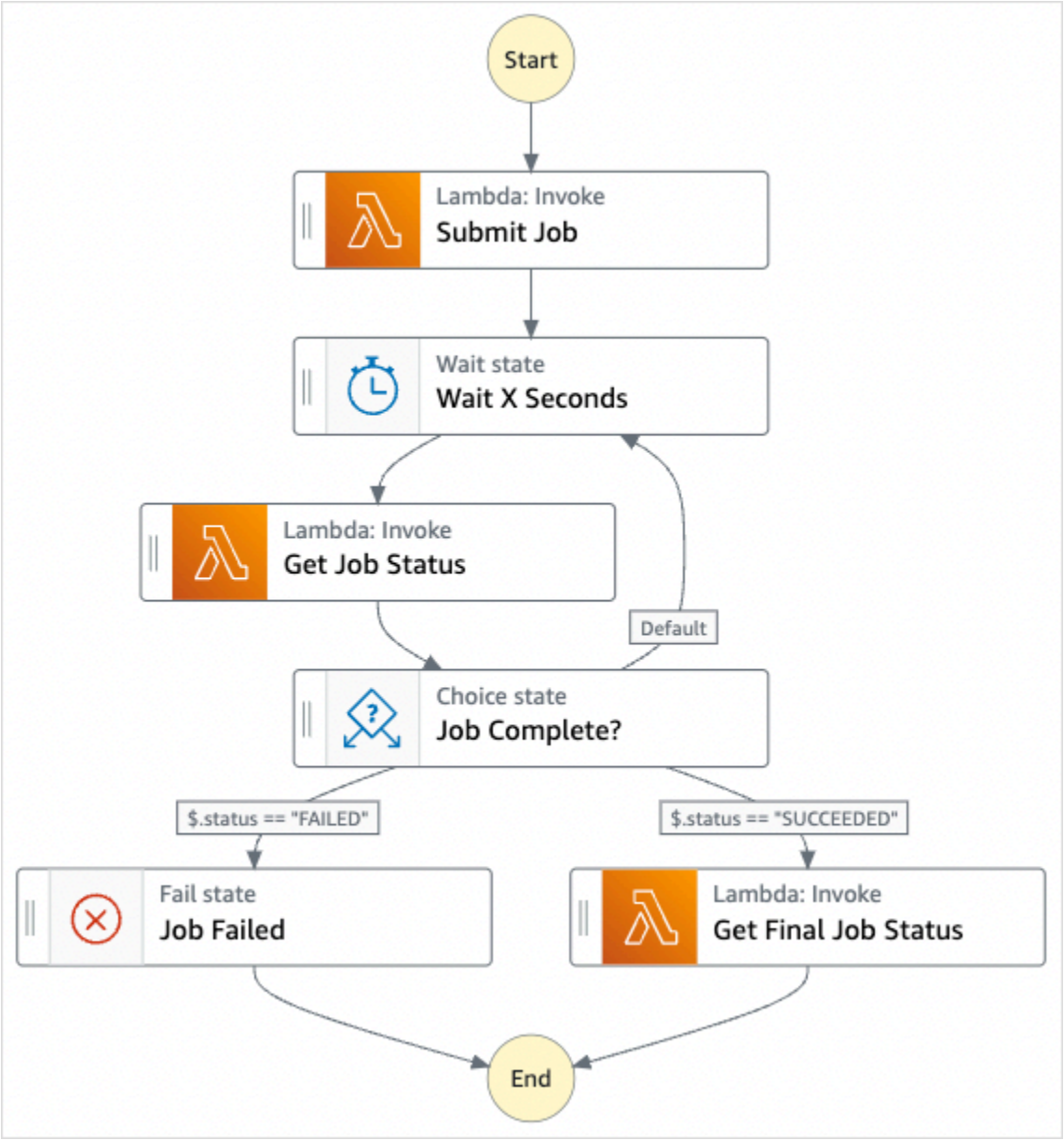
1. 打开 Step Functions 控制台，然后选择创建状态机。

2. 在搜索框中键入 **Job Poller**，然后从返回的搜索结果中选择任务轮询器。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 三个 Lambda 函数用于提交 AWS Batch 任务、获取已提交 AWS Batch 任务的当前状态和最终的任务完成状态。
   - 一份 AWS Batch 工作
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了任务轮询器示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

配置并部署完所有资源后，将显示启动执行对话框，其中包含类似于以下内容的示例输入。

```
{
    "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` 指示 `Wait` 状态每 60 秒循环。

- 在启动执行对话框中，执行以下操作：

  1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

     > **Note**
     >
     > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

  2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

     如果您选择运行演示，则无需提供任何执行输入。

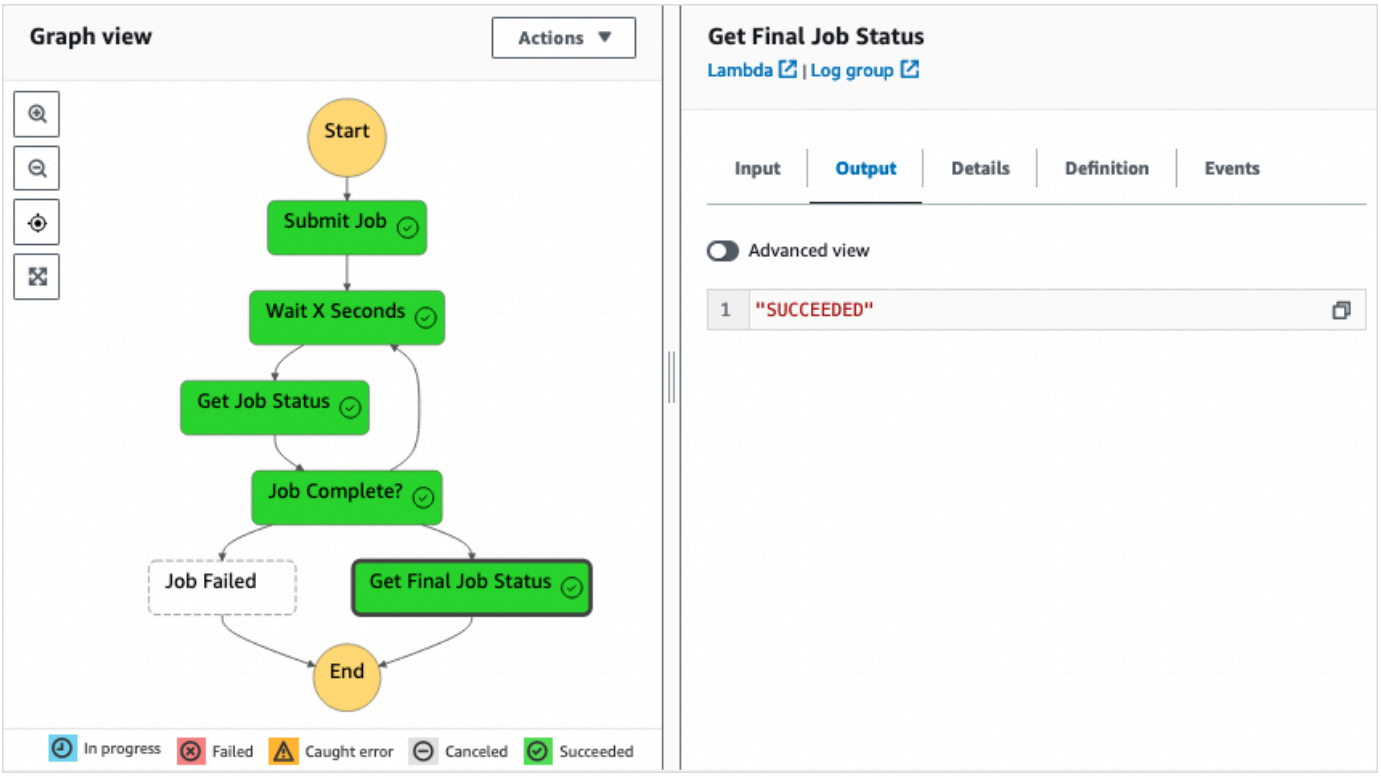
     > **Note**
     >
     > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

  3. 选择启动执行。
  4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

     要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

  例如，要查看 AWS Batch 任务的变化状态和执行的循环结果，请选择输出选项卡。

  下图显示了图表视图中的执行状态图。它还会在输出选项卡中显示所选步骤的执行输出。



示例状态机代码

此示例项目中的状态机与集成 AWS Lambda 以提交作 AWS Batch 业。浏览此示例状态机，了解 Step Functions 如何控制 Lambda 和。 AWS Batch

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅[与其他服务 AWS Step Functions 一起使用]。

```
{
    "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
      }
    }
```

```
}
```

任务计时器（Lambda、Amazon SNS）

本示例项目将创建一个任务计时器。它实现了一个实现 AWS Step Functions 状态的状态机，并使用一个`Wait`发送亚马逊简单通知服务 (Amazon SNS) Simple SNS Service 通知的 AWS Lambda 函数。Wait 状态是等待触发器执行单个工作单元的状态类型。

> **Note**
>
> 此示例项目实现了一个发送亚马逊简单通知服务 (Amazon SNS) 通知的 AWS Lambda 函数。您也可以直接从 Amazon States Language 发送 Amazon SNS 通知。请参阅 与其他服务 AWS Step Functions 一起使用。

此示例项目创建状态机、Lambda 函数和 Amazon SNS 主题，并配置 AWS Identity and Access Management 相关的 (IAM) 权限。有关使用任务计时器 示例项目创建的资源的更多信息，请参阅下面内容：

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

- AWS CloudFormation 用户指南
- Amazon Simple Notification Service 开发人员指南
- AWS Lambda 开发人员指南
- IAM 入门指南

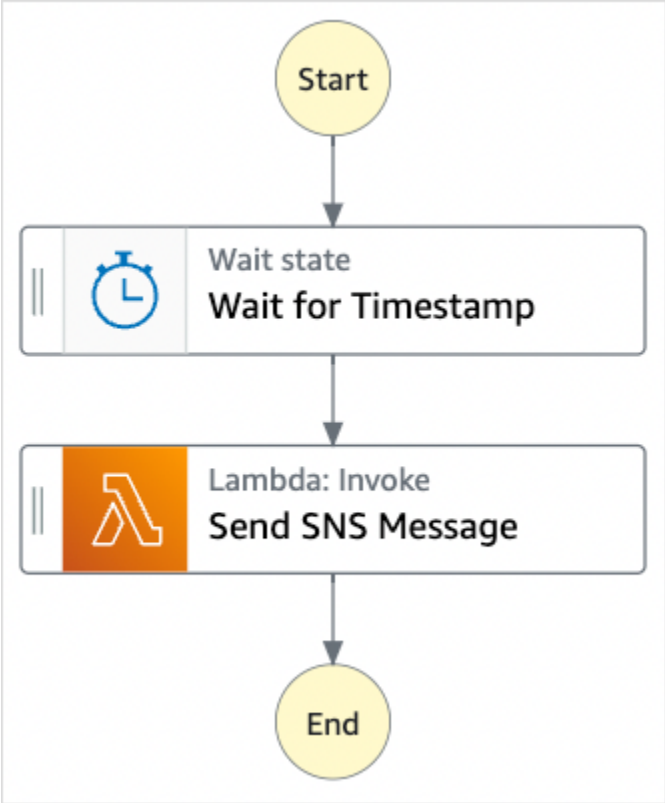
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Task Timer`**，然后从返回的搜索结果中选择任务计时器。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

该示例项目部署了以下资源：

- 一个用于发送 Amazon SNS 通知的 Lambda 函数。
- 一台 AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了任务计时器示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

配置并部署完所有资源后，将显示启动执行对话框，其中包含类似于以下内容的示例输入。

```
{
    "jobName": "my-job",{
  "topic": "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-TaskTimercc68840e-c3d3-42a8-911e-821b7ce248e5-SNSTopic-44UjcFxzhACT",
  "message": "HelloWorld",
  "timer_seconds": 10
}
```

- 在启动执行对话框中，执行以下操作：

1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

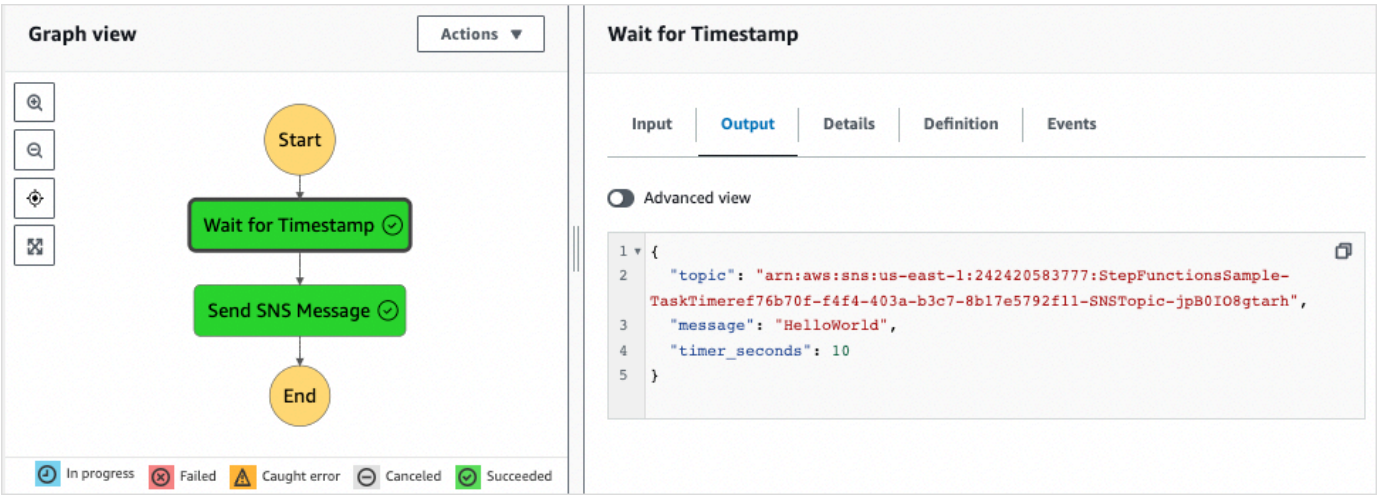
   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。

4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

例如，下图显示了所选步骤等待时间戳的输出。该步骤的输出将作为发送 SNS 消息步骤的输入。



回调模式示例（Amazon SQS、Amazon SNS、Lambda）

此示例项目演示了如何在任务期间 AWS Step Functions 暂停，并等待外部进程返回任务启动时生成的任务令牌。

部署此示例项目并开始执行时，将执行以下步骤：

1. Step Functions 将包含任务令牌的消息传递到 Amazon Simple Queue Service (Amazon SQS) 队列
2. 然后 Step Functions 暂停，等待该令牌返回。
3. Amazon SQS 队列会触发一个`SendTaskSuccess`使用相同任务令牌调用的 AWS Lambda 函数。
4. 收到任务令牌后，工作流程将继续。
5. `"Notify Success"` 任务将发布一条Amazon Simple Notification Service (Amazon SNS) 消息，告知已收到回调。

要了解如何在 Step Functions 中实施回调模式，请参阅等待具有任务令牌的回调。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

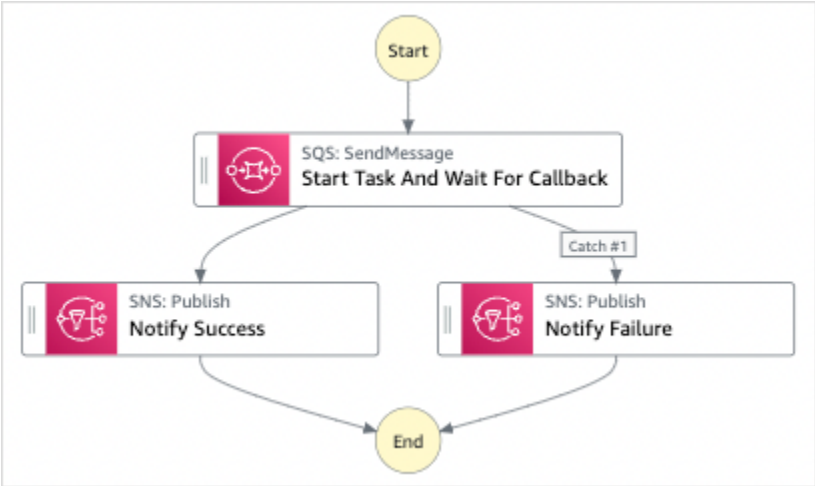
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Callback pattern example`**，然后从返回的搜索结果中选择回调模式示例。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon SQS 消息队列。
   - 一个调用 Step Functions API 操作的 Lambda 函数。SendTaskSuccess
   - 一个 Amazon SNS 主题，用于通知任务成功或失败，指明工作流能否继续。
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

下图显示了回调模式示例示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

  > **Important**
  >
  > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

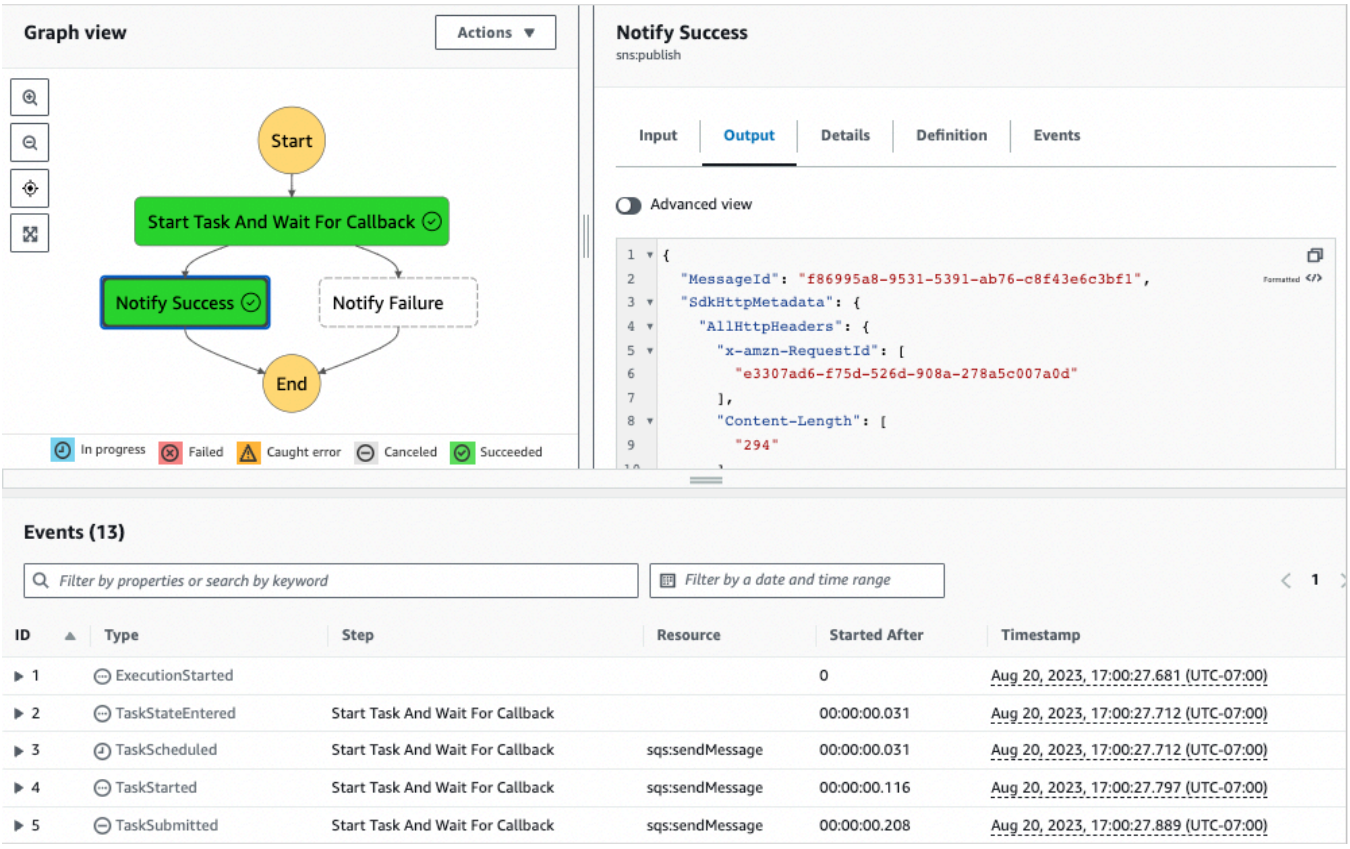
      如果您选择运行演示，则无需提供任何执行输入。

      > **Note**
      >
      > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

例如，要查看 Step Functions 在工作流中的进展情况以及从 Amazon SQS 接收回调的情况，请查看事件表中的条目。下图显示了通知成功步骤的执行输出。它还显示了执行事件历史记录中的前五个事件。展开每个活动可查看有关该事件的更多详细信息。



Lambda 回调示例

要了解此示例项目的各个组件是如何协同工作的，请参阅在您的 AWS 账户中部署的资源。例如，这是使用任务令牌调用 Step Functions 的 Lambda 函数。

```
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);
        });
    }
};
```

管理 Amazon EMR 任务

此示例项目演示了 Amazon EMR 和集成。 AWS Step Functions

它展示如何创建 Amazon EMR 集群、添加多个步骤并运行它们，然后终止集群。

> **Important**
>
> Amazon EMR 没有免费定价套餐。运行示例项目将产生成本。您可以在 Amazon EMR 定价页面上找到定价信息。Amazon EMR 服务集成的可用性取决于 Amazon EMR API 的可用性。因此，此示例项目可能无法在某些 AWS 地区正常运行。请查看 Amazon EMR 文档，了解特殊区域的限制。

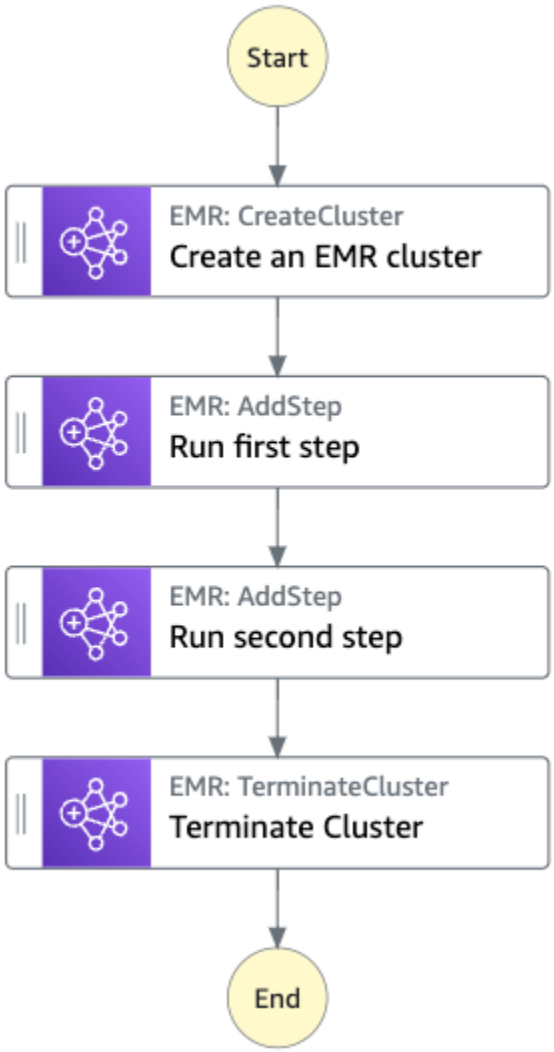
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Manage an EMR job`**，然后从返回的搜索结果中选择管理 EMR 任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon S3 存储桶
   - 一个 Amazon EMR 集群
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了管理 EMR 任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源来与 Amazon EMR 集成。浏览此示例状态机，了解 Step Functions 如何使用状态机同步调用 Amazon EMR 任务，等待任务成功或失败，并终止集群。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。最佳实操是在您的 IAM 策略仅包含这些必需的权限。

```
{
    "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"
            ]
        }
    ]
}
```

以下策略可确保 addStep 具有足够的权限。

```
{
    "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"
            ]
        }
    ]
}
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

运行 EMR Serverless 作业

此示例项目演示了如何创建和启动 EMR Serverless 应用程序。该项目还展示了如何在该应用程序中运行多个作业。

此示例项目创建状态机、支持 AWS 资源并配置相关的 IAM 权限。探索此示例项目，以了解如何使用 Step Functions 状态机运行 EMR Serverless 作业，或将其用作您自己的项目的起点。

> **Important**
>
> EMR Serverless 没有免费定价套餐。运行示例项目将产生成本。您可以在 Amazon EMR Serverless 定价页面上找到定价信息。
> 另外，EMR Serverless 服务集成的可用性取决于 EMR Serverless API 的可用性。因此，此示例项目在某些 AWS 区域可能无法正常工作或无法使用。有关 EMR Serverless 在 AWS 区域的可用性信息，请参阅其它考虑因素主题。

AWS CloudFormation 模板和其他资源

您可以使用 CloudFormation 模板来部署此示例项目。此模板在您的中创建了以下资源 AWS 账户：

- 一个 Step Functions 状态机
- 状态机的执行角色。此角色授予您的状态机访问其他 AWS 服务 资源所需的权限，例如EMR ServerlessCreateApplication操作。
- EMR Serverless 的作业执行角色。此角色授予 EMR Serverless 作业运行在代表您调用其他服务时可承担的权限。

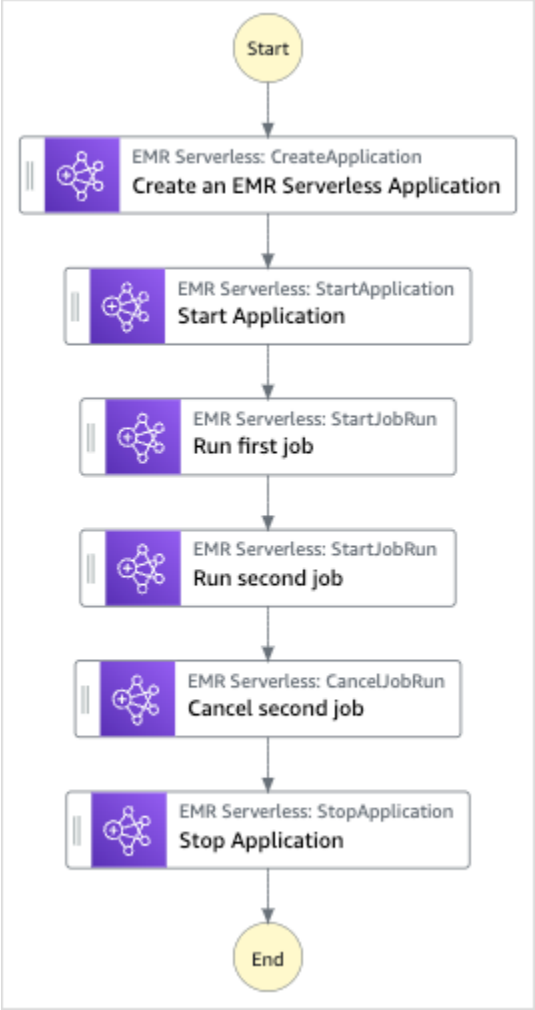
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`EMR Serverless`**，然后从返回的搜索结果中选择运行 EMR Serverless 作业。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

- 一个 Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了运行 EMR Serverless 作业示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

在工作流内启动工作流（Step Functions、Lambda）

此示例项目演示了如何使用 AWS Step Functions 状态机启动其他状态机执行。有关从其他状态机中启动状态机这姓的信息，请参阅从 Task 状态启动工作流执行。

第 1 步：创建状态机并预置资源

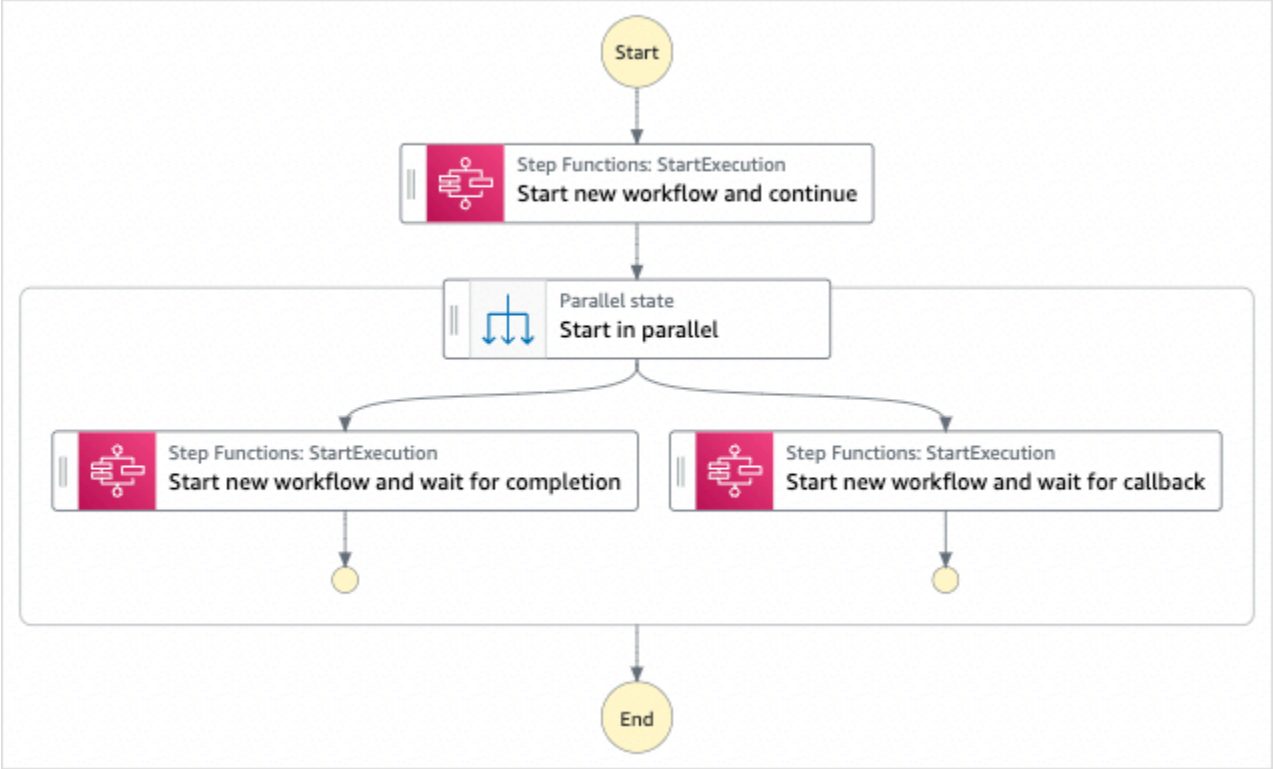
1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Start a workflow within a workflow`**，然后从返回的搜索结果中选择 在工作流内启动工作流。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个额外的状态机。此状态机的执行由您运行的状态机启动。
   - 一个回调 Lambda 函数。此函数在额外的状态机中，用于实现回调机制。

- AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图展示了在工作流内启动工作流示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

  > **Important**
  >
  > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 AWS Lambda 通过将参数直接传递给这些资源来集成另一台状态机。

浏览此示例状态机以了解 Step Functions 如何对这一其他状态机调用 `StartExecution` API 操作。它并行启动该其他状态机的两个实例：使用 运行作业 (.sync) 模式启动一个实例，使用 等待具有任务令牌的回调 模式启动另一个实例。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
            }
          }
        }
      ]
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用 Map 状态动态处理数据

该示例项目演示使用 Map 状态的动态并行性。

在这个项目中，Step Functions 使用一个 AWS Lambda 函数将消息从亚马逊 SQS 队列中提取出来，并将这些消息的 JSON 数组传递给一个Map状态。对于队列中的每个消息，状态机将消息写入 DynamoDB，调用其他 Lambda 函数以从 Amazon SQS 中删除消息，然后将消息发布到 Amazon SNS 主题。

有关 Map 状态和 Step Functions 服务集成的更多信息，请参阅以下内容。

- Map
- 与其他服务 AWS Step Functions 一起使用

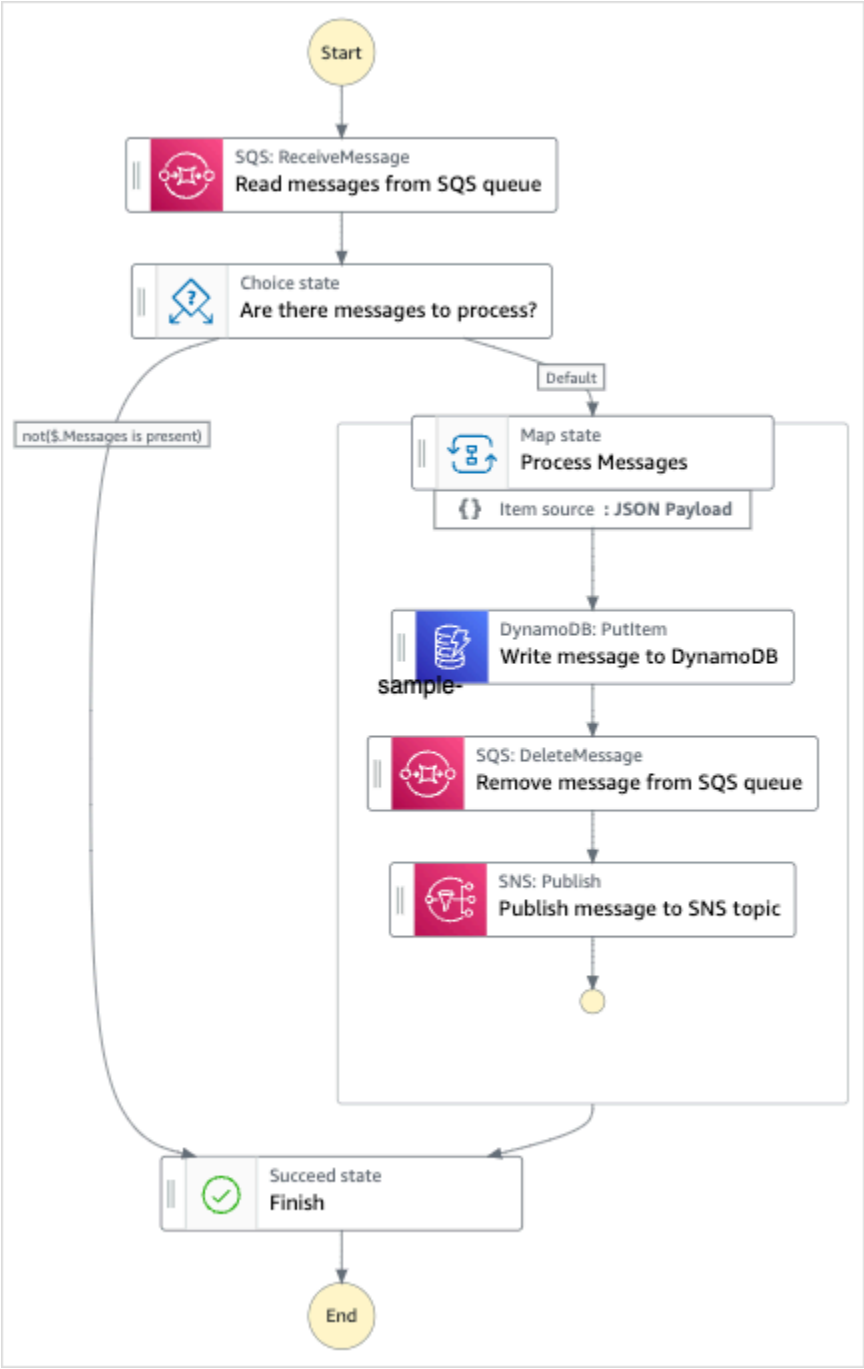
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **Dynamically process data with a Map state**，然后从返回的搜索结果中选择使用 Map 状态动态处理数据。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

- 一个 Amazon SQS 队列，Map 状态从中以迭代方式读取和删除消息。
- 一个 DynamoDB 表，Map 状态以迭代方式将消息写入其中。
- 一个 Amazon SNS 主题，Step Functions 将其从 Amazon SQS 队列中读取的消息发布到该主题。
- 两个 AWS Lambda 函数
- 一台 AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图展示了使用 Map 状态动态处理数据示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控

制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

部署了示例项目的资源后，您必须在开始运行状态机之前，将各个项添加到 Amazon SQS 队列并订阅 Amazon SNS 主题。

第 2 步：订阅 Amazon SNS 主题

1. 打开 Amazon SNS 控制台。
2. 选择主题，然后选择 `Map` 状态示例项目创建的主题。

   这个名字将类似于 MapSampleProj-snstopic- 1cqo4hq3ir1kn。
3. 选择创建订阅。

将显示创建订阅页面，其中列出该主题的主题 ARN。

4. 对于协议，选择电子邮件。
5. 在 Endpoint (端点) 下，输入一个电子邮件地址以订阅该主题。
6. 选择创建订阅。

> **Note**
>
> 您必须先在电子邮件中确认订阅，然后才能激活该订阅。

7. 在相关账户中打开订阅确认电子邮件，然后打开确认订阅 URL。

   订阅确认！页面随即显示出来。

第 3 步：向 Amazon SQS 队列添加消息

1. 打开 Amazon SQS 控制台。
2. 选择 `Map` 状态示例项目创建的队列。

   这个名字将类似于 MapSampleProj-sqsqueue- 1udic9vzdorn7。
3. 选择发送和接收消息。
4. 在发送和接收消息页面上，输入消息并选择发送消息。
5. 输入另一条消息，然后选择发送消息。继续输入更多消息，直到 Amazon SQS 队列中有几条消息为止。

第 4 步：运行状态机

> **Note**
>
> Amazon SNS 中的队列具有最终一致性。为了获得最佳结果，请在填充队列与运行状态机执行之间等待几分钟。

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

   > **Note**
   >
   > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源来与 Amazon SQS、Amazon SNS 和 Lambda 集成。

浏览此示例状态机，通过连接到 `Resource` 字段中的 Amazon 资源名称 (ARN)，并将 `Parameters` 传递给服务 API，查看 Step Functions 如何控制 Lambda、DynamoDB 和 Amazon SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用分布式 Map 处理 CSV 文件

本示例项目演示了如何使用分布式 Map 状态迭代使用 Lambda 函数生成的 CSV 文件的 1 万行。CSV 文件包含客户订单的配送信息，并存储在 Amazon S3 存储桶中。分布式 Map 对 CSV 文件中的 10 行进行批量迭代，以进行数据分析。

分布式 Map 包含一个 Lambda 函数，用于检测任何延迟订单。分布式 Map 还包含一个内联 Map，用于处理批次中的延迟订单，并以数组形式返回这些延迟订单。对于每个延迟订单，内联 Map 都会向 Amazon SQS 队列发送一条消息。最后，本示例项目会将 Map Run 结果存储到 AWS 账户中的另一个 Amazon S3 存储桶中。

使用分布式 Map，一次最多可运行 1 万个并行子工作流。在本示例项目中，分布式 Map 的最大并发量设置为 1000，即限制为 1000 个并行子工作流执行。

此示例项目创建状态机、支持 AWS 资源并配置相关的 IAM 权限。探索此示例项目，了解如何使用分布式 Map 编排大规模并行工作负载，或将其作为您自己项目的起点。

AWS CloudFormation 模板和其他资源

您可以使用 CloudFormation 模板来部署此示例项目。此模板在您的中创建了以下资源 AWS 账户：

- 一个 Step Functions 状态机。
- 状态机的执行角色。此角色授予您的状态机访问其他 AWS 服务 资源所需的权限，例如 Lambda 函数的 Invoke 操作。
- 一个名为 `CSVGeneratorFunction` 的 Lambda 函数，用于生成包含客户订单详细信息的 CSV 文件。
- CSV 生成器 Lambda 函数的执行角色。此角色授予函数访问其他角色的权限 AWS 服务。
- 一个 Amazon S3 输入存储桶，用于存储生成的 CSV 文件。
- 延迟订单检测 Lambda 函数，用于分析 CSV 文件数据并检测任何延迟订单。
- 延迟订单 Lambda 函数的执行角色。此角色授予函数访问其他角色的权限 AWS 服务。
- 一个 Amazon S3 输出存储桶，用于存储客户订单分析结果。
- 一个 Amazon SQS 队列，Step Functions 为每个延迟的订单向该队列发送消息。这些消息包含客户及其订单的 ID。

- 存储与状态机执行历史相关的信息的 CloudWatch 日志组。

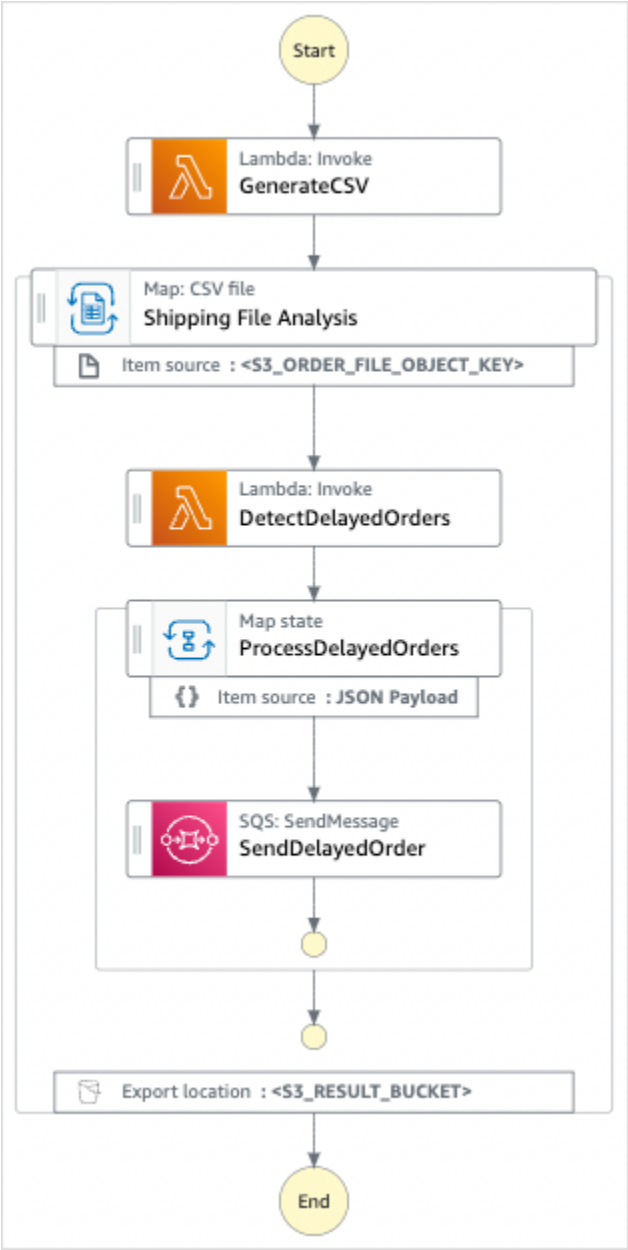
> **Important**
>
> 标准费用适用于每项服务。

第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Distributed Map to process a CSV file in S3`**，然后从返回的搜索结果中选择分布式 Map 处理 S3 中的 CSV 文件。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   有关将为此示例项目创建的资源信息，请参阅 AWS CloudFormation 模板和其他资源。

   下图显示了分布式 Map 处理 S3 中的 CSV 文件示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language (ASL) 定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

配置并部署完所有资源后，您可以运行状态机。

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   a. （可选）以 JSON 格式键入输入值以运行您的示例项目。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

b. 选择启动执行。

c. （可选）Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   执行完成后，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。

   - 有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。
   - 有关在控制台中查看分布式 Map 状态 执行的更多信息，请参阅检查 Map Run。

d. （可选）查看导出到 Amazon S3 存储桶的执行结果。这些结果包括数据，例如执行输入和输出、ARN 和执行状态。有关更多信息，请参阅 ResultWriter。

使用分布式 Map 处理 Amazon S3 存储桶中的数据

此示例项目演示了如何使用分布式 Map 状态 处理大规模数据，例如，分析历史天气数据并确定每月地球上平均温度最高的气象站。天气数据记录在 1.2 万多个 CSV 文件中，这些文件又存储在 Amazon S3 存储桶中。

此示例项目包括两个分布式 Map 状态，分别名为 Distributed S3 copy NOA Data 和 ProcessNOAAData。分布式 S3 副本 NOA 数据会迭代名noaa-gsod-pds为的 Amazon S3 公共存储桶中的 CSV 文件，并将它们复制到您的中的 Amazon S3 存储桶中。 AWS 账户ProcessNOAAData 对复制的文件进行迭代，并包含一个用于执行温度分析的 Lambda 函数。

示例项目首先通过调用 ListObjectsV2 API 操作来检查 Amazon S3 存储桶的内容。根据响应此调用返回的密钥数量，示例项目会做出以下决定之一：

- 如果密钥计数大于或等于 1，则项目将转换到 ProcessNOAAData 状态。此分布式地图状态包括一个名为的函数 TemperatureFunction，该Lambda函数用于查找每月平均温度最高的气象站。此函数返回一个以 `year-month` 为密钥的目录和一个包含将气象站相关信息作为值的目录。
- 如果返回的密钥计数不超过 1，则分布式 S3 副本 NOA 数据状态将列出公共存储桶中的所有对象，noaa-gsod-pds并以 100 个为一批迭代地将各个对象复制到您账户中的另一个存储桶。内联 Map 会执行对象的迭代复制。

复制完所有对象后，项目将转换到 ProcessNOAAData 状态，处理天气数据。

示例项目最终转换为 reduc Lambda er 函数，该函数对TemperatureFunction函数返回的结果进行最终聚合，并将结果写入Amazon DynamoDB表中。

使用分布式 Map，一次最多可运行 1 万个并行子工作流。在此示例项目中，ProcessNOAAData 分布式 Map 的最大并发量设置为 3000，这就限制了它只能并行执行 3000 个子工作流。

此示例项目创建状态机、支持 AWS 资源并配置相关的 IAM 权限。探索此示例项目，了解如何使用分布式 Map 编排大规模并行工作负载，或将其作为您自己项目的起点。

> **Important**
>
> 此示例项目仅在美国东部（弗吉尼亚州北部）区域可用。

AWS CloudFormation 模板和其他资源

您可以使用 CloudFormation 模板来部署此示例项目。此模板在您的中创建了以下资源 AWS 账户：

- 一个 Step Functions 状态机。
- 状态机的执行角色。此角色授予您的状态机访问其他 AWS 服务 资源所需的权限，例如 Lambda 函数的 Invoke 操作。
- 一个名为 `NOAADataBucket` 的 Amazon S3 存储桶。此存储桶包含带有天气数据的 CSV 文件。
- 一个名为 `ReducerFunction` 的 Lambda 函数，用于对天气数据进行最终聚合，并将结果写入 Amazon DynamoDB 表。
- reducer Lambda 函数的执行角色。此角色授予函数访问其他角色的权限 AWS 服务。
- 一个名为 `ResultsBucket` 的 Amazon S3 输出桶，用于存储天气分析结果。
- 一个名为 `ResultsDynamoDBTable` 的 DynamoDB 表，其中包含 `ReducerFunction` 返回的结果。
- 一个名为 `TemperatureFunction` 的 Lambda 函数，用于查找最高月平均气温。
- Lambda 函数的执行角色。此角色授予函数访问其他角色的权限 AWS 服务。
- 存储与状态机执行历史相关的信息的 CloudWatch 日志组。

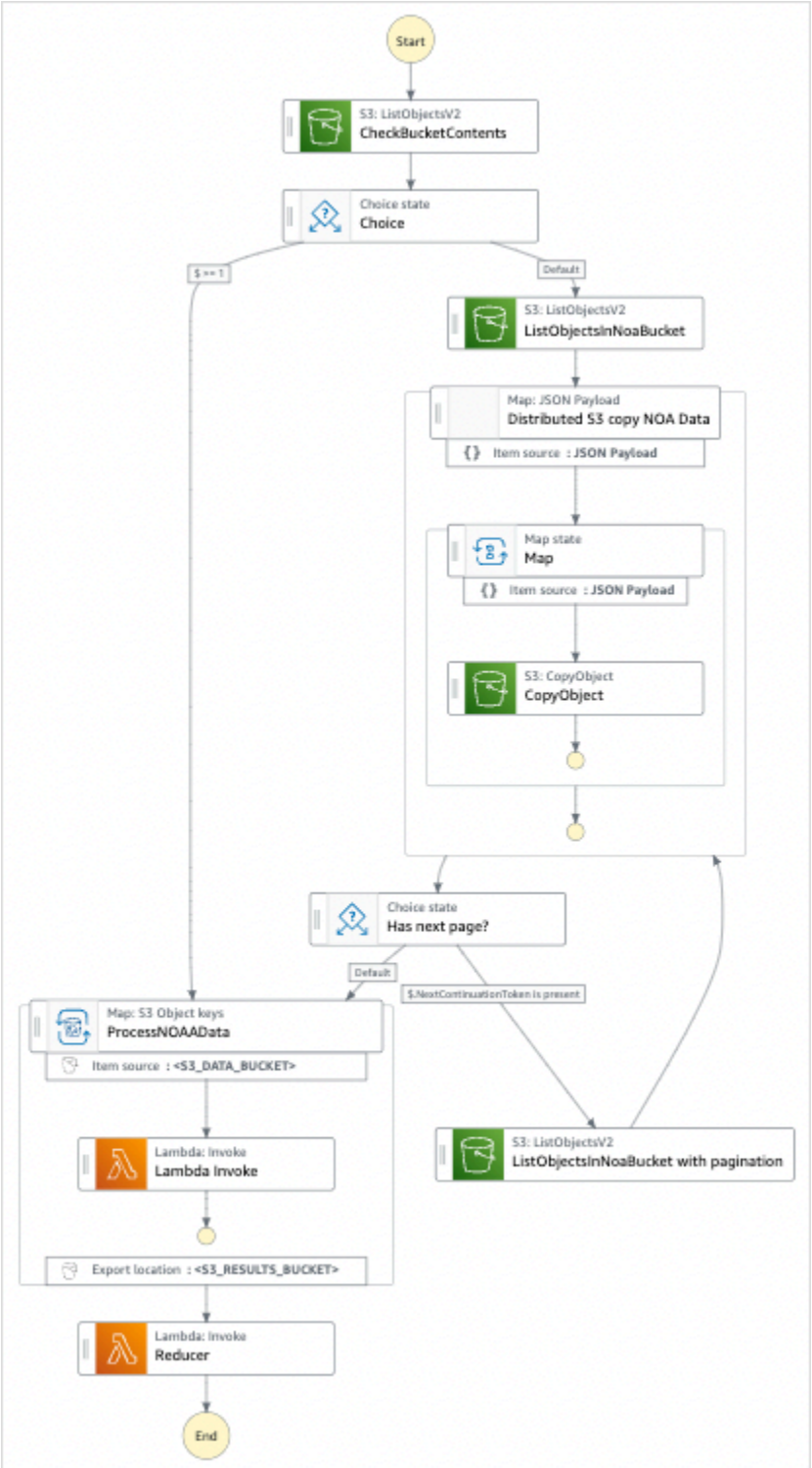
> **Important**
>
> 标准费用适用于每项服务。

第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Distributed Map to process files in S3`**，然后从返回的搜索结果中选择分布式 Map 处理 S3 中的文件。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   有关将为此示例项目创建的资源信息，请参阅 AWS CloudFormation 模板和其他资源。

   下图显示了分布式 Map 处理 S3 中的文件示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language (ASL) 定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了 “运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

配置并部署完所有资源后，您可以运行状态机。

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

a. （可选）以 JSON 格式键入输入值以运行您的示例项目。

如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

b. 选择启动执行。

c. （可选）Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

执行完成后，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。

- 有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。
- 有关在控制台中查看分布式 Map 状态 执行的更多信息，请参阅检查 Map Run。

d. （可选）查看导出到 Amazon S3 存储桶的执行结果。这些结果包括数据，例如执行输入和输出、ARN 和执行状态。有关更多信息，请参阅 ResultWriter。

训练机器学习模型

此示例项目演示了如何使用 SageMaker 和 AWS Step Functions 训练机器学习模型以及如何批量转换测试数据集。

在此项目中，Step Functions 使用 Lambda 函数通过测试数据集为 Amazon S3 存储桶添加种子。然后，它使用SageMaker 服务集成训练机器学习模型并执行批量转换。

有关 SageMaker 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- SageMaker 使用 Step Functions 进行管理

> **Note**
>
> 此示例项目可能会产生费用。

对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅SageMaker 定价。

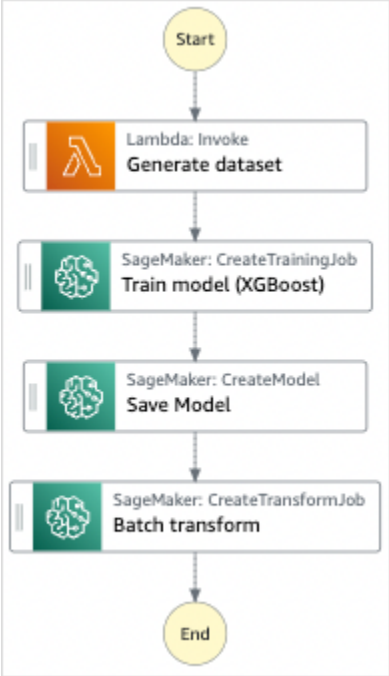
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Train a machine learning model`**，然后从返回的搜索结果中选择训练机器学习模型。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 AWS Lambda 函数
   - 一个 Amazon Simple Storage Service (Amazon S3) 存储桶
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了训练机器学习模型示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

     > **Tip**
     >
     > 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

      > **Note**
      >
      > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

   3. 选择启动执行。

4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 AWS Lambda 通过将参数直接传递给这些资源进行 SageMaker 集成，并使用 Amazon S3 存储桶作为训练数据源和输出。

浏览此示例状态机，了解 Step Functions 如何控制 Lambda 和。 SageMaker

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

以下策略允许 Lambda 函数使用示例数据为 Amazon S3 存储桶添加种子。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::trainandbatchtransform-s3bucket-1jn1le6gadwfz/*",
            "Effect": "Allow"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

调整机器学习模型

此示例项目演示了 SageMaker 如何使用调整机器学习模型的超参数和批量转换测试数据集。

在此项目中，Step Functions 使用 Lambda 函数通过测试数据集为 Amazon S3 存储桶添加种子。然后，它使用SageMaker服务集成创建超参数调整作业。然后，它使用 Lambda 函数提取数据路径，保存调整模型，提取模型名称，然后运行批处理转换作业以在中执行推理。 SageMaker

有关 SageMaker 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- SageMaker 使用 Step Functions 进行管理

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅SageMaker定价。

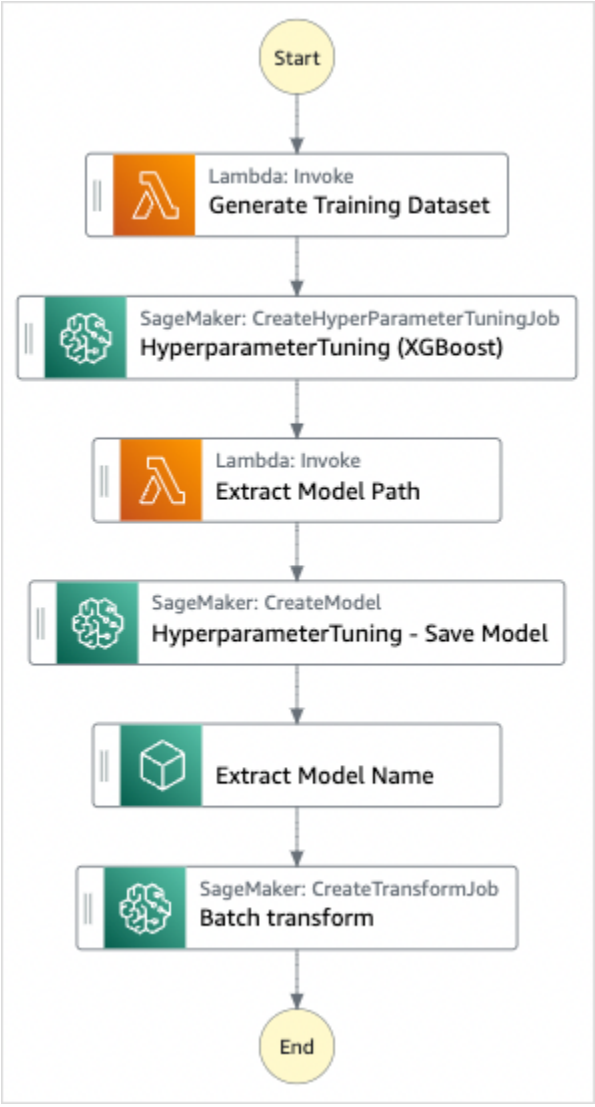
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Tune a machine learning model`**，然后从返回的搜索结果中选择调整机器学习模型。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 三种 AWS Lambda 功能
   - 一个 Amazon Simple Storage Service (Amazon S3) 存储桶
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了调整机器学习模型示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 AWS Lambda 通过将参数直接传递给这些资源进行 SageMaker 集成，并使用 Amazon S3 存储桶作为训练数据源和输出。

浏览此示例状态机，了解 Step Functions 如何控制 Lambda 和。 SageMaker

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
    "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
        }
    }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

以下 IAM 策略附加到状态机，允许状态机执行访问必要 SageMaker的 Lambda 和 Amazon S3 资源。

```
{
    "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"
        }
    ]
```

```
}
```

以下 IAM 策略在 HyperparameterTuning 状态的 TrainingJobDefinition 和 HyperparameterTuning 字段中引用。

```
{
    "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"
        }
```

```
    ]
}
```

以下 IAM 策略允许 Lambda 函数使用示例数据为 Amazon S3 存储桶添加种子。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/*",
            "Effect": "Allow"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

处理来自 Amazon SQS（快速工作流）的大批量消息

此示例项目演示了如何使用 AWS Step Functions Express Workflow 来处理来自高容量事件源（例如亚马逊简单队列服务 (Amazon SQS)）的消息或数据。由于快速工作流可以非常高的速率启动，因此它们非常适合大批量事件处理或流数据工作负载。

以下是从事件源执行状态机的两种常用方法：

- 配置 Amazon CloudWatch Events 规则，以便在事件源发出事件时启动状态机执行。有关更多信息，请参阅创建在 CloudWatch 事件上触发的事件规则。
- 将事件源映射到 Lambda 函数，并编写函数代码来执行状态机。每次事件源发出事件时，都会调用该 AWS Lambda 函数，进而启动状态机执行。有关更多信息，请参阅结合使用 AWS Lambda 与 Amazon SQS。

此示例项目使用第二种方法在 Amazon SQS 队列每次发送消息时启动执行。您可以使用类似的配置，触发来自其他事件源的快速工作流执行，比如 Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB 和 Amazon Kinesis。

有关快速工作流和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 标准和快速工作流
- 与其他服务 AWS Step Functions 一起使用
- 配额

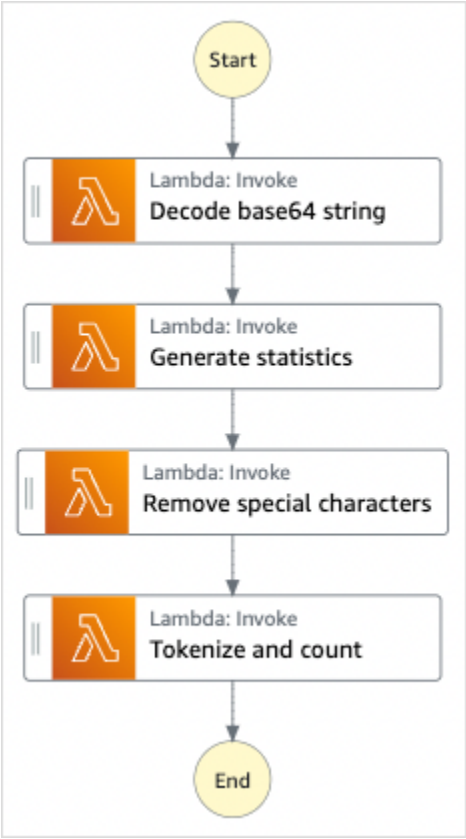
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Process high-volume messages from SQS`**，然后从返回的搜索结果中选择处理来自 SQS 的大批量消息。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 四个 Lambda 函数
   - 一个 Amazon SQS 队列
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了处理来自 SQS 的大批量消息示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language (ASL) 定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：触发状态机执行

1. 打开 Amazon SQS 控制台。
2. 选择示例项目创建的队列。

   该名称将类似于 Example-SQSQueue-wJalrXUtnFEMI。
3. 在 Queue Actions (队列操作) 列表中，选择 Send a Message (发送消息)。
4. 使用复制按钮复制以下消息，然后在 Send a Message (发送邮件) 窗口中输入该消息，然后选择 Send Message (发送邮件)。

> **Note**
>
> 在此示例消息中，已使用换行符对 `input:` 进行格式化以适应页面。使用复制按钮或以其他方式确保它作为一行输入，没有中断。

```
{
      "input":
 "QW5kIGxpa2UgdGhlIGJhc2VsZXNzIGZhYnJpYyBvZiB0aGlzIHZpc2lvbiwgVGhlIGNsb3VkLWNhcHBlZCB0b3dlc
```

```
  91cyBwYWxhY2VzLCBUaGUgc29sZW1uIHRlbXBsZXMsIHRoZSBncmVhdCBnbG9iZSBpdHNlbGbigJQgWWVhLCBhbGw
  ZXJpdOKAlHNoYWxsIGRpc3NvbHZlLCBBbmQgbGlrZSB0aGlzIGluc3Vic3RhbnRpYWwgcGFnZWFudCBmYWRlZCwgT
  FjayBiZWhpbmQuIFdlIGFyZSBzdWNoIHN0dWZmIEFzIGRyZWFtcyBhcmUgbWFkZSBvbiwgYW5kIG91ciBsaXR0bGU
  ZGVkIHdpdGggYSBzbGVlcC4gU2lyLCBJIGFtIHZleGVkLiBCZWFyIHdpdGggbXkgd2Vha25lc3MuIE15IG9sZCBic
  xlZC4gQmUgbm90IGRpc3R1cmJlZCB3aXRoIG15IGluZmlybWl0eS4gSWYgeW91IGJlIHBsZWFzZWQsIHJldGlyZSB
  QW5kIHRoZXJlIHJlcG9zZS4gQSB0dXJuIG9yIHR3byBJ4oCZbGwgd2FsayBUbyBzdGlsbCBteSBiZWF0aW5nIG1pb
}
```

5. 选择关闭。
6. 打开 Step Functions 控制台。
7. 前往您的 Amazon CloudWatch 日志组并检查日志。日志组的名称将类似于示例 ExpressLogGroup-wj alrxutnFemi。

示例 Lambda 函数代码

以下是 Lambda 函数代码，它显示了启动的 Lambda 函数如何使用软件开发工具包启动状态机执行。AWS

```
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
    )
```

示例状态机代码

此示例项目中的快速工作流包含一组用于文本处理的 Lambda 函数。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

以下策略可确保有足够的 CloudWatch 日志权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

选择性检查点示例（快速工作流）

此示例项目演示了如何通过运行执行选择性检查点的模拟电子商务工作流来组合标准和快速工作流。部署此示例项目将创建标准工作流状态机、嵌套的快速工作流状态机、 AWS Lambda 函数、Amazon Simple Queue Service (Amazon SQS) 队列和 Amazon Simple Notification Service (Amazon SNS) 主题。

有关快速工作流、嵌套工作流和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 标准和快速工作流
- 从 Task 状态启动工作流执行
- 与其他服务 AWS Step Functions 一起使用

第 1 步：创建状态机并预置资源

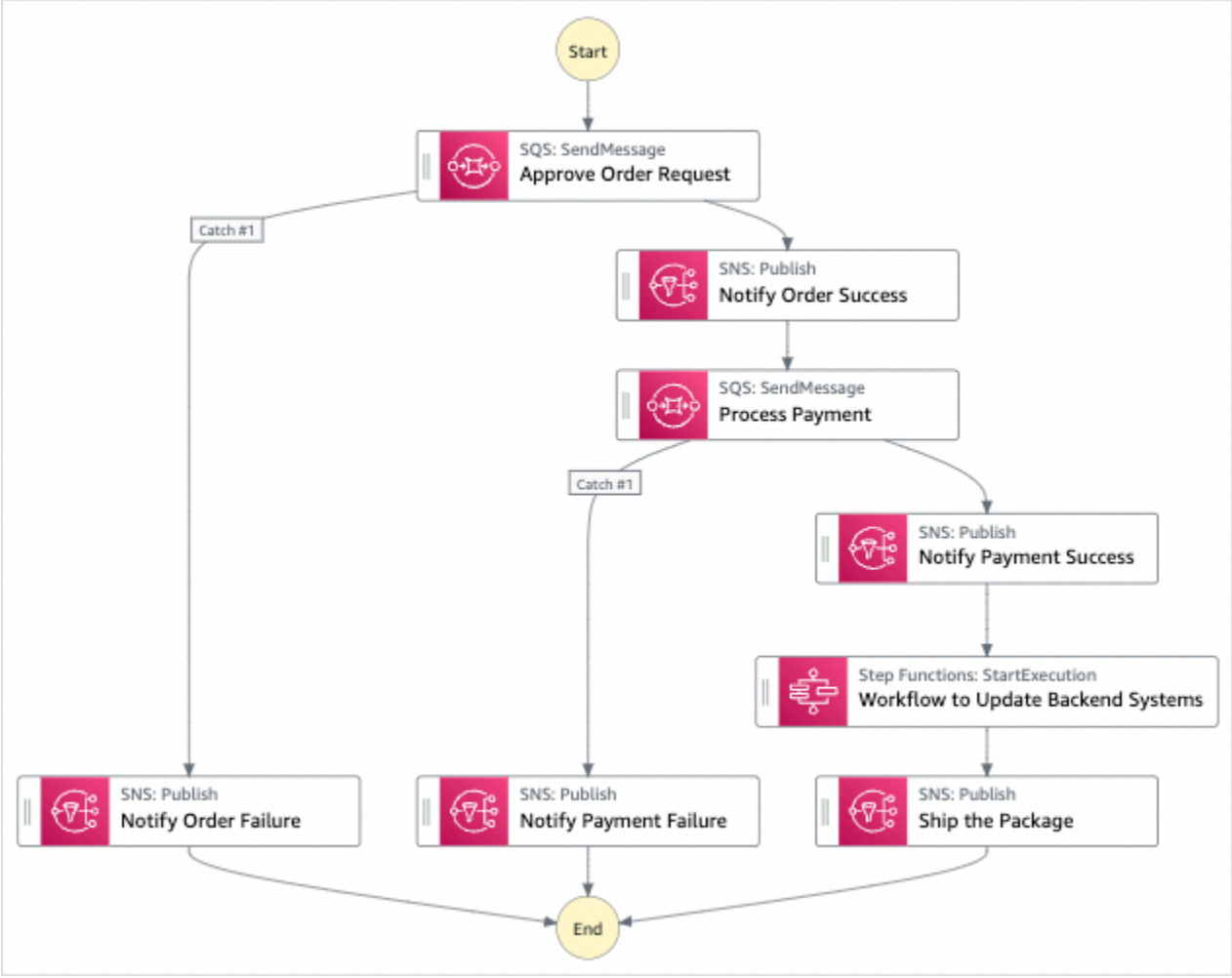
1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Selective checkpointing example`**，然后从返回的搜索结果中选择选择性检查点示例。
3. 选择下一步以继续。

4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 AWS Lambda 函数
   - 一个 Amazon SQS 队列
   - 一个 Amazon SNS 主题
   - 标准类型的 AWS Step Functions 状态机
   - 一个快速类型的 Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了选择性检查点示例示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

     > **Tip**
     >
     > 要查看示例项目的状态机定义，请选择代码。

     准备就绪后，选择部署并运行以部署示例项目并创建资源。

     创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

     创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

     > **Important**
     >
     > CloudFormation 模板中使用的每项服务都可能收取标准费用。

部署示例项目的资源后，执行以下操作。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

      > **Note**
      >
      > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

      要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。
4. 转到您的CloudWatch 日志日志组并检查日志。日志组的名称将类似于示例 ExpressLogGroup-wjalrxutnFemi。

父级（标准工作流）的示例状态机代码

此示例项目中的状态机与 Amazon SQS、Amazon SNS 和 Step Functions 快速工作流集成。

浏览此示例状态机，了解 Step Functions 如何处理来自 Amazon SQS 和 Amazon SNS 的输入，然后使用嵌套的快速工作流状态机更新后端系统。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅[与其他服务 AWS Step Functions 一起使用]。

```
{
    "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
        }
    }
}
```

父状态机的示例 IAM 角色

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

Amazon SNS 策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
```

```
            "Resource": "arn:aws:sns:us-east-1:123456789012:Checkpoint-SNSTopic-
wJalrXUtnFEMI",
            "Effect": "Allow"
        }
    ]
}
```

Amazon SQS 策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:us-east-1:123456789012:Checkpoint-SQSQueue-
je7MtGbClwBF",
            "Effect": "Allow"
        }
    ]
}
```

状态执行策略：

```
{
    "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"
        }
    ]
}
```

嵌套状态机（快速工作流）的示例状态机代码

此示例项目中的状态机在父状态机调用时更新后端信息。

浏览此示例状态机，了解 Step Functions 如何更新模拟电子商务后端系统的不同组件。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅[与其他服务 AWS Step Functions 一起使用](#)。


```
{
  "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
    }
  }
}
```

子状态的示例 IAM 角色

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:Example-
UpdateDatabaseLambdaFunction-wJalrXUtnFEMI"
            ],
            "Effect": "Allow"
        }
    ]
}
```

以下策略可确保有足够的 CloudWatch 日志权限。

```
{
    "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"
        }
```

```
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

构建 AWS CodeBuild 项目（CodeBuild，亚马逊 SNS）

此示例项目演示 AWS Step Functions 如何使用构建 AWS CodeBuild 项目、运行测试以及发送 Amazon SNS 通知。

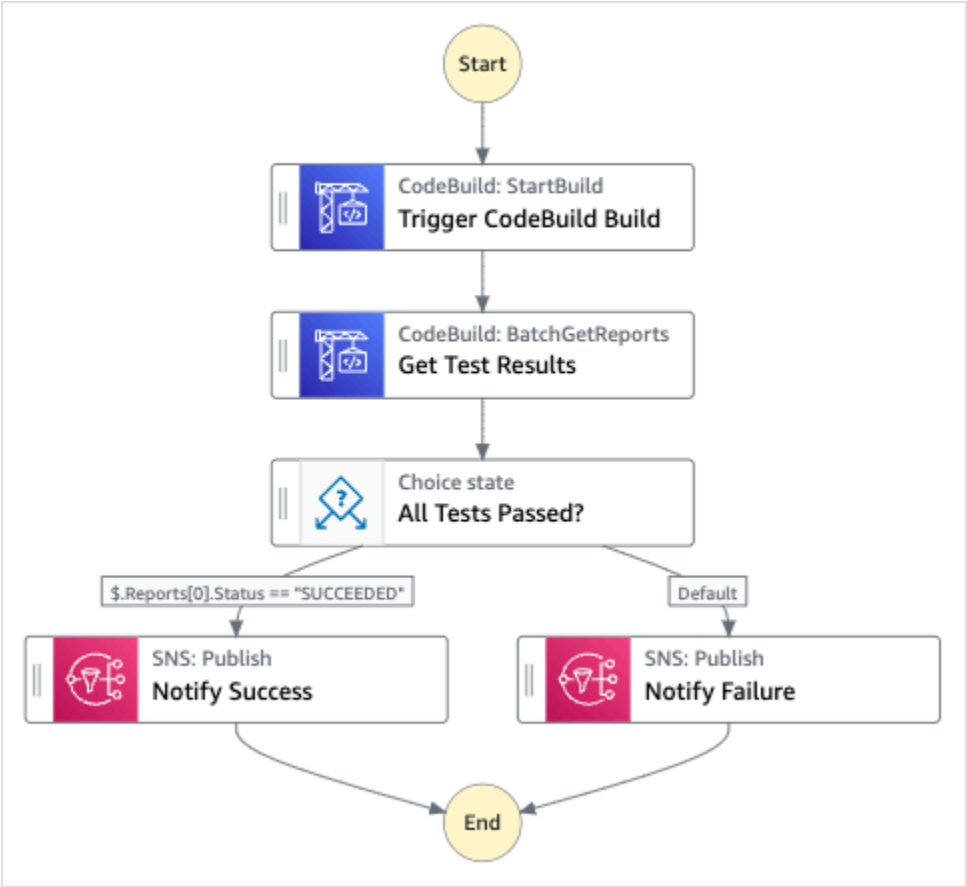
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. **`Start a CodeBuild build`**在搜索框中键入，然后从返回的搜索结果中选择 “启动 CodeBuild 构建”。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 AWS CodeBuild 构建
   - 一个 Amazon SNS 主题
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了 “启动 CodeBuild 构建” 示例项目的工作流程图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机与 CodeBuild Amazon SNS 集成。

浏览此示例状态机，了解 Step Functions 如何使用状态机构建 CodeBuild 项目，然后发送一个 Amazon SNS 主题，其中包含有关任务成功还是失败的消息。

有关 Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

预处理数据并训练机器学习模型

此示例项目演示了如何使用 SageMaker 和 AWS Step Functions 预处理数据，以及如何训练机器学习模型。

在此项目中，Step Functions 使用 Lambda 函数通过测试数据集为 Amazon S3 存储桶添加种子，并通过 Python 脚本进行数据处理。然后，它使用SageMaker 服务集成训练机器学习模型并执行批量转换。

有关 SageMaker 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- SageMaker 使用 Step Functions 进行管理

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅SageMaker 定价。

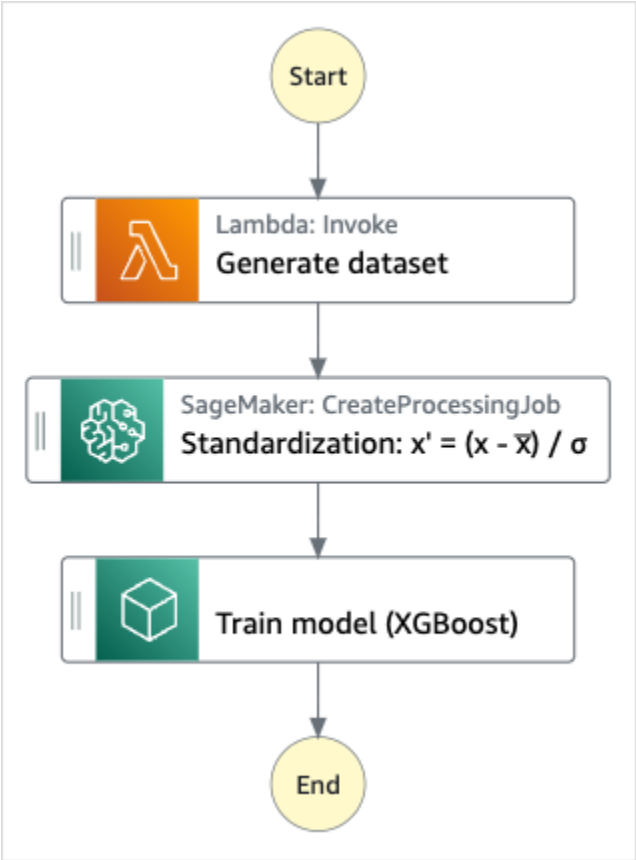
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Preprocess data and train a machine learning model`**，然后从返回的搜索结果中选择预处理数据并训练机器学习模型。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 AWS Lambda 函数
   - 一个 Amazon S3 存储桶
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

下图显示了预处理数据并训练机器学习模型示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 AWS Lambda 通过将参数直接传递给这些资源进行 SageMaker 集成，并使用 Amazon S3 存储桶作为训练数据源和输出。

浏览此示例状态机，了解 Step Functions 如何控制 Lambda 和。 SageMaker

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

以下策略允许 Lambda 函数使用示例数据为 Amazon S3 存储桶添加种子。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::featuretransform-
bucketforcodeanddata-1jn1le6gadwfz/*",
            "Effect": "Allow"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

Lambda 编排示例

此示例项目演示了如何在 Step Functions 状态机中集成 AWS Lambda 函数。

在这个项目中，Step Functions 使用 Lambda 函数来检查股票价格并确定买入或卖出交易建议。然后向用户提供此建议，用户可以选择是买入还是卖出股票。交易结果将使用 SNS 主题返回。

有关 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 适用于以下内容的 IAM 策略：
  - 适用的 IAM 政策 AWS Lambda
  - 亚马逊 SQS 的 IAM 政策

- 亚马逊 SNS 的 IAM 政策

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅定价。

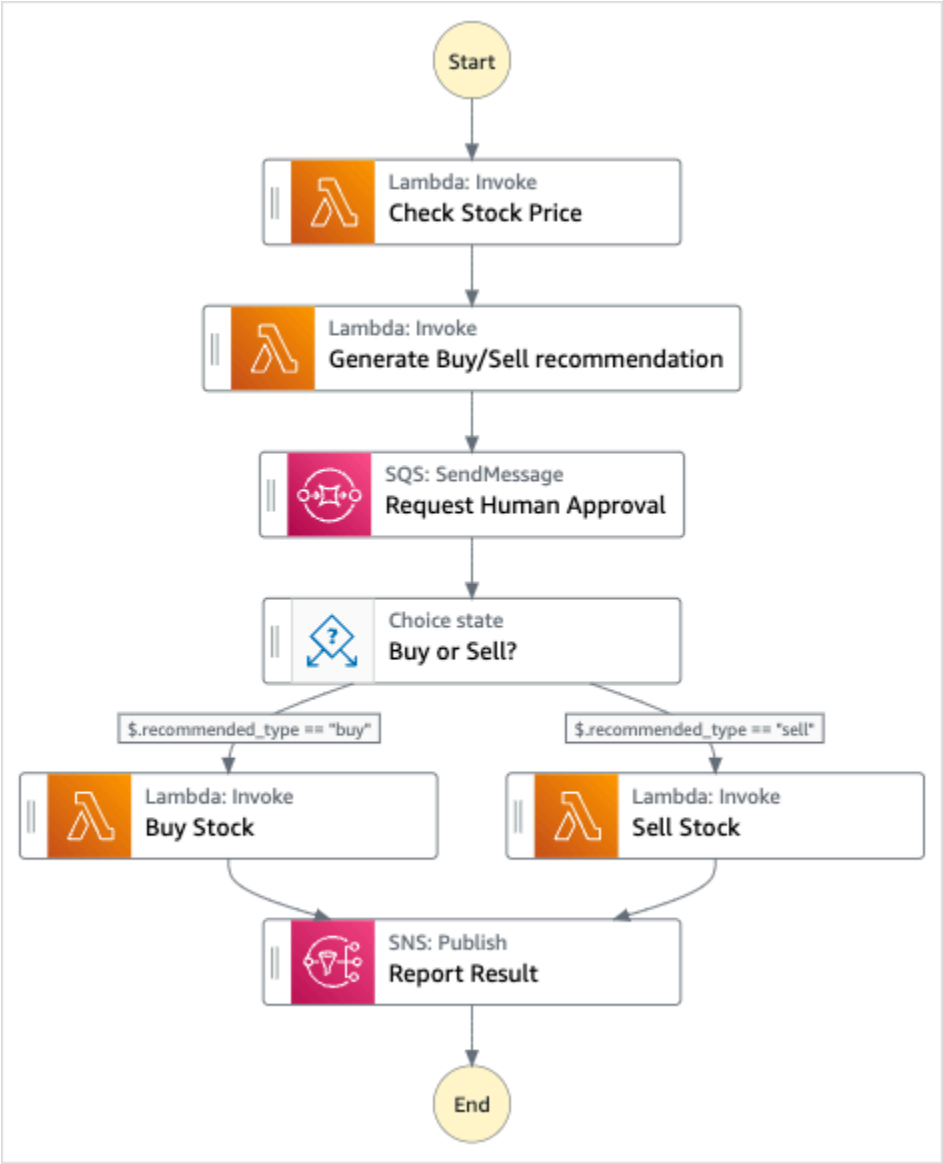
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Orchestrate Lambda functions`**，然后从返回的搜索结果中选择编排 Lambda 函数。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 AWS 账户，或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 五个 Lambda 函数
   - 一个 Amazon Simple Queue Service 队列
   - 一个 Amazon Simple Notification Service 主题
   - 一个 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了编排 Lambda 函数示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

配置并部署完所有资源后，将显示启动执行对话框。

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

关于状态机及其执行

此示例项目中的状态机 AWS Lambda 通过将参数直接传递给这些资源来集成，使用 Amazon SQS 队列管理人工批准请求，并使用 Amazon SNS 主题返回查询结果。

Step Functions 执行接收 JSON 文本作为输入并将该输入传递到工作流中的第一个状态。各个状态将接收输入形式的 JSON 数据并通常将 JSON 数据以输出形式传递到下一个状态。在本示例项目中，每个步骤的输出都作为输入传递给工作流中的下一个步骤。例如，生成买入/卖出建议步骤接收检查股价步骤的输出作为输入。此外，生成买入/卖出建议步骤的输出将作为输入传递到下一个步骤请求人工审批，该步骤模拟人工审批步骤。

> **Note**
>
> 要查看步骤返回的输出和传递给步骤的输入，请打开工作流执行的执行详细信息页。在步骤详细信息 部分，在视图模式 下查看选择的每个步骤的输入和输出。

要实现人工审批步骤，通常需要暂停工作流的执行，直到任务令牌返回。在此示例项目中，一条消息被传递到 Amazon SQS 队列，该队列被用作定义用于处理回调功能的 Lambda 函数的触发器。该消息包含任务令牌和上一步返回的输出。Lambda 函数使用消息的有效载荷被调用。工作流执行将暂停，直到它通过 SendTaskSuccess API 调用收到返回的任务令牌。有关任务令牌的更多信息，请参阅等待具有任务令牌的回调。

StepFunctionsSample-HelloLambda-ApproveSqsLambda 函数的以下代码显示了如何定义该函数，以便通过 Step Functions 状态机自动批准 Amazon SQS 队列提交的任何任务。

处理回调功能并返回任务令牌的 Lambda 函数代码示例

```
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);
      });
  }
};
```

浏览此示例状态机以了解 Step Functions 如何控制 Lambda 和 Amazon SQS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅[与其他服务 AWS Step Functions 一起使用]。

```
{
    "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
        }
    }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

启动 Athena 查询

本示例项目基于标准工作流，演示了如何使用 Step Functions 和 Amazon Athena 启动 Athena 查询并发送包含查询结果的通知。

在这个项目中，Step Functions 使用 Lambda 函数和 AWS Glue 爬虫来生成一组示例数据。然后，它使用 Athena 服务集成执行查询，并使用 SNS 主题返回结果。

有关 Athena 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 使用 Step Functions 调用 Athena

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅 Athena 定价。

第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Start an Athena query`**，然后从返回的搜索结果中选择启动 Athena 查询。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon Athena 查询
   - 一个 AWS Glue 爬网程序
   - 一个 Amazon SNS 主题

- 一个 AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了启动 Athena 查询示例项目的工作流程图：

5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源与 Athena AWS Lambda 集成，并使用 SNS 主题返回查询结果。

浏览此示例状态机以了解 Step Functions 如何控制 Lambda 和 Athena。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅[集成服务的 IAM 策略](#)。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
```

```
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

执行多个查询（Amazon Athena、Amazon SNS）

本示例项目演示了如何连续和并行运行 Athena 查询，处理错误，然后根据查询成功或失败发送 Amazon SNS 通知。

在本项目中，Step Functions 使用状态机同步运行 Athena 查询。返回查询结果后，进入并行状态，并行执行两个 Athena 查询。然后它等待作业成功或失败，并发送一个 Amazon SNS 主题，其中包含有关作业是成功还是失败的消息。

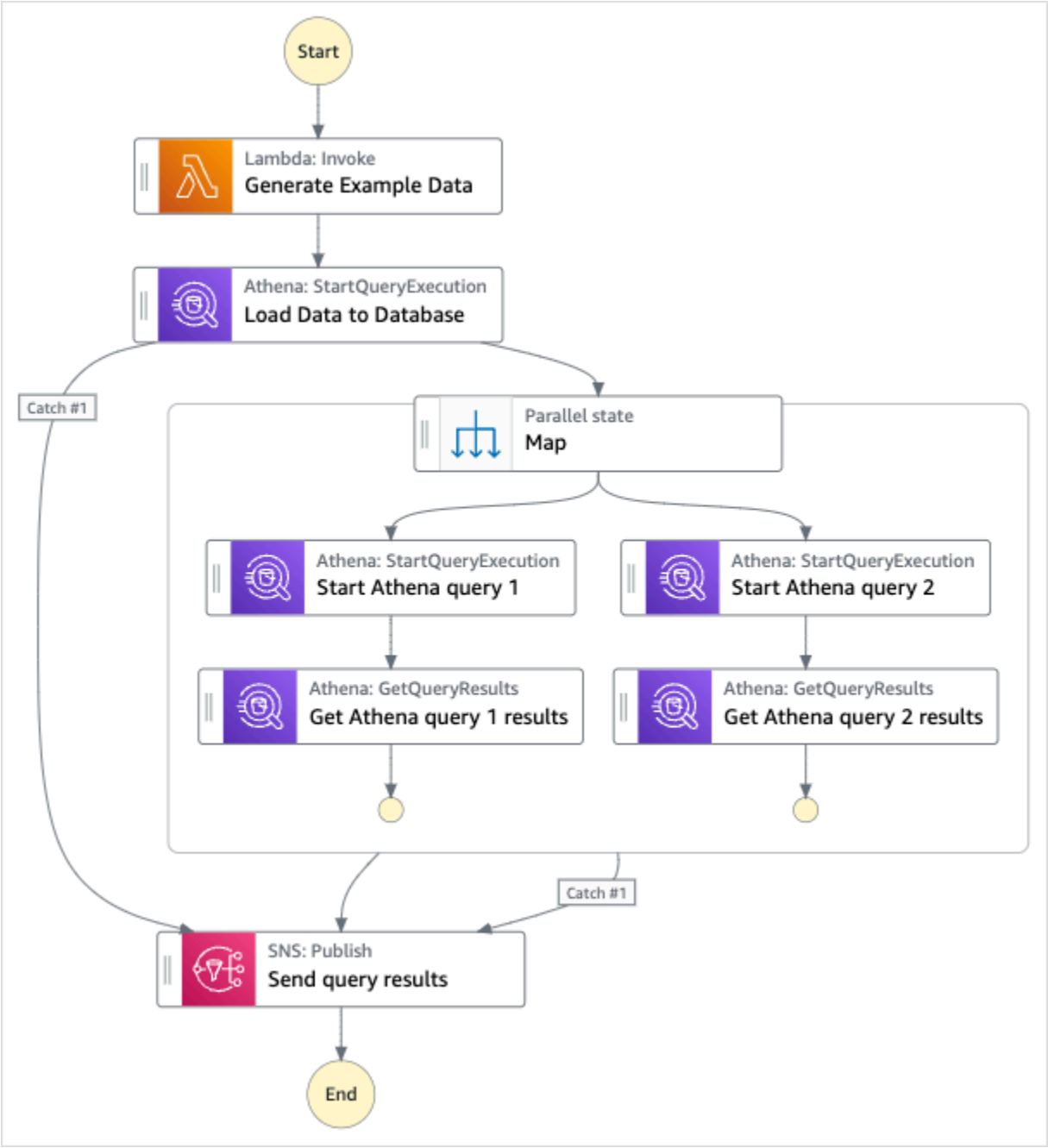
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Execute multiple queries`**，然后从返回的搜索结果中选择执行多个查询。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - Amazon Athena 查询
   - 一个 Amazon SNS 主题
   - 一个 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了执行多个查询示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控

制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源来与 Amazon Athena 和 Amazon SNS 集成。

浏览此示例状态机，了解 Step Functions 如何通过连接到 `Resource` 字段中的 Amazon 资源名称 (ARN)，以及向服务 API 传递 `Parameters` 来控制 Amazon Athena 和 Amazon SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
    "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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"
            ]
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

查询大型数据集（亚马逊 Athena、亚马逊 S3、亚马逊 SN AWS Glue S）

此示例项目演示了如何在 Amazon S3 中提取大型数据集并通过 AWS Glue Crawlers 对其进行分区，然后对该分区执行 Amazon Athena 查询。

在此项目中，Step Functions 状态机调用一个 AWS Glue 抓取程序，该爬虫在 Amazon S3 中对大型数据集进行分区。 AWS Glue 抓取器返回成功消息后，工作流程将对该分区执行 Athena 查询。成功执行查询后，将向 Amazon SNS 主题发送到 Amazon SNS 通知。

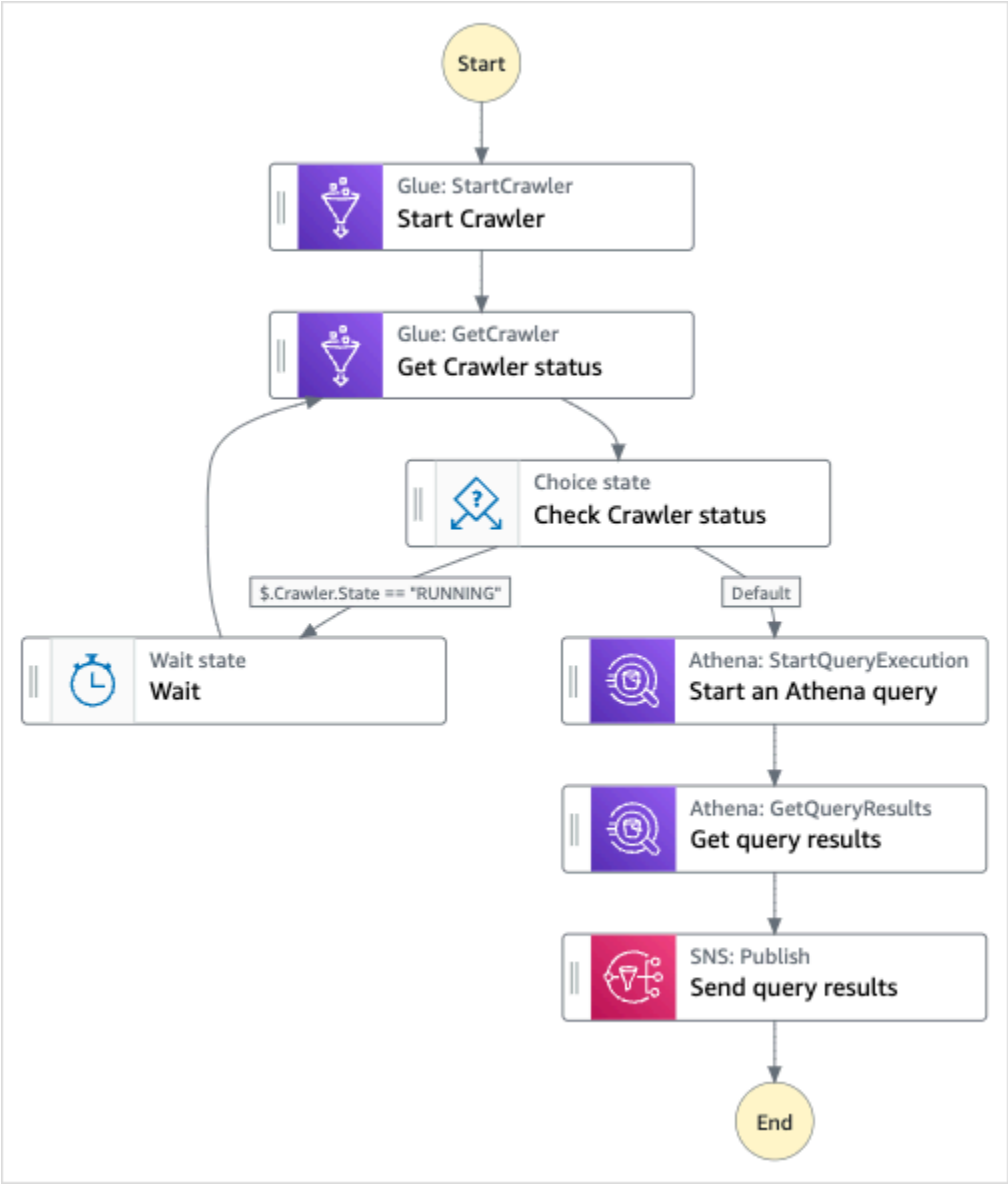
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Query large datasets`**，然后从返回的搜索结果中选择查询大型数据集。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon S3 存储桶
   - 一个 AWS Glue 爬网程序
   - 一个 Amazon SNS 主题
   - 一个 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了查询大型数据集示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择一个仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给这些资源来与 Amazon S3 AWS Glue、、Amazon Athena 和 Amazon SNS 集成。

浏览此示例状态机，了解 Step Functions 如何通过连接到字段中的`Resource`亚马逊资源名称 (ARN) 并传递到服务 API 来控制亚马逊 S3、、Amazon Athena 和 Amazon SNS。 AWS Glue`Parameters`

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
    "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 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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"
            ]
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

保持数据最新（亚马逊 Athena、Amazon S3、）AWS Glue

此示例项目演示如何使用 Catalo AWS Glue g 查询目标表以获取当前数据，然后使用 Amazon Athena 使用来自其他来源的新数据对其进行更新。

在此项目中，Step Functions 状态机调用 C AWS Glue atalog 来验证 Amazon S3 存储桶中是否存在目标表。如果找不到表，它将创建一个新表。然后，Step Functions 将运行 Athena 查询，从不同的数据来源向目标表添加行：首先查询目标表以获取最近日期，然后查询源表以获取更多最新数据，并将其插入到目标表中。

第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Keep data up to date`**，然后从返回的搜索结果中选择使数据保持最新状态。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon S3 存储桶

- Amazon Athena 查询
- 一个 AWS Glue Data Catalog 调用
- 一个 AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了使数据保持最新状态示例项目的工作流图：

5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

  > **Important**
  >
  > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon S3 和 Amazon Athena 资源来集成。 AWS Glue

浏览此示例状态机，了解 Step Functions 如何通过连接到字段中的`Resource`亚马逊资源名称 (ARN) 并传递到服务 API 来`Parameters`控制亚马逊 S3 和亚马逊 Athena。 AWS Glue

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
    "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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": [
                "*"
            ]
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

管理 Amazon EKS 集群

此示例项目演示了如何使用 Step Functions 和 Amazon Elastic Kubernetes Service 创建带有节点组的 Amazon EKS 集群，在 Amazon EKS 上运行作业，然后检查输出。完成后，它会删除节点组和 Amazon EKS 集群。

有关 Step Functions 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 使用 Step Functions 调用 Amazon EKS

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅 Amazon EKS 定价。

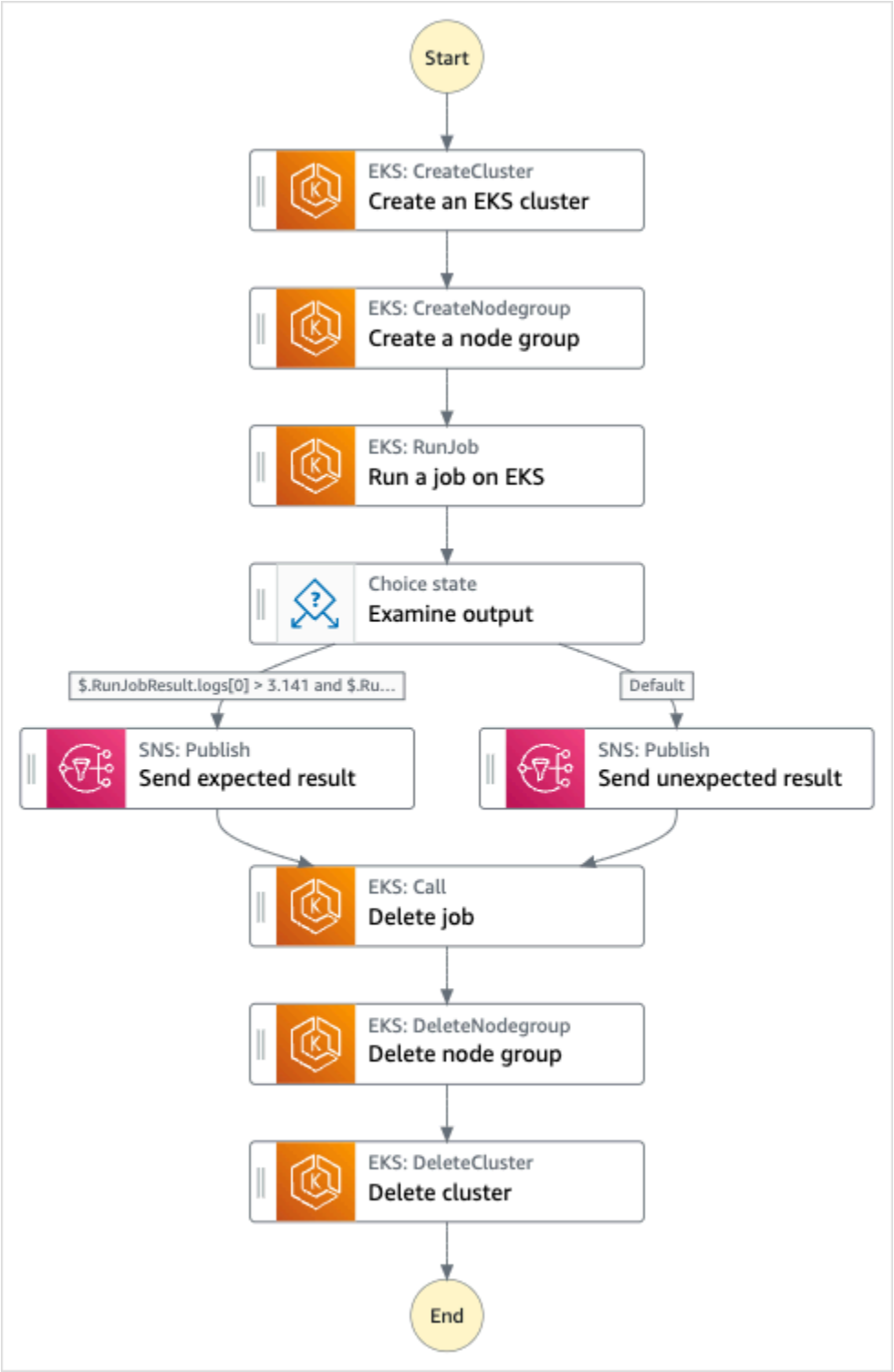
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Manage an EKS cluster`**，然后从返回的搜索结果中选择管理 EKS 集群。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon Elastic Kubernetes Service 集群
   - 一个 Amazon SNS 主题
   - 一个 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了管理 EKS 集群示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过创建 Amazon EKS 集群和节点组与 Amazon EKS 集成，并使用 SNS 主题返回结果。

浏览此示例状态机，了解 Step Functions 如何管理 Amazon EKS 集群和节点组。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
            ]
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

调用 API Gateway

此示例项目演示了如何使用 Step Functions 调用 API Gateway 并检查调用是否成功。

有关 API Gateway 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 使用 Step Functions 调用 API Gateway

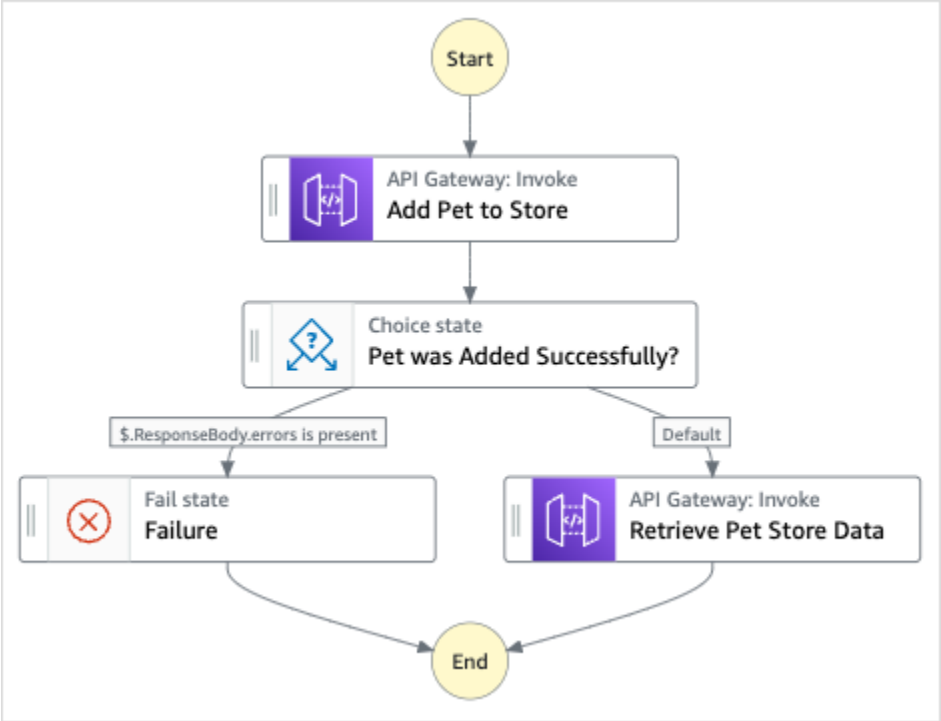
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **Make a call to API Gateway**，然后从返回的搜索结果中选择调用 API Gateway。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

- 一个 Amazon API Gateway REST API，被状态机调用。
- 一个 AWS Step Functions 状态机
- 相关 AWS Identity and Access Management (IAM) 角色

下图显示了调用 API Gateway 示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

- 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

  在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过调用 API Gateway REST API 并传递任何必要的参数与 API Gateway 集成。

浏览此示例状态机，了解 Step Functions 如何与 API Gateway 交互。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
```

```
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用 API Gateway 集成调用在 Fargate 上运行的微服务

此示例项目演示了如何使用 Step Functions 调用 API Gateway AWS Fargate，以便与开启的服务进行交互，并检查调用是否成功。

有关 API Gateway 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 使用 Step Functions 调用 API Gateway

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅定价。

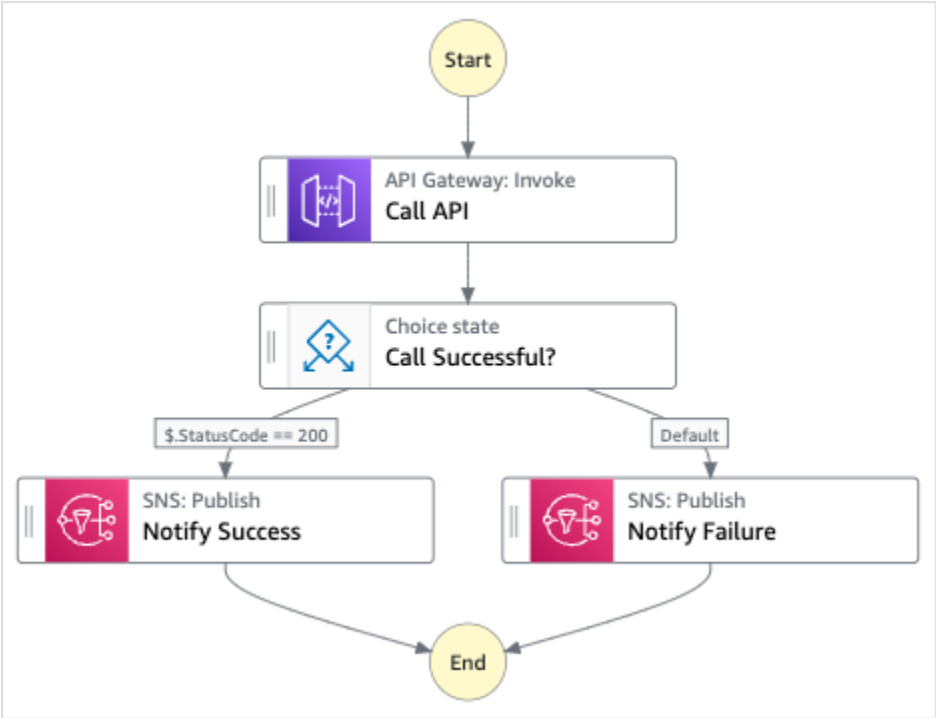
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Call a microservice with API Gateway`**，然后从返回的搜索结果中选择使用 API Gateway 调用微服务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon API Gateway HTTP API，被状态机调用。
   - 一个 Amazon API Gateway Amazon VPC 链接。
   - Amazon Virtual Private Cloud。
   - Application Load Balancer。
   - 一个 Fargate 集群。
   - 一个 Amazon SNS 主题
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色
   - 要使这些资源协同工作，还需要多项其他服务。

   下图显示了使用 API Gateway 调用微服务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过调用连接到 Fargate 上服务的 API Gateway HTTP API 与 API Gateway 集成。它托管在私有子网上，并通过私有应用程序负载均衡器进行访问。

浏览此示例状态机，了解 Step Functions 如何与 API Gateway 交互并返回结果。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

将自定义事件发送到 EventBridge

此示例项目演示如何使用 Step Functions 将自定义事件发送到与具有多个目标的规则匹配的事件总线（亚马逊 EventBridge AWS Lambda、、亚马逊简单通知服务、亚马逊简单队列服务）。

有关 Step Functions 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- EventBridge 使用 Step Functions 调用

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅EventBridge 定价。

第 1 步：创建状态机并预置资源

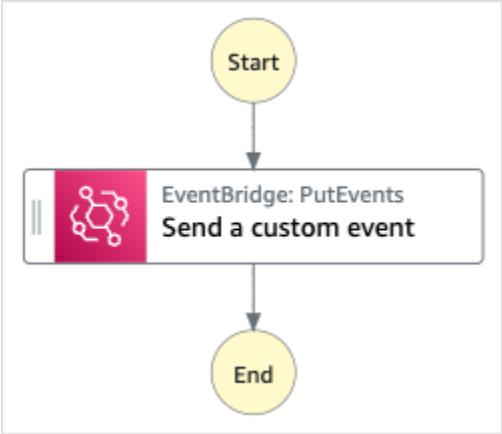
1. 打开 Step Functions 控制台，然后选择创建状态机。

2. 在搜索框中键入 **`Send a custom event to EventBridge`**，然后从返回的搜索结果中选择向 EventBridge 发送自定义事件。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon EventBridge 事件。
   - 一个 Amazon SNS 主题
   - 一个 Amazon SQS 队列
   - 一个 Lambda 函数
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了向 EventBridge 发送自定义事件示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控

制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 EventBridge 通过向事件总线发送自定义事件与集成。 EventBridge 发送到事件总线的事件与触发向亚马逊 SNS 主题和亚马逊 SQS 队列发送消息的 Lambda 函数的 EventBridge 规则相匹配。

浏览此示例状态机，了解 Step Functions 是如何管理 EventBridge的。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:1234567890:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

调用同步快速工作流

此示例项目演示了如何通过 Amazon API Gateway 调用同步快速工作流来管理员工数据库。

在这个项目中，Step Functions 使用 API Gateway 端点启动 Step Functions 同步快速工作流。然后，它们使用 DynamoDB 在员工数据库中搜索、添加和删除员工。

有关 Step Functions 同步快速工作流程的更多信息，请参阅同步和异步快速工作流。

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅 Step Functions 定价。

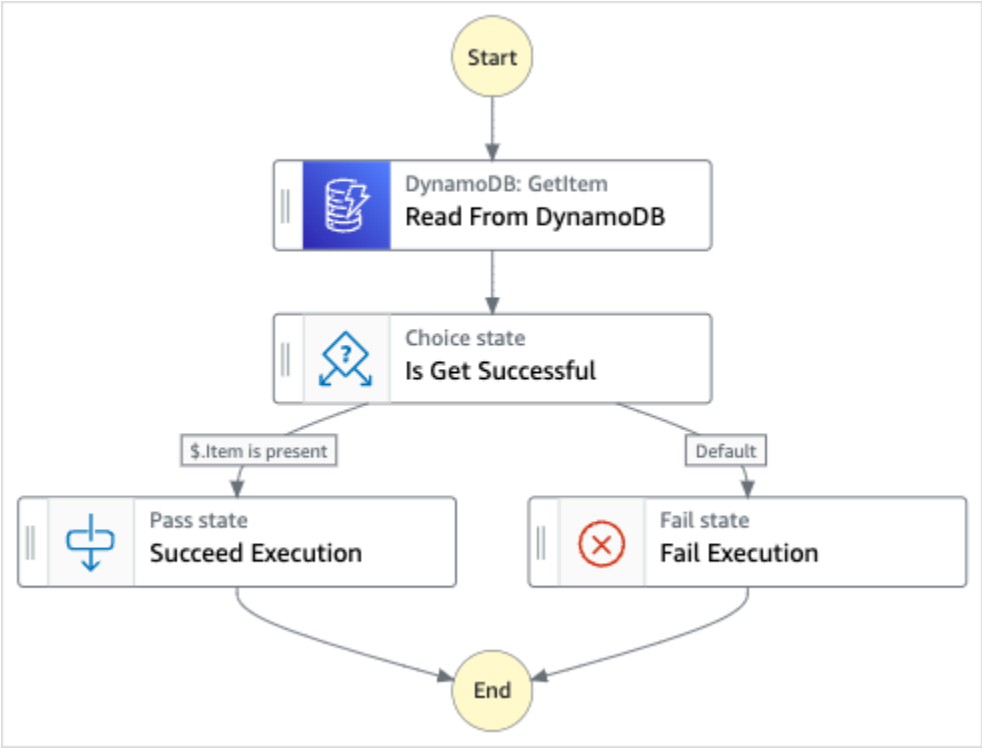
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Invoke Synchronous Express Workflows through API Gateway`**，然后从返回的搜索结果中选择通过 API Gateway 调用同步快速工作流。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon API Gateway HTTPS API，被状态机调用。
   - 一个 Amazon DynamoDB 表。
   - 三台 AWS Step Functions 状态机。
   - 相关 AWS Identity and Access Management (IAM) 角色。

   下图显示了通过 API Gateway 调用同步快速工作流示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

> **Note**
>
> 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过使用 API Gateway 调用同步快速工作流与 API Gateway 和 DynamoDB 集成，然后该工作流使用 DynamoDB 更新员工数据库或从员工数据库读取数据。

浏览此示例状态机，了解 Step Functions 如何从 DynamoDB 读取数据，检索员工信息。

要了解有关如何使用 API Gateway 调用 Step Functions 的更多信息，请参阅以下内容。

- 使用 Step Functions 调用 API Gateway
- API Gateway 开发人员指南中的如何调用私有 Gateway

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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"
    }
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
            ]
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用 Amazon Redshift 运行 ETL/ELT 工作流（Lambda、Amazon Redshift 数据 API）

此示例项目演示了如何使用 Step Functions 和 Amazon Redshift 数据 API 来运行 ETL/ELT 工作流，该工作流可将数据加载到 Amazon Redshift 数据仓库。

在此项目中，Step Functions 使用 AWS Lambda 函数和 Amazon Redshift 数据 API 来创建所需的数据库对象并生成一组示例数据，然后并行执行两个执行加载维度表的任务，然后执行一个事实表。两个维度加载任务成功结束后，Step Functions 将执行事实数据表的加载任务，运行验证作业，然后暂停 Amazon Redshift 集群。

> **Note**
>
> 您可以修改 ETL 逻辑来接收来自其他来源的数据，例如 Amazon S3，后者可以使用 COPY 命令将数据从 Amazon S3 复制到 Amazon Redshift 表。

有关 Amazon Redshift 和 Step Functions 服务集成的更多信息，请参阅以下内容：

- 与其他服务 AWS Step Functions 一起使用
- 使用 Amazon Redshift 数据 API
- Amazon Redshift 数据 API 服务
- 创建使用 Lambda 的 Step Functions 状态机
- 适用于以下内容的 IAM 策略：
  - 适用的 IAM 政策 AWS Lambda
  - 授予对 Amazon Redshift 数据 API 的访问权限

> **Note**
>
> 此示例项目可能会产生费用。
> 对于新 AWS 用户，可以使用免费使用套餐。在此套餐中，低于某种使用水平的服务是免费的。有关 AWS 费用和免费套餐的更多信息，请参阅AWS Step Functions 定价。

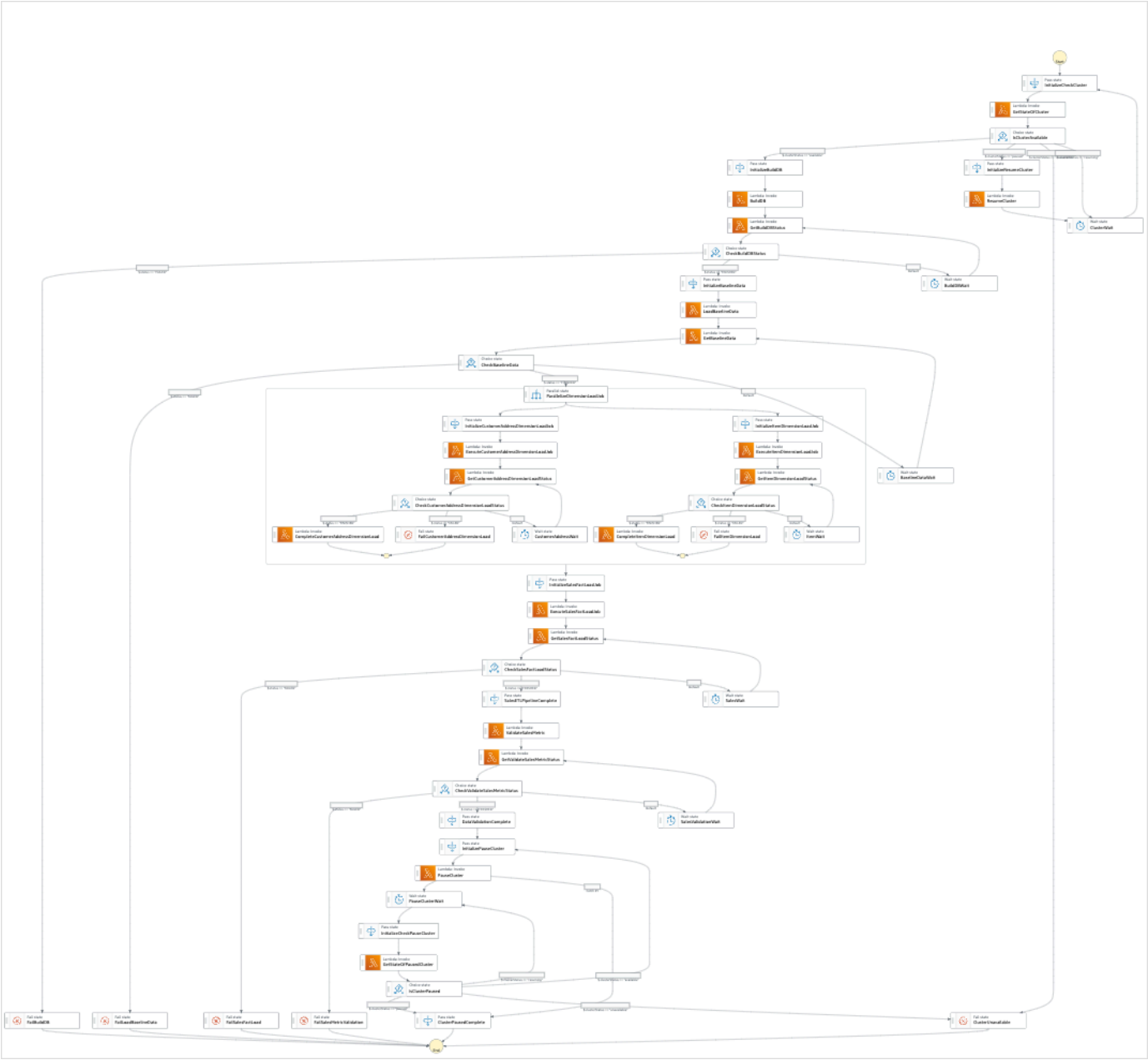
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`ETL job in Amazon Redshift`**，然后从返回的搜索结果中选择Amazon Redshift 中的 ETL 作业。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Amazon Redshift 集群
   - 两个 Lambda 函数
   - 一个 Amazon Redshift 架构
   - 五张 Amazon Redshift 表
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色。

   下图显示了 Amazon Redshift 中的 ETL 作业示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机 AWS Lambda 通过将 ETL 逻辑作为 InputPath 直接传递给这些资源并使用 Amazon Redshift 数据 API 异步执行，与之集成。

浏览此示例状态机，了解 Step Functions 是如何控制的， AWS Lambda 以及 Amazon Redshift 数据 API。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
"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
    }
```

```
  }
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

IAM 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

```
{
    "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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用 Step Functions 和 AWS Batch 进行错误处理

本示例项目演示了如何使用 Step Functions 通过具有错误处理功能的状态机运行 AWS Batch 作业。

在此项目中，Step Functions 使用状态机同步调用 AWS Batch 作业。然后，它会等待作业成功或失败，在作业失败时重试并捕捉错误，然后发送 Amazon SNS 主题，其中包含有关作业是成功还是失败的消息。

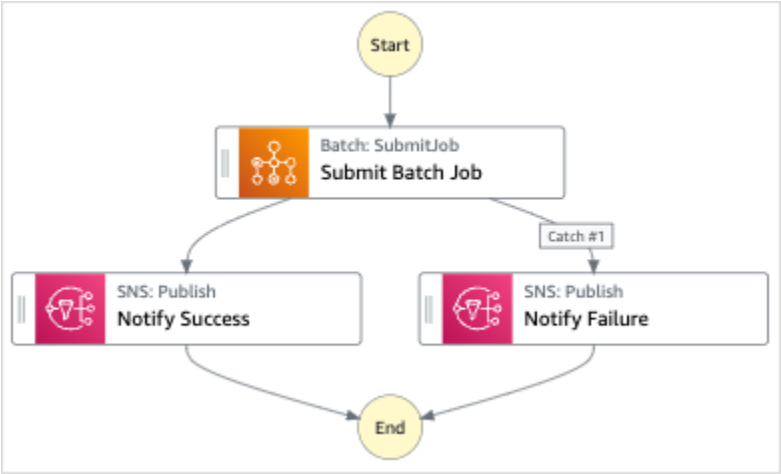
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **Manage a batch job**，然后从返回的搜索结果中选择管理批处理任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 AWS 账户，或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一份 AWS Batch 工作
   - 一个 Amazon SNS 主题
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了管理批处理任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控

制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

> **Tip**
>
> 要查看示例项目的状态机定义，请选择代码。

准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon SNS 资源来与 AWS Batch 这些资源集成。

浏览此示例状态机，通过连接到`Resource`字段中的亚马逊资源名称 (ARN) AWS Batch 并传递`Parameters`到服务 API，了解 Step Functions 如何控制和亚马逊 SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的此示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

分散一份 AWS Batch 工作

此示例项目演示了如何使用 Step Functions 的Map状态来分散 AWS Batch 作业。

在此项目中，Step Functions 使用状态机调用 Lambda 函数来进行简单的预处理，然后使用该状态并行调用多个作业 AWS Batch 。Map

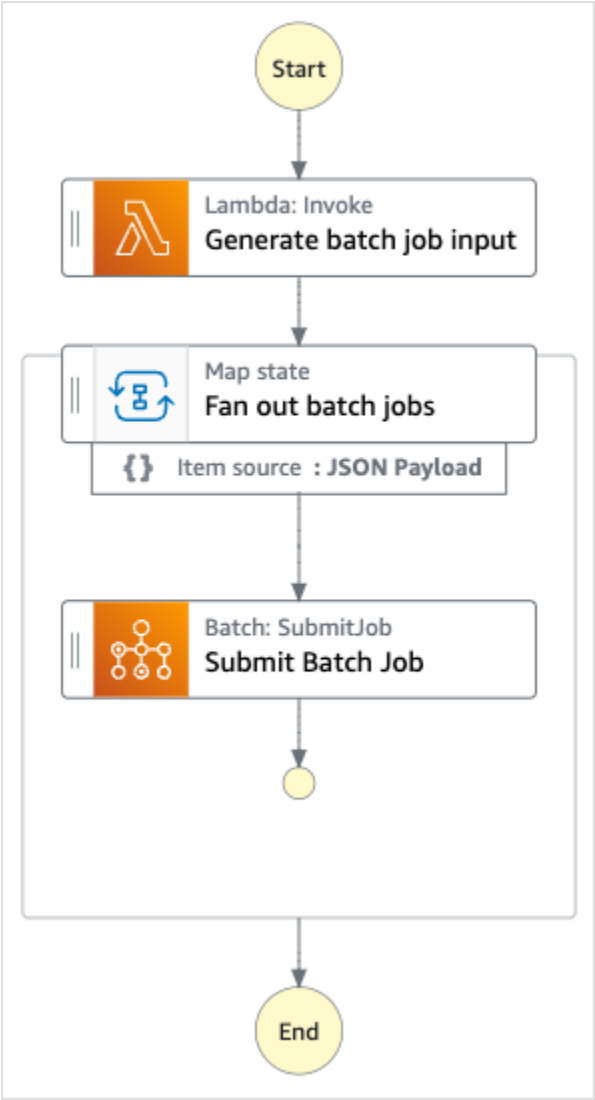
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **Fan out a batch job**，然后从返回的搜索结果中选择扇出批处理任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的 ， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Lambda 函数
   - 一个 AWS Batch 任务队列
   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了扇出批处理任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

   > **Note**
   >
   > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

3. 选择启动执行。
4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon SNS 资源来与 AWS Batch 这些资源集成。

浏览此示例状态机，通过连接到`Resource`字段中的亚马逊资源名称 (ARN) AWS Batch 并传递`Parameters`到服务 API，了解 Step Functions 如何控制和亚马逊 SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

AWS Batch 用 Lambda

此示例项目演示了如何使用 Step Functions 对包含 AWS Lambda 函数的数据进行预处理，然后编排 AWS Batch 作业。

在此项目中，Step Functions 使用状态机调用 Lambda 函数，以便在提交 AWS Batch 任务之前进行简单的预处理。根据前一个作业的结果或成功与否，可能会调用多个作业。

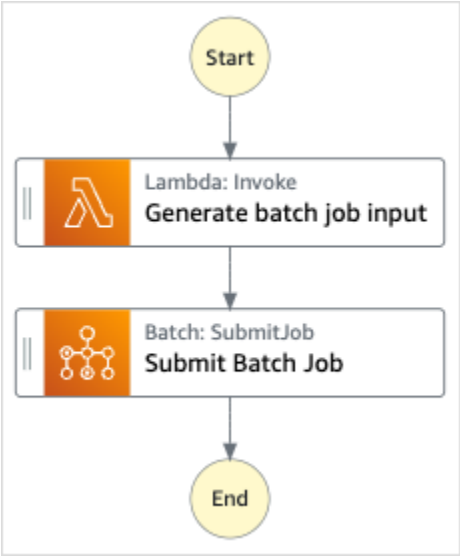
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`Batch job with Lambda`**，然后从返回的搜索结果中选择使用 Lambda 进行批处理任务。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - 一个 Lambda 函数
   - 一个 AWS Batch 作业
   - 一台 AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了使用 Lambda 进行批处理任务示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

     > **Important**
     >
     > 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

   - 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

     > **Tip**
     >
     > 要查看示例项目的状态机定义，请选择代码。

     准备就绪后，选择部署并运行以部署示例项目并创建资源。

创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

> **Important**
>
> CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：
   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

      > **Note**
      >
      > Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

   2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

      如果您选择运行演示，则无需提供任何执行输入。

      > **Note**
      >
      > 如果您部署的演示项目包含预先填充的执行输入数据，请使用该输入来运行状态机。

   3. 选择启动执行。
   4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

示例状态机代码

此示例项目中的状态机通过将参数直接传递给 Amazon SNS 资源来与 AWS Batch 这些资源集成。

浏览此示例状态机，通过连接到`Resource`字段中的亚马逊资源名称 (ARN) AWS Batch 并传递`Parameters`到服务 API，了解 Step Functions 如何控制和亚马逊 SNS。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

```
{
  "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 示例

示例项目生成的这些示例 AWS Identity and Access Management (IAM) 策略包括执行状态机和相关资源所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。

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"
        }
    ]
}
```

有关在将 Step Functions 与其他 AWS 服务一起使用时如何配置 IAM 的信息，请参阅集成服务的 IAM 策略。

使用 Amazon Bedrock 执行 AI 提示链接

此示例项目演示了如何与 Amazon Bedrock 集成以执行 AI 提示链接。此示例项目展示了如何使用 Amazon Bedrock 构建高质量聊天机器人。此项目将一些提示链接在一起，并按照提供的顺序解析它们。将提示链接起来，可增强用于提供优质响应的语言模型的能力。

此示例项目创建状态机、支持 AWS 资源并配置相关的 IAM 权限。探索此示例项目，了解如何使用 Amazon Bedrock 与 Step Functions 状态机的优化服务集成，或将其作为您自己的项目的起点。

主题

- AWS CloudFormation 模板和其他资源
- 先决条件
- 第 1 步：创建状态机并预置资源
- 第 2 步：运行状态机

AWS CloudFormation 模板和其他资源

您可以使用 CloudFormation 模板来部署此示例项目。此模板在您的中创建了以下资源 AWS 账户：

- 一个 Step Functions 状态机
- 状态机的执行角色。此角色授予您的状态机访问其他 AWS 服务 资源所需的权限，例如Amazon BedrockInvokeModel操作。

先决条件

此示例项目使用 Cohere Command 大型语言模型 (LLM)。要成功运行此示例项目，必须从 Amazon Bedrock 控制台添加对此 LLM 的访问权限。要添加模型访问权限，请执行以下操作：

1. 打开 Amazon Bedrock 控制台。
2. 在导航窗格中，选择模型访问权限。
3. 选择管理模型访问权限。
4. 选中 Cohere 旁边的复选框。
5. 选择请求访问权限。Cohere 模型的访问权限状态显示为已授予访问权限。

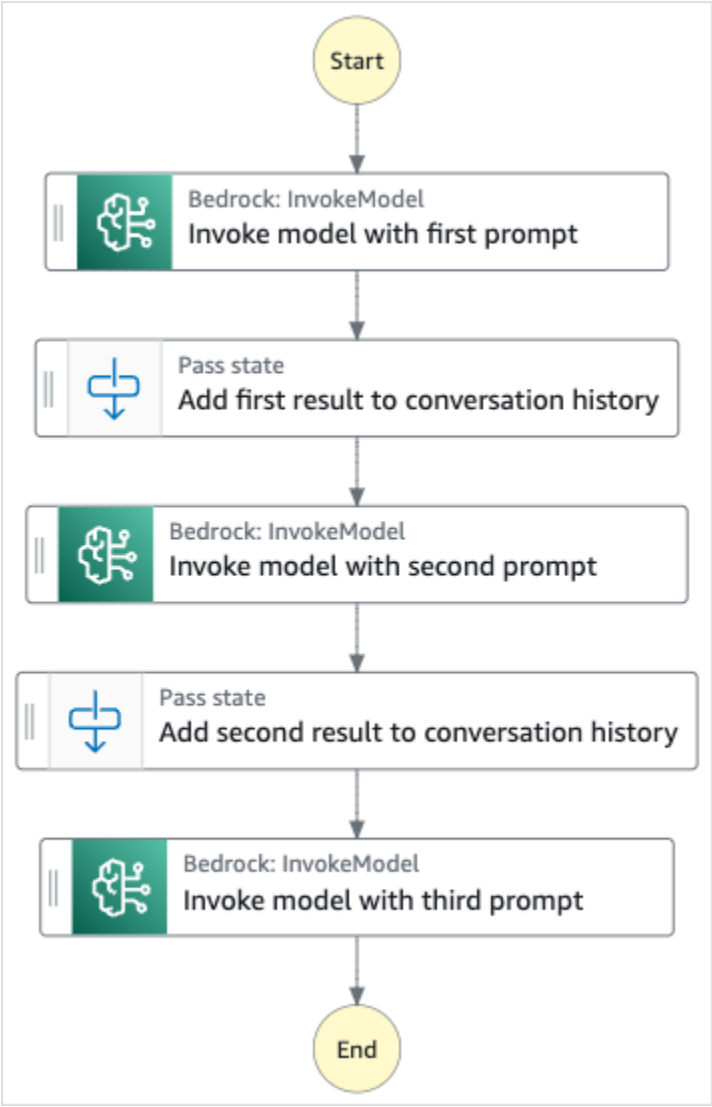
第 1 步：创建状态机并预置资源

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在搜索框中键入 **`bedrock`**，然后从返回的搜索结果中选择使用 Bedrock 执行 AI 提示链接。
3. 选择下一步以继续。
4. Step Functions 列出了您选择的示例项目中 AWS 服务 使用的。它还显示了示例项目的工作流图。将此项目部署到您的， AWS 账户 或者将其用作构建您自己的项目的起点。根据您想继续的方式，选择运行演示或构建依据。

   该示例项目部署了以下资源：

   - AWS Step Functions 状态机
   - 相关 AWS Identity and Access Management (IAM) 角色

   下图显示了使用 Bedrock 执行 AI 提示链接示例项目的工作流图：



5. 选择使用模板继续进行选择。
6. 请执行以下操作之一：

   - 如果您选择构建依据，Step Functions 将为您选择的示例项目创建工作流原型。Step Functions 不会部署工作流定义中列出的资源。

     在 Workflow Studio 的设计模式下，从状态浏览器中拖放状态，继续构建工作流原型。或者切换到代码模式，该模式提供了一个类似于 VS Code 的集成代码编辑器，用于在 Step Functions 控制台中更新状态机的 Amazon States Language（ASL）定义。有关使用 Workflow Studio 构建状态机的更多信息，请参阅使用 Workflow Studio。

> **Important**
>
> 请记住，在运行工作流之前，为示例项目中使用的资源更新占位符 Amazon 资源名称 (ARN)。

- 如果您选择了“运行演示”，Step Functions 将创建一个只读示例项目，该项目使用 AWS CloudFormation 模板将该模板中列出的 AWS 资源部署到您的 AWS 账户。

  > **Tip**
  >
  > 要查看示例项目的状态机定义，请选择代码。

  准备就绪后，选择部署并运行以部署示例项目并创建资源。

  创建这些资源和相关 IAM 权限可能需要长达 10 分钟的时间。在部署资源时，您可以打开 CloudFormation 堆栈 ID 链接以查看正在配置哪些资源。

  创建示例项目中的所有资源后，您可以在状态机页面上看到新的示例项目。

  > **Important**
  >
  > CloudFormation 模板中使用的每项服务都可能收取标准费用。

第 2 步：运行状态机

1. 在状态机页面上，选择您的示例项目。
2. 在示例项目页面上，选择启动执行。
3. 在启动执行对话框中，执行以下操作：

   1. （可选）要识别您的执行，您可以在名称框中为其指定一个名称。默认情况下，Step Functions 会自动生成一个唯一的执行名称。

> **Note**
>
> Step Functions 允许您为状态机、执行、活动和包含非 ASCII 字符的标签创建名称。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

2. （可选）在输入框中，以 JSON 格式输入输入值以便运行工作流。

   如果您选择运行演示，则无需提供任何执行输入。

3. 选择启动执行。

4. Step Functions 控制台会将您引导到一个以您的执行 ID 为标题的页面。该页面被称为执行详细信息页面。在此页面上，您可以随着执行的进展或者在执行完成后查看执行结果。

   要查看执行结果，请在图表视图上选择各个状态，然后在步骤详细信息窗格中选择各个选项卡，分别查看每个状态的详细信息，包括输入、输出和定义。有关可在执行详细信息页面上查看的执行信息的详细信息，请参阅“执行详细信息”页面 – 界面概述。

配额

AWS Step Functions 对某些状态机参数的大小设置配额，例如特定时间段内的 API 操作数量或您可以定义的状态机数量。虽然这些配额旨在防止错误配置的状态机消耗系统的所有资源，但是其中的很多限制并不是硬配额。

要请求增加服务限额，您可以执行以下操作之一：

- 访问 https://console.aws.amazon.com/servicequotas/，使用服务限额控制台。有关请求增加配额的信息，请参阅《服务限额用户指南》中的请求增加配额。
- 使用中的 Su pport AWS Management Console Center 页面申请按区域增加提供的资源的配额。AWS Step Functions 有关更多信息，请参阅 AWS 一般参考 中的 AWS 服务限额。

> **Note**
>
> 如果状态机执行或活动执行的特定阶段耗时太长，您可以配置状态机超时来引发超时事件。

主题

- 常规配额
- 与账户相关的配额
- 与 HTTP 任务相关的配额
- 与状态限制相关的配额
- 与 API 操作限制相关的配额
- 与状态机执行相关的配额
- 与任务执行相关的配额
- 与版本与别名功能相关的配额
- 与标记相关的限制

常规配额

限额	描述
Step Functions 中的名称	状态机、执行和活动任务的名称长度不得超过 80 个字符。对于您的账户和 AWS 地区，这些名称必须是唯一的，并且不得包含以下任何内容： • 空格 • 通配符 (? *) • 方括号字符 (< > { } [ ]) • 特殊字符 (" # % \ ^ | ~ ` $ & , ; : /) • 控制字符 (\\u0000 - \\u001f 或 \\u007f - \\u009f) 如果您的状态机是快速类型，则可以为状态机的多个执行提供相同的名称。即使多个执行的名称相同，Step Functions 也会为每个快速状态机执行生成唯一的执行 ARN。 Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。

与账户相关的配额

资源	默认限额	可以增加至
已注册状态机的最大数量	10000	25000
已注册活动的最大数量	10000	15000

资源	默认限额	可以增加至
最大请求大小	每个请求 1 MB。这是每个 Step Functions API 请求的总数据大小，包括请求标头以及所有其他关联的请求数据。	硬配额
每个账户打开的最大执行数	每个 AWS 区域的每个 AWS 账户 100 万次执行。超过此限制会导致 `Execution LimitExceeded` 错误。这不适用于快速工作流。	数百万
打开 Map Run 的最大数量 打开的 Map Run 是指已经开始但尚未完成的 Map Run。预定地图运行在MapRunStarted活动开始时等待，直到打开的地图运行总数少于默认配额 1000。	1000 此配额适用于分布式 Map 状态。	硬配额
Map Run redrives的最大值。	1000 此配额适用于分布式 Map 状态。	硬配额
并行 Map Run 子执行的最大数量	10000	硬配额

与 HTTP 任务相关的配额

HTTP 任务会使用令牌桶方案节流，以维护 Step Functions 服务带宽。

资源	存储桶大小	每秒的重填速率
HTTP 任务	300	300

下表列出了 HTTP 任务持续时间的配额。

资源	默认限额
HTTP 任务持续时间 HTTP 任务持续时间是指 HTTP 任务发送 HTTP 请求和收到响应所花费的时间。	60 秒 这是硬性限额，无法更改。

与状态限制相关的配额

Step Functions 状态转换使用令牌存储桶方案进行限制，以便维护服务带宽。标准工作流和快速工作流具有不同的状态转换限制。标准工作流程配额是软配额，可以提高。

> **Note**
>
> `StateTransition`服务指标的限制报告与 `ExecutionThrottled` Amazon中的情况相同。CloudWatch有关更多信息，请参阅`ExecutionThrottled` CloudWatch 指标。

	Standard		Express	
服务指标	存储桶大小	每秒的重填速率	存储桶大小	每秒的重填速率
`StateTransition` — 美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）、欧洲地区（爱尔兰）	5000	5000	无限制	无限制
`StateTransition` — 所有其他区域	800	800	无限制	无限制

与 API 操作限制相关的配额

一些 Step Functions API 操作会使用令牌存储桶方案进行限制，以便维护服务带宽。这些配额是软配额，可以提高。

> **Note**
>
> 限制配额是按账户和区域划分的 AWS 。
> AWS Step Functions 可以随时增加桶的大小和填充率。

	Standard		Express	
API 名称	存储桶大小	每秒的重填速率	存储桶大小	每秒的重填速率
`StartExecution` — 美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）、欧洲地区（爱尔兰）	1,300	300	6000	6000
`StartExecution` — 所有其他区域	800	150	6000	6000

与 TestState API 相关的配额

API 名称	限额	可以增加至
TestState	每秒 1 个事务 (TPS)	硬配额

其他配额

这些配额是软配额，可以提高。

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名称	存储桶大小	每秒的重填速率	存储桶大小	每秒的重填速率
CreateActivity	100	1	100	1
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	3000	500	1500	300
GetExecutionHistory	400	20	400	20

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名称	存储桶大小	每秒的重填速率	存储桶大小	每秒的重填速率
`ListActivities`	100	10	100	5
`ListExecutions`	200	5	100	2
`ListStateMachines`	100	5	100	5
`ListTagsForResource`	100	1	100	1
`SendTaskFailure`	3000	500	1500	300
`SendTaskHeartbeat`	3000	500	1500	300
`SendTaskSuccess`	3000	500	1500	300
StartSyncExecution	同步快速执行 API 调用不会影响现有的账户容量限制。Step Functions 按需提供容量，并根据持续的工作负载自动扩展。在容量扩展之前，可以限制工作负载激增。 如果您遇到限制，请稍后重试。有关同步快递工作流的信息，请参见同步和异步快速工作流。			
`StopExecution`	1000	200	500	25
`TagResource`	200	1	200	1
`UntagResource`	200	1	200	1

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 名称	存储桶大小	每秒的重填速率	存储桶大小	每秒的重填速率
`UpdateStateMachine`	100	1	100	1

与状态机执行相关的配额

下表描述了与状态机执行相关的配额。状态机执行配额是硬配额，不可更改，执行历史记录保留时间配额除外。

限额	Standard	Express
最长执行时间	1 年。如果执行的持续时间超过 1 年的最大值，它将因 `States.Timeout` 错误而失败并发出一个`ExecutionsTimedOut` CloudWatch 指标。	5 分钟。如果执行的运行时间超过 5 分钟的最大值，它将因`States.Timeout` 错误而失败并发出`ExecutionsTimedOut` CloudWatch 指标。
最大执行历史记录大小	单个状态机执行历史记录中有 25000 个事件。如果执行历史记录达到此配额，则执行将失败。要避免这种情况，请参阅避免达到历史记录的配额。	无限制。
最长执行空闲时间	1 年（受最长执行时间限制）。	5 分钟（受最长执行时间限制）。
执行历史记录保留时间	执行结束后 90 天。此时间过后，无法再检索或查看执行历史。对 Step Functions 保留的已关闭执行数没有进一步的配额。	要查看执行历史记录，必须配置 Amazon CloudWatch 日志记录。有关更多信息，请参阅使用 CloudWatch Logs 进行日志记录。

限额	Standard	Express
	为了满足合规性、组织或监管要求，您可以通过发送配额请求，将执行历史记录的保留期缩短至 30 天。为此，请使用 AWS Support Center Console 并创建一个新案例。 将保留期缩短至 30 天的更改适用于区域中的每个账户。	
执行redrivable期 Redrivable期是指您可以redrive特定标准工作流执行的时间。这段时间从状态机完成执行的当天开始计算。	14 天。 此硬配额适用于分布式 Map 状态。	Redrive目前不支持快速工作流。

与任务执行相关的配额

下表描述了与任务执行相关的配额。这些都是无法更改的硬配额。

限额	Standard	Express
最长任务执行时间	1 年（受最长执行时间限制）	5 分钟（受最长执行时间限制）
Step Functions 将任务保留在队列中的最长时间	1 年（受最长执行时间限制）	5 分钟（受最长执行时间限制）
每个 Amazon 资源名称 (ARN) 的最大活动轮询器数	每 ARN 1000 个调用 `GetActivityTask` 的轮询器。超出此配额将导致以下错误：“The maximum number of workers concurrently polling for activity tasks has been	不适用于快速工作流。

限额	Standard	Express
	reached (已达到并发轮询活动任务的最大工作线程数目)。”	
任务、状态或执行的最大输入或输出大小	256 KB 数据，UTF-8 编码字符串。此配额影响计划任务、进入状态或启动执行时的任务（活动、Lambda 函数或集成服务）、状态或执行输出以及输入数据。	256 KB 数据，UTF-8 编码字符串。此配额影响计划任务、进入状态或启动执行时的任务（活动、Lambda 函数或集成服务）、状态或执行输出以及输入数据。

与版本与别名功能相关的配额

资源	默认限额
已发布状态机版本的最大数量	每个状态机 1000 个。 要请求提高此软限制，请使用 AWS Management Console中的支持中心页面。
最大状态机别名数	每个状态机 100 个。 要请求提高此软限制，请使用 AWS Management Console中的支持中心页面。

与标记相关的限制

在标记 Step Functions 资源时，请注意以下限制。

> **Note**
>
> 标记限制不能像其他配额一样提高。

限制	描述
每个资源的最大标签数	50
最大密钥长度	128 个 Unicode 字符（采用 UTF-8 格式）
最大值长度	256 个 Unicode 字符（采用 UTF-8 格式）
前缀限制	请勿在标签名称或值中使用`aws:`前缀，因为它已保留供 AWS 使用。您无法编辑或删除带此前缀的标签名称或值。具有此前缀的标签不计入每个资源的标签数配额。
字符限制	标签只能包含 Unicode 字母、数字、空格或以下符号：`_ . : / = + - @`

登录和监控 AWS Step Functions

记录和监控对于维护 Step Functions 和 AWS 解决方案的可靠性、可用性和性能非常重要。有多种工具可与 Step Functions 配合使用：

> **Tip**
>
> 要将示例工作流程部署到您 AWS 账户 并学习如何监控工作流程执行的指标、日志和跟踪，请参阅模块 12- AWS Step Functions 研讨会的可观察性。

主题

- 使用监控 Step Functions CloudWatch
- EventBridge Step Functions 的（CloudWatch 事件）执行状态发生了变化
- 使用录制 API 调用 AWS CloudTrail
- 使用 CloudWatch Logs 进行日志记录
- AWS X-Ray 和 Step Functions
- 配合使用 AWS 用户通知服务和 AWS Step Functions

使用监控 Step Functions CloudWatch

监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS Step Functions 您应该从所使用的 AWS 服务中收集尽可能多的监控数据，以便可以调试多点故障。在开始监控 Step Functions 之前，您应创建一个监控计划，回答以下问题：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

下一步是为您的环境中的正常 性能设置基准。为此，在不同时间和不同负载条件下测量性能。监控 Step Functions 时，请考虑存储历史监控数据。此类数据可为您提供与当前性能数据进行比较的基准，用于确定正常性能模式和性能异常，以及设计解决问题的方法。

例如，使用 Step Functions，您可以监控有多少活动或 AWS Lambda 任务由于心跳超时而失败。在性能低于您建立的基准时，您可能需要更改检测信号时间间隔。

要建立基准，您至少应监控以下指标：

- `ActivitiesStarted`
- `ActivitiesTimedOut`
- `ExecutionsStarted`
- `ExecutionsTimedOut`
- `LambdaFunctionsStarted`
- `LambdaFunctionsTimedOut`

以下各节介绍了 Step Functions 向亚马逊提供的指标 CloudWatch。您可以使用这些指标来跟踪状态机和活动，以及设置有关阈值的警报。您可以使用查看指标 AWS Management Console。

报告时间间隔的指标

Step Functions 的某些 Step Function CloudWatch s 指标是时间间隔，始终以毫秒为单位进行测量。这些指标通常对应于可以设置状态机、活动和 Lambda 函数超时的执行的各个阶段，并且带有说明性名称。

例如，`ActivityRunTime` 指标测量活动在开始执行后，完成活动所需的时间。您可以为同一个时段设置超时值。

在 CloudWatch 控制台中，如果选择平均值作为时间间隔指标的显示统计数据，则可以获得最佳结果。

报告计数的指标

某些 Step Function CloudWatch s 指标将结果报告为计数。例如，`ExecutionsFailed` 记录失败的状态机执行次数。

Step Functions 每次执行状态机都会发出两个`ExecutionsStarted`指标。这会使`ExecutionsStarted`指标的SampleCount统计数据在每次执行状态机时显示值为 2。SampleCount 统计数据显示执行`ExecutionStarted=0`完成的时间`ExecutionStarted=1`和时间。

> **Tip**
>
> 对于在 CloudWatch 控制台中报告计数的指标，我们建议选择 Sum 作为显示统计数据。

执行指标

AWS/States命名空间包括所有 Step Functions 执行的以下指标。这些是适用于您所在地区账户的无维度指标。

指标	描述
OpenExecutionCount	当前打开的执行的大致数量，即您的账户中当前正在进行的工作流程。 目的是深入了解您的工作流程何时接近最大执行限制，以避免在调用StartExecution 或使用标准工作流程时RedriveExecution 出ExecutionLimitExceeded错。 该指标取决于活动的工作流状态转换，因此在较低的级别上，估计值可能与观察到的运行工作流程数量不一致。
OpenExecutionLimit	打开的最大执行次数。有关更多信息，请参阅 与账户相关的配额。 此限制不适用于 Express 工作流程。

带有版本或别名的状态机的执行指标

当您使用版本或别名运行状态机执行时，Step Functions 会发出以下指标。只有在执行受限的情况下，才会发出该ExecutionThrottled指标。这些指标将包括StateMachineArn用于识别特定状态机的指标。

指标	描述
ExecutionTime	从开始执行到执行结束的时间间隔，以毫秒为单位。

指标	描述
ExecutionThrottled	已被限制StateEntered 的事件和重试次数。这与 StateTransition 限制有关。有关更多信息，请参阅 与状态限制相关的配额。
ExecutionsAborted	中止或终止的执行次数。
ExecutionsFailed	失败的执行次数。
ExecutionsStarted	已启动的执行次数。
ExecutionsSucceeded	成功完成的执行次数。
ExecutionsTimedOut	因任何原因而超时的执行次数。

快速工作流的执行指标

AWS/States 命名空间包括 Step Functions 快速工作流执行的以下指标。

指标	描述
ExpressExecutionMemory	快速工作流消耗的总内存。
ExpressExecutionBilledDuration	快速工作流的收费时长。
ExpressExecutionBilledMemory	快速工作流程收费的内存消耗量。

标准工作流的Redrive执行指标

redrive执行状态机时，Step Functions 会发出以下指标。

对于所有redriven执行，都会发出 Executions* 指标。例如，假设redriven执行中止。该执行将同时发出 RedrivenExecutionsAborted 和 ExecutionsAborted 的非零数据点。

指标	描述
ExecutionsRedriven	redriven处决次数。
RedrivenExecutions Aborted	取消或终止的redriven执行次数。
RedrivenExecutions TimedOut	因任何原因而超时的redriven执行次数。
RedrivenExecutions Succeeded	成功完成的redriven执行次数。
RedrivenExecutions Failed	失败的redriven执行次数。

Step Functions 执行指标的维度

维度	描述
StateMachineArn	所涉执行的状态机的 Amazon 资源名称 (ARN)。

使用版本的执行维度

维度	描述
StateMachineArn	由版本启动执行的状态机的 Amazon 资源名称 (ARN)。
Version	用于启动执行的状态机版本。

使用别名的执行维度

维度	描述
StateMachineArn	由别名启动执行的状态机的 Amazon 资源名称 (ARN)。

维度	描述
Alias	用于启动执行的状态机别名。

版本与别名功能的资源计数指标

AWS/States 命名空间包括状态机版本与别名功能计数的以下指标。

指标	描述
AliasCount	为状态机创建的别名数量。 您最多可以为每个状态机创建 100 个别名。
VersionCount	为状态机发布的版本数。 您最多可以发布 1000 个版本的状态机。

版本与别名功能的资源计数指标维度

维度	描述
ResourceArn	使用别名或版本的状态机的 Amazon 资源名称 (ARN)。

活动指标

AWS/States 命名空间包括 Step Functions 活动的以下指标。

指标	描述
ActivityRunTime	活动开始和关闭之间的间隔，以毫秒为单位。
ActivityScheduleTime	活动保持计划状态的时间间隔，以毫秒为单位。
ActivityTime	活动计划时间与活动结束时间之间的间隔，以毫秒为单位。

指标	描述
ActivitiesFailed	失败的活动数。
ActivitiesHeartbeatTimedOut	由于心跳超时而超时的活动数量。
ActivitiesScheduled	预定活动数量。
ActivitiesStarted	已启动的活动数。
ActivitiesSucceeded	成功完成的活动数量。
ActivitiesTimedOut	关闭时超时的活动数量。

Step Functions 活动指标维度

维度	描述
ActivityArn	活动的 ARN。

Lambda 函数指标

AWS/States 命名空间包括 Step Functions Lambda 函数的以下指标。

指标	描述
LambdaFunctionRunTime	Lambda 函数启动和关闭之间的间隔，以毫秒为单位。
LambdaFunctionScheduleTime	Lambda 函数保持计划状态的时间间隔，以毫秒为单位。
LambdaFunctionTime	调度 Lambda 函数的时间与其关闭之间的间隔，以毫秒为单位。
LambdaFunctionsFailed	失败的 Lambda 函数的数量。

指标	描述
LambdaFunctionsScheduled	计划的 Lambda 函数的数量。
LambdaFunctionsStarted	已启动的 Lambda 函数的数量。
LambdaFunctionsSucceeded	成功完成的 Lambda 函数的数量。
LambdaFunctionsTimedOut	关闭时超时的 Lambda 函数的数量。

Step Functions Lambda 函数指标维度

维度	描述
LambdaFunctionArn	Lambda 函数的 ARN。

> **Note**
>
> 对于在 `Resource` 字段中指定 Lambda 函数 ARN 的 Task 状态，会发出 Lambda 函数指标。而使用 `"Resource": "arn:aws:states:::lambda:invoke"` 的 Task 状态会发出服务集成指标。有关更多信息，请参阅 使用 Step Functions 调用 Lambda。

服务集成指标

`AWS/States` 命名空间包括 Step Functions 服务集成的以下指标。有关更多信息，请参阅 与其他服务 AWS Step Functions 一起使用。

指标	描述
ServiceIntegrationRunTime	服务任务启动和关闭之间的间隔，以毫秒为单位。

指标	描述
ServiceIntegrationScheduleTime	服务任务保持计划状态的时间间隔，以毫秒为单位。
ServiceIntegrationTime	从调度服务任务到服务任务关闭时间之间的间隔，以毫秒为单位。
ServiceIntegrationsFailed	失败的服务任务数。
ServiceIntegrationsScheduled	计划服务任务数。
ServiceIntegrationsStarted	已启动的服务任务数。
ServiceIntegrationsSucceeded	成功完成的服务任务数。
ServiceIntegrationsTimedOut	关闭时超时的服务任务数。

Step Functions 服务集成指标的维度

维度	描述
ServiceIntegrationResourceArn	集成服务的资源 ARN。

服务指标

AWS/States 命名空间包括 Step Functions 服务的以下指标。

指标	描述
ThrottledEvents	已被限制的请求数。

指标	描述
ProvisionedBucketSize	每秒可用请求数。
ProvisionedRefillRate	每秒允许进入存储桶的请求数。
ConsumedCapacity	每秒请求数。

Step Functions 服务指标的维度

维度	描述
ServiceMetric	筛选数据以显示状态转换指标。

API 指标

AWS/States 命名空间包括 Step Functions API 的以下指标。

指标	描述
ThrottledEvents	已被限制的请求数。
ProvisionedBucketSize	每秒可用请求数。
ProvisionedRefillRate	每秒允许进入存储桶的请求数。
ConsumedCapacity	每秒请求数。

Step Functions API 指标的维度

维度	描述
APIName	按照指定 API 名称的 API 来筛选数据。

尽力交付 CloudWatch 指标

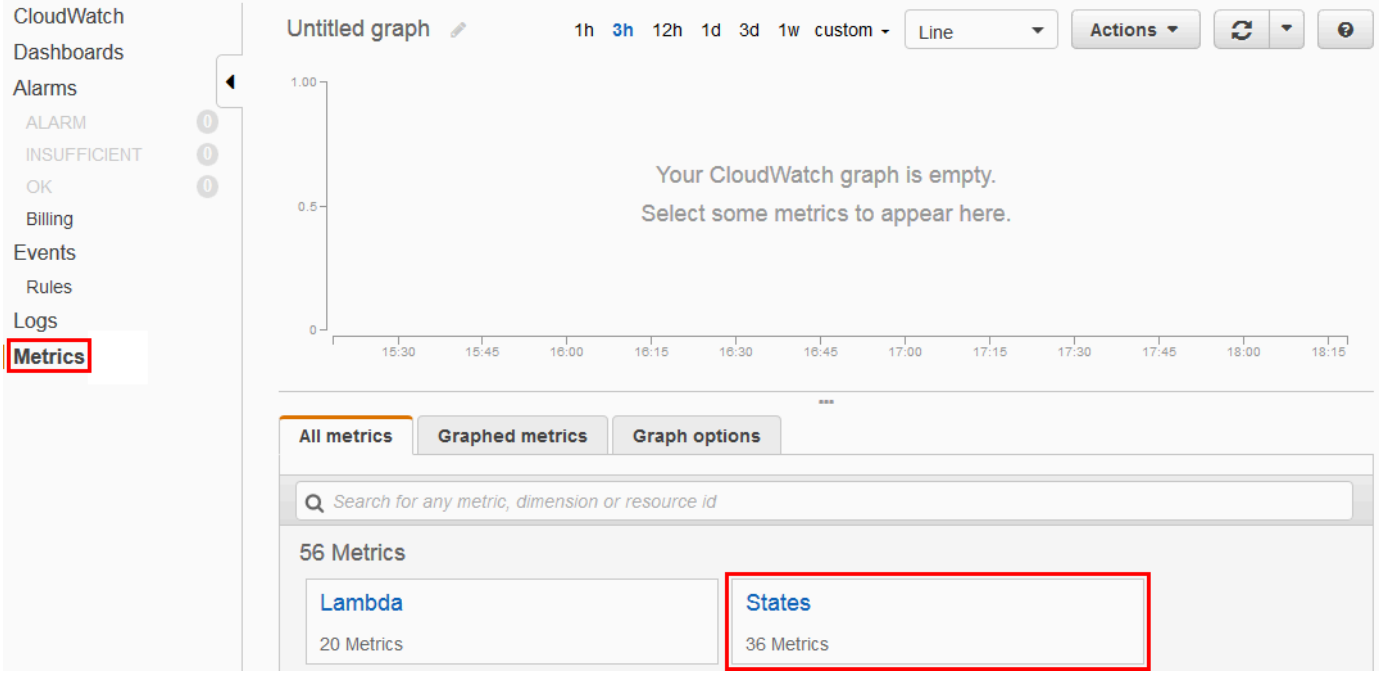
系统将以最大努力传输 CloudWatch 指标。

无法保证指标的完整性和及时性。可能返回特定请求的数据点，其时间戳晚于实际处理请求的时间。数据点在通过之前可能会延迟一分钟 CloudWatch，或者可能根本无法传送。 CloudWatch请求指标可以让你近乎实时地了解状态机的执行情况。这并不代表会完整记录所有与执行相关的指标。

根据此特征的最大努力性质，在账单和成本管理控制面板提供的报告中可能有一个或多个访问请求不会出现在执行指标中。

查看 Step Functions 的指标

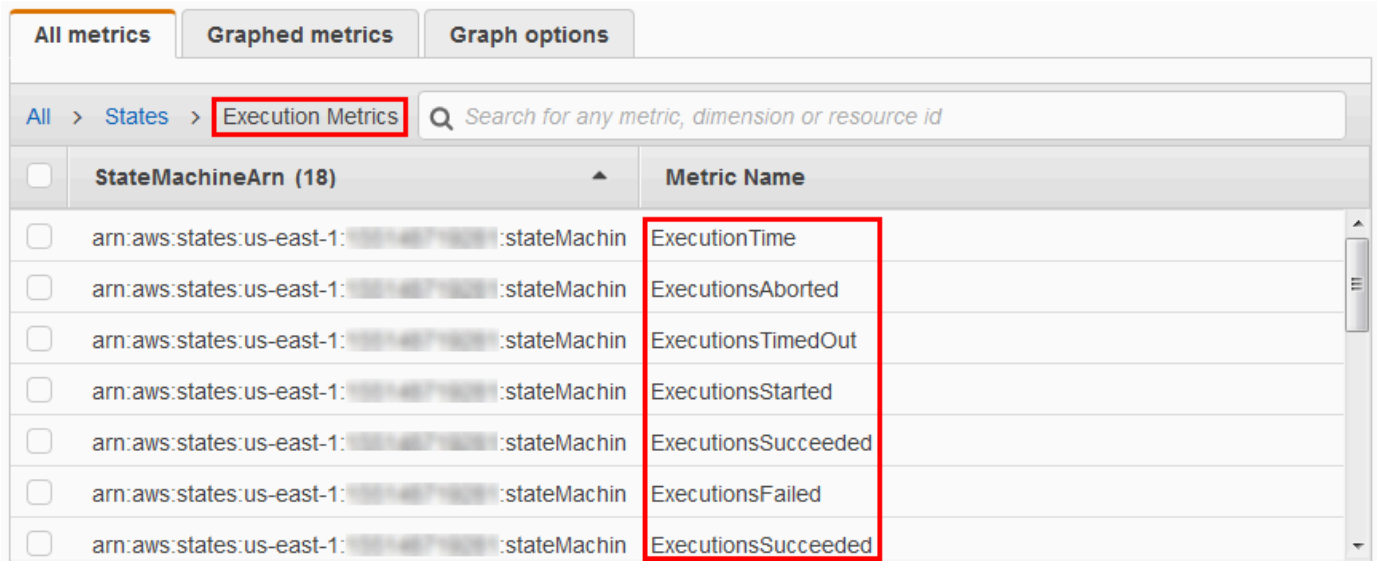
1. 登录 AWS Management Console 并打开 CloudWatch 控制台。
2. 选择指标，在全部指标选项卡上选择状态。



如果您近期运行了任何执行，最多可以看到四种类型的指标：

- 执行指标
- 活动函数指标
- Lambda 函数指标
- 服务集成指标

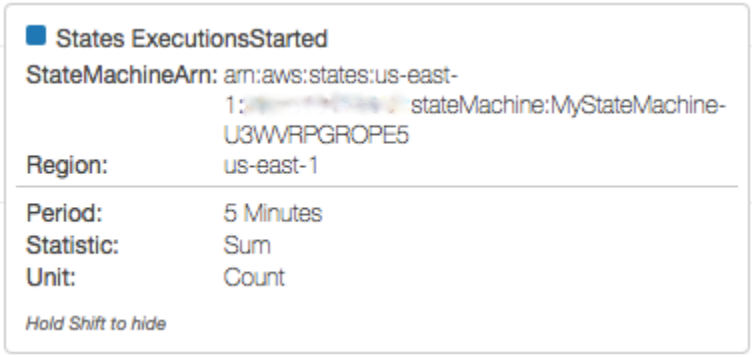
3. 选择指标类型以查看指标列表。



- 要按指标名称或对指标进行排序 StateMachineArn，请使用列标题。
- 要查看指标图表，请选中列表上指标旁边的复选框。您可以使用图表视图上方的时间范围控件更改图表参数。

  您可以使用相对或绝对值 (特定天和时间) 选择自定义时间范围。您还可以使用下拉列表将值显示为线条、堆叠区域或数字（值）。
- 要查看有关图表的详细信息，请悬停在图表下方显示的指标颜色代码上。

  此时将显示指标详细信息。



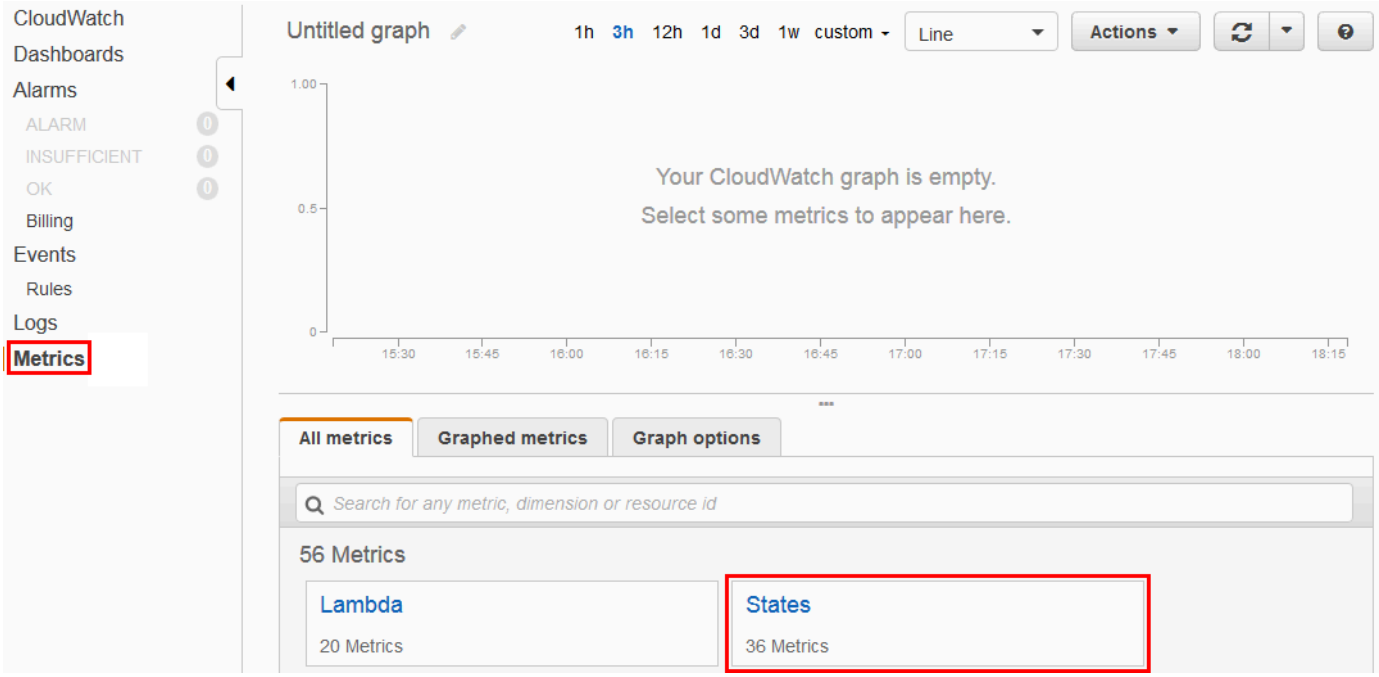
有关使用 CloudWatch 指标的更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch 指标。

为 Step Functions 设置警报

您可以使用 Amazon CloudWatch 警报来执行操作。例如，如果您希望知道何时达到警报阈值，您可以设置警报以发送通知到 Amazon SNS 主题，或者在 `StateMachinesFailed` 指标上升到超过特定阈值时发送电子邮件。

在指标上设置警报

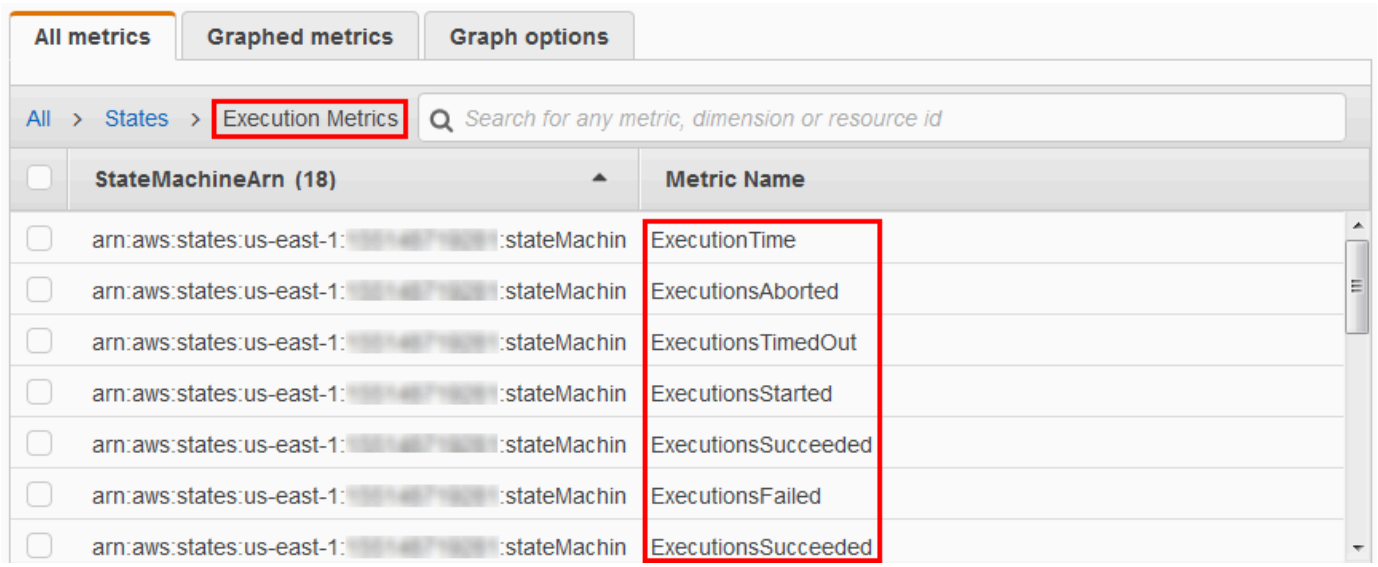
1. 登录 AWS Management Console 并打开 CloudWatch 控制台。
2. 选择指标，在全部指标选项卡上选择状态。



如果您近期运行了任何执行，最多可以看到四种类型的指标：

- 执行指标
- 活动函数指标
- Lambda 函数指标
- 服务集成指标

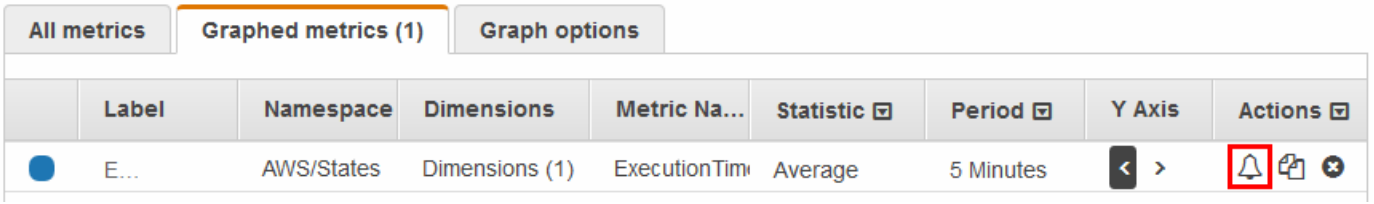
3. 选择指标类型以查看指标列表。



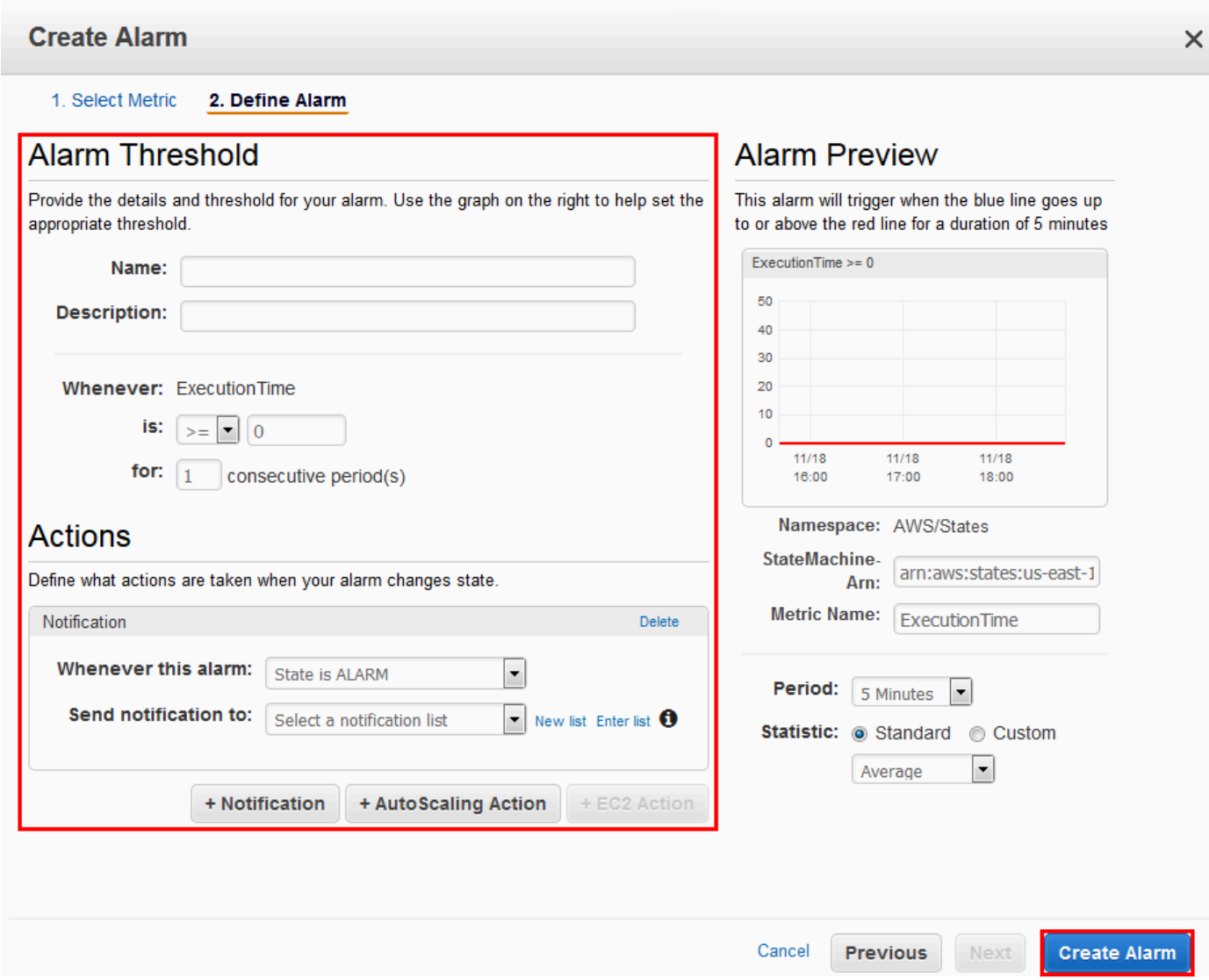
4. 选择一个指标，然后选择绘成图表的指标。

5. 选择列表上指标旁边的

。



此时将显示创建警报页面。



6. 为警报阈值和操作输入值，然后选择创建警报。

有关设置和使用 CloudWatch 警报的更多信息，请参阅亚马逊 CloudWatch 用户指南中的创建亚马逊 CloudWatch 警报。

EventBridge Step Functions 的（CloudWatch 事件）执行状态发生了变化

Amazon EventBridge 是一项使您能够响应 AWS 资源状态变化的 AWS 服务。例如，您可以使用以下两种方式响应 Step Function EventBridge s 标准工作流程的执行状态变化：

- 您可以配置 EventBridge 规则以对在 Step Functions 状态机的执行状态发生变化时发出的事件做出反应。这使您能够监控您的工作流，而不必持续不断地使用 DescribeExecution API 进行轮询。根据状态机执行的变化，您可以使用 EventBridge 目标来启动新的状态机执行、调用 AWS Lambda 函数、向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 主题发布消息等。
- 您也可以将 Step Functions 状态机配置为中的目标 EventBridge。这使您能够触发 Step Functions 工作流的执行，用于响应另一项 AWS 服务中的事件。

  有关更多信息，请参阅 Amazon EventBridge 用户指南。

但是，Express Workflows 不会向发送事件。 EventBridge要监控 Express 工作流程的执行情况，您可以使用 CloudWatch 日志。为此，在状态机的执行详细信息 页面上，选择监控和日志记录选项卡。在“监控” 选项卡上，您可以查看事件的 CloudWatch 指标，例如执行时长、执行错误和计费内存。在日志记录选项卡上，您可以查看最近的日志和日志记录配置。

> **Tip**
>
> 要向你部署 Express 工作流程的示例 AWS 账户 并学习如何监控 Express Workflows，请参阅 AWS Step Functions 研讨会的 “监控 Express 工作流程” 模块。

EventBridge 有效载荷

EventBridge 事件的定义中可以包含输入属性。对于某些事件， EventBridge事件还可以在其定义中包含输出属性。

- 如果发送到的转义输入和转义输出的组合 EventBridge 超过 248KB，则该输入将被排除在外。同样，如果转义输出也超过 248KB，则该输出也将被排除在外。这是 EventBridge活动配额的结果。
- 您可以使用 `inputDetails` 和 `outputDetails` 属性确定有效负载是否已被截断。有关更多信息，请参阅 `CloudWatchEventsExecutionDataDetails` 数据类型。
- 对于标准工作流程，您可以使用查看完整的输入和输出DescribeExecution。
- `DescribeExecution` 不适用于快速工作流。如果想查看完整的输入/输出，则可以用标准工作流封装快速工作流。另一种方法是使用 Amazon S3 ARN。有关使用 ARN 的信息，请参阅the section called “使用 Amazon S3 ARN 而不是传递大量有效负载”。

主题

- Step Functions 事件示例

- 将 Step Functions 事件路由到 EventBridge 控制台 EventBridge 中

Step Functions 事件示例

以下是 Step Functions 向发送事件的示例 EventBridge：

主题

- 执行已启动
- 执行已成功
- 执行已失败
- 执行超时
- 执行已中止

在每种情况下，事件数据中的 `detail` 部分都提供与 `DescribeExecution` API 相同的信息。`status` 字段指明发送事件时执行的状态，根据发送的事件，状态可以是：`RUNNING`、`SUCCEEDED`、`FAILED`、`TIMED_OUT` 或 `ABORTED`。

执行已启动

```
{
    "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
    }
}
```

执行已成功

```
{
    "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
        }
    }
}
```

执行已失败

```
{
    "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
    }
}
```

执行超时

```
{
    "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
    }
}
```

执行已中止

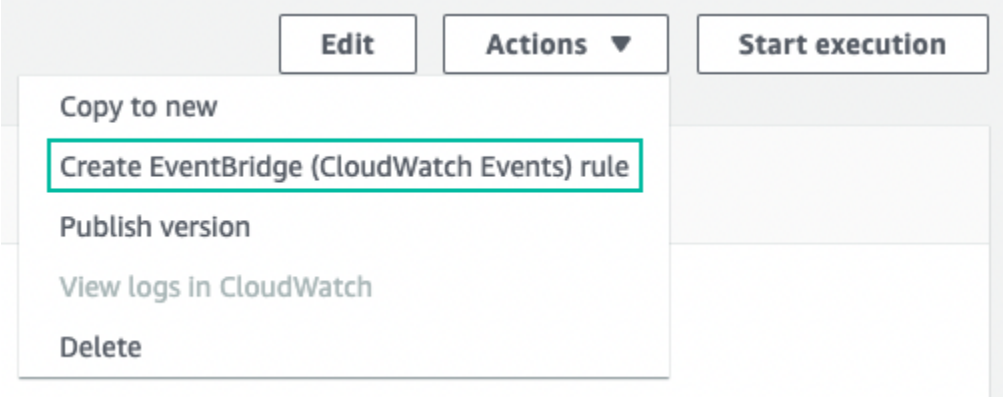
```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
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
    }
}
```

将 Step Functions 事件路由到 EventBridge 控制台 EventBridge 中

使用以下说明，学习如何在特定的 Step Functions 状态机成功运行时触发 Step Functions 状态机的执行。您可以使用 Amazon EventBridge 控制台指定要触发其执行的状态机。

1. 在状态机的详细信息页面上，选择操作，然后选择创建 EventBridge （CloudWatch 事件）规则。



   或者，也可以通过 https://console.aws.amazon.com/events/ 打开 EventBridge 控制台。在导航窗格中的总线下，选择规则。
2. 选择创建规则。这将打开定义规则详细信息页面。
3. 输入规则的名称（例如 *StepFunctionsEventRule*），然后根据需要输入规则的描述。
4. 对于事件总线和规则类型，请保留默认选择。
5. 选择下一步。这将打开构建事件模式页面。
6. 在 “事件源” 下，保留AWS 事件或 EventBridge 合作伙伴事件的默认选择。
7. 保留示例事件和创建方法部分的默认选择。
8. 对于事件模式，执行以下操作：

   a. 在事件来源下拉列表中，保留AWS 服务的默认选择。

   b. 从 AWS 服务下拉列表中选择 Step Functions。

c. 从事件类型下拉列表中，选择 Step Functions 执行状态更改。

d. （可选）配置特定的状态、状态机 Amazon 资源名称 (ARN) 或执行 ARN。在此过程中，选择特定状态，然后从下拉列表中选择成功。

9. 选择下一步。这将打开选择目标页面。
10. 在目标类型下，保留默认的 AWS 服务选择。
11. 从 “选择目标” 下拉列表中，选择一项 AWS 服务。例如，您可以启动 Lambda 函数或运行 Step Functions 状态机。对于此过程，请选择 Step Functions 状态机。
12. 从状态机下拉列表中，选择一个状态机。
13. 在执行角色下，保留为此特定资源创建新角色的默认选择。
14. 选择下一步。这将打开配置标签页面。
15. 再次选择下一步。这将打开查看并创建页面。
16. 查看规则详细信息并选择创建规则。

    规则已创建并显示 “规则” 页面，其中列出了您的所有 Amazon EventBridge 规则。

使用录制 API 调用 AWS CloudTrail

AWS Step Functions 与AWS CloudTrail一项服务集成，该服务提供用户、角色或角色所执行操作的记录 AWS 服务。 CloudTrail 将所有 API 调用捕获 Step Functions 为事件。捕获的调用包括来自 Step Functions 控制台的调用和对 Step Functions API 操作的代码调用。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 Step Functions、发出请求的 IP 地址、发出请求的时间以及其他详细信息。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

CloudTrail 在您创建账户 AWS 账户 时在您的账户中处于活动状态，并且您自动可以访问 CloudTrail 活动历史记录。 CloudTrail 事件历史记录提供了过去 90 天中记录的管理事件的可查看、可搜索、可下载且不可变的记录。 AWS 区域有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 事件历史记录”。查看活动历史记录不 CloudTrail收取任何费用。

要持续记录 AWS 账户 过去 90 天内的事件，请创建跟踪或 CloudTrailLake 事件数据存储。

CloudTrail 步道

跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。使用创建的所有跟踪 AWS Management Console 都是多区域的。您可以通过使用 AWS CLI创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可以捕获账户 AWS 区域 中的所有活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以免费将正在进行的管理事件的一份副本传送到您的 Amazon S3 存储桶，但会收取 Amazon S3 存储费用。 CloudTrail 有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail 湖泊事件数据存储

CloudTrail L@@ ak e 允许你对自己的事件运行基于 SQL 的查询。 CloudTrail Lake 将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅AWS CloudTrail 用户指南中的使用 AWS CloudTrail Lake。

CloudTrail 湖泊事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

中的数据事件 CloudTrail

数据事件可提供对资源或在资源中所执行资源操作（例如，读取或写入 Amazon S3 对象）的相关信息。这些也称为数据层面操作。数据事件通常是高容量活动。默认情况下， CloudTrail 不记录数据事件。 CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录 Step Functions 资源类型的数据事件。AWS CLI有关如何记录数据事件的更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS Management Console记录数据事件和使用 AWS Command Line Interface记录数据事件。

下表列出了您可以记录数据事件的 Step Functions 资源类型。数据事件类型列显示要从 CloudTrail 控制台上的数据事件类型列表中选择的值。res ources.type 值列显示该`resources.type`值，您将在使用或 API 配置高级事件选择器时指定该值。 AWS CLI CloudTrail “记录到的数据 API CloudTrail” 列显示了 CloudTrail 针对该资源类型记录的 API 调用。

您可以将高级事件选择器配置为在 `eventName`、`readOnly` 和 `resources.ARN` 字段上进行筛选，从而仅记录那些对您很重要的事件。有关这些字段的更多信息，请参阅《AWS CloudTrail API 参考》中的 AdvancedFieldSelector。

数据事件类型	resources.type 值	数据 API 已登录到 CloudTrail
Step Functions 状态机	`AWS::StepFunctions::StateMachine`	• InvokeHTTPEndpoint

中的管理活动 CloudTrail

管理事件提供有关对中的资源执行的管理操作的信息 AWS 账户。这些也称为控制层面操作。默认情况下，CloudTrail 记录管理事件。

状态机

- CreateStateMachine
- ListStateMachines
- DescribeStateMachine
- UpdateStateMachine
- DeleteStateMachine
- ValidateStateMachineDefinition
- TestState

状态机别名

- CreateStateMachineAlias
- ListStateMachineAliases
- DescribeStateMachineAlias
- UpdateStateMachineAlias
- DeleteStateMachineAlias

状态机版本

- ListStateMachineVersions

- PublishStateMachineVersion
- DeleteStateMachineVersion

执行

- StartExecution
- StartSyncExecution
- RedriveExecution
- ListExecutions
- DescribeExecution
- GetExecutionHistory
- DescribeStateMachineForExecution
- StopExecution

活动

- CreateActivity
- ListActivities
- DescribeActivity
- DeleteActivity
- GetActivityTask

任务令牌

- SendTaskSuccess
- SendTaskHeartbeat
- SendTaskFailure

MapRun

- ListMapRuns
- DescribeMapRun
- UpdateMapRun

标签

- ListTagsForResource
- TagResource
- UntagResource

事件示例

事件代表来自任何来源的单个请求，包括有关所请求的 API 操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此事件不会按任何特定顺序出现。

以下示例显示了一个演示 CloudTrail 的数据事件InvokeHTTPEndpoint。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "states.amazonaws.com"
    },
    "eventTime": "2024-05-01T01:23:45Z",
    "eventSource": "states.amazonaws.com",
    "eventName": "InvokeHTTPEndpoint",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "states.amazonaws.com",
    "userAgent": "states.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::StepFunctions::StateMachine",
            "ARN": "arn:aws:states:us-
east-1:123456789012:stateMachine:ExampleStateMachine"
        }
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": false,
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "httpMethod": "GET",
```

```
        "httpEndpoint": "https://example.com"
    },
    "eventCategory": "Data"
}
```

以下示例显示了一个演示该CreateStateMachine操作的 CloudTrail 管理事件。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAJYDLDBVBI4EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/test-user",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "test-user"
    },
    "eventTime": "2024-05-01T01:23:45Z",
    "eventSource": "states.amazonaws.com",
    "eventName": "CreateStateMachine",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "name": "MyStateMachine",
        "definition": "HIDDEN_DUE_TO_SECURITY_REASONS",
        "roleArn": "arn:aws:iam::123456789012:role/MyStateMachineRole",
        "type": "STANDARD",
        "loggingConfiguration": {
            "level": "OFF",
            "includeExecutionData": false
        },
        "tags": [],
        "tracingConfiguration": {
            "enabled": false
        },
        "publish": false
    },
    "responseElements": {
        "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:MyStateMachine",
        "creationDate": "May 1, 2024 1:23:45 AM"
    },
```

```
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

有关 CloudTrail 录音内容的信息，请参阅《AWS CloudTrail 用户指南》中的CloudTrail录制内容。

使用 CloudWatch Logs 进行日志记录

标准工作流在 AWS Step Functions 中记录执行历史记录，但您也可以选择将其配置为记录到 Amazon CloudWatch Logs 中。

与标准工作流不同，快速工作流不会在 AWS Step Functions 中记录执行历史记录。要查看快速工作流的执行历史记录和结果，必须将日志记录配置为 Amazon CloudWatch Logs。发布日志不会阻止或减慢执行速度。

> **Note**
>
> 配置日志记录时，将收取 CloudWatch Logs 费用，并且将按照公开日志费率计费。有关更多信息，请参阅 CloudWatch 定价页面上日志选项卡下的公开日志。

配置日志记录

使用 Step Functions 控制台创建标准工作流时，不会将其配置为支持记录到 CloudWatch Logs 中。默认情况下，使用 Step Functions 控制台创建的快速工作流会被配置为支持记录到 CloudWatch Logs 中。

对于快速工作流，Step Functions 可以使用必要的 AWS Identity and Access Management (IAM) 策略（针对 CloudWatch Logs）创建角色。如果您使用 API、CLI 或 AWS CloudFormation 创建标准工作流，Step Functions 默认不会启用日志记录，并且您需要确保您的角色具有必要的权限。

对于从控制台启动的每个执行，Step Functions 都会提供一个指向 CloudWatch Logs 的链接，并配置了正确的筛选器，以获取该执行特定的日志事件。

要配置日志记录，可以在使用 CreateStateMachine 或 UpdateStateMachine 时传递 LoggingConfiguration 参数。您可以使用 CloudWatch Logs Insights 进一步分析 CloudWatch Logs 中的数据。有关更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。

CloudWatch Logs 有效负载

执行历史事件的定义中可能包含输入或输出属性。如果发送到 CloudWatch Logs 的转义输入或转义输出超过 248KB，则会因为 CloudWatch Logs 配额而被截断。

- 您可以通过查看 `inputDetails` 和 `outputDetails` 属性来确定有效负载是否已被截断。有关更多信息，请参阅 `HistoryEventExecutionDataDetails` 数据类型。
- 对于标准工作流，您可以使用 `GetExecutionHistory` 查看完整的执行历史记录。
- `GetExecutionHistory` 不适用于快速工作流。如果要查看完整的输入和输出，则可以使用 Amazon S3 ARN。有关更多信息，请参阅the section called “使用 Amazon S3 ARN 而不是传递大量有效负载”。

用于记录到 CloudWatch Logs 的 IAM 策略

您还需要配置状态机的执行 IAM 角色，使其拥有记录到 CloudWatch Logs 的适当权限，如下例所示。

IAM 策略示例

以下是可用于配置权限的示例策略。如以下示例所示，您需要在 `Resource` 字段中指定 *，因为 CloudWatch API 操作（例如 CreateLogDelivery 和 DescribeLogGroups）不支持 Amazon CloudWatch Logs 定义的资源类型。有关更多信息，请参阅 Amazon CloudWatch Logs 定义的操作。

- 有关 CloudWatch 资源的信息，请参阅《Amazon CloudWatch 用户指南》中的 CloudWatch Logs 资源和操作。
- 有关设置向 CloudWatch Logs 发送日志所需权限的信息，请参阅《发送到 CloudWatch Logs 的日志》部分中的用户权限。

```
{
    "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": "*"
        }
    ]
}
```

无法访问 CloudWatch Logs

如果您无法访问 CloudWatch Logs，请确保您已完成以下操作：

1. 配置状态机的执行 IAM 角色，使其拥有向 CloudWatch Logs 记录日志的适当权限。

   如果您使用的是 CreateStateMachine 或 UpdateStateMachine 请求，请确保在包含权限的 `roleArn` 参数中指定了 IAM 角色，如上例所示。
2. 检查 CloudWatch Logs 资源策略，确认未超过 CloudWatch Logs 资源策略的 5120 字符限制。

   如果超过了字符限制，请删除 CloudWatch Logs 资源策略中不必要的权限，或在日志组名称前加上 `/aws/vendedlogs`，这将为日志组授予权限，而无需在资源策略中添加更多字符。在 Step Functions 控制台中创建日志组时，日志组名称的前缀为 `/aws/vendedlogs/states`。有关更多信息，请参阅Amazon CloudWatch Logs 资源策略大小限制。

日志级别

您可以选择 `OFF`、`ALL`、`ERROR` 或 `FATAL`。设置 `OFF` 为时，不会记录任何事件类型；设置为 `ALL` 时，会记录所有事件类型。对于 `ERROR` 和 `FATAL`，请参阅下表。

有关根据这些日志级别为快速工作流执行显示的执行数据的更多信息，请参阅控制台中的标准和快速工作流执行。

事件类型	ALL	ERROR	FATAL	OFF
ChoiceStateEntered	✓			
ChoiceStateExited	✓			
ExecutionAborted	✓	✓	✓	
ExecutionFailed	✓	✓	✓	
ExecutionStarted	✓			
ExecutionSucceeded	✓			
ExecutionTimedOut	✓	✓	✓	
FailStateEntered	✓	✓		
LambdaFunctionFailed	✓	✓		
LambdaFunctionScheduled	✓			
LambdaFunctionScheduleFailed	✓	✓		
LambdaFunctionStarted	✓			
LambdaFunctionStartFailed	✓	✓		

事件类型	ALL	ERROR	FATAL	OFF
LambdaFunctionSucceeded	✓			
LambdaFunctionTimedOut	✓	✓		
MapIterationAborted	✓	✓		
MapIterationFailed	✓	✓		
MapIterationStarted	✓			
MapIterationSucceeded	✓			
MapRunAborted	✓	✓		
MapRunFailed	✓	✓		
MapStateAborted	✓	✓		
MapStateEntered	✓			
MapStateExited	✓			
MapStateFailed	✓	✓		
MapStateStarted	✓			
MapStateSucceeded	✓			

事件类型	ALL	ERROR	FATAL	OFF
ParallelStateAborted	✓	✓		
ParallelStateEntered	✓			
ParallelStateExited	✓			
ParallelStateFailed	✓	✓		
ParallelStateStarted	✓			
ParallelStateSucceeded	✓			
PassStateEntered	✓			
PassStateExited	✓			
SucceedStateEntered	✓			
SucceedStateExited	✓			
TaskFailed	✓	✓		
TaskScheduled	✓			
TaskStarted	✓			
TaskStartFailed	✓	✓		

事件类型	ALL	ERROR	FATAL	OFF
TaskState Aborted	✓	✓		
TaskState Entered	✓			
TaskStateExited	✓			
TaskSubmitFailed	✓	✓		
TaskSubmitted	✓			
TaskSucceeded	✓			
TaskTimedOut	✓	✓		
WaitState Aborted	✓	✓		
WaitState Entered	✓			
WaitStateExited	✓			

AWS X-Ray 和 Step Functions

您可以使用 AWS X-Ray 可视化状态机的组件、确定性能瓶颈以及对导致错误的请求进行故障排除。状态机会向 X-Ray 发送跟踪数据，X-Ray 会处理这些数据，生成服务图和可搜索的跟踪摘要。

为状态机启用 X-Ray 后，您可以跟踪所有可用 X-Ray 的 AWS 区域在 Step Functions 中执行的请求。这样，您就可以详细了解整个 Step Functions 请求的全貌。即使上游服务没有传递跟踪 ID，Step Functions 也会向 X-Ray 发送状态机执行的跟踪。您可以使用 X-Ray 服务地图来查看请求的延迟，包括与 X-Ray 集成的任何 AWS 服务。您也可以配置采样规则，以根据您指定的标准指示 X-Ray 记录哪些请求以及以哪种采样率进行记录。

当状态机未启用 X-Ray 且上游服务未传递跟踪 ID 时，Step Functions 不会向 X-Ray 发送状态机执行的跟踪。但是，如果上游服务传递了跟踪 ID，Step Functions 就会向 X-Ray 发送状态机执行的跟踪。

在同时支持 Step Functions 的区域，你可以 AWS X-Ray 与 Step Functions 一起使用。有关 X-Ray 和 Step Functions 的区域支持信息，请参阅 Step Functions 和 X-Ray 端点和配额页面。

> **X-Ray 和 Step Functions 组合配额**
>
> 您最多可以向跟踪中添加七天的数据，并查询三十天前的跟踪数据，即 X-Ray 存储跟踪数据的时间长度。您的跟踪将受到 X-Ray 配额的限制。除了其他配额外，X-Ray 还为 Step Functions 状态机提供了 100KB 的最小保证跟踪大小。如果向 X-Ray 提供的跟踪数据超过 100KB，可能会导致跟踪被冻结。有关 X-Ray 其他配额的更多信息，请参阅 X-Ray 端点和配额页面的服务限额部分。

> **Important**
>
> Step Functions 不支持对分布式 Map 状态 启动的子工作流执行进行 X-Ray 跟踪，因为此类执行很容易超出跟踪文档大小限制。

主题

- 设置和配置
- 概念
- Step Functions 服务集成和 X-Ray
- 查看 X-Ray 控制台
- 查看 Step Functions 的 X-Ray 跟踪信息
- 跟踪
- 服务图
- 分段和子分段
- 分析
- 配置
- 如果追踪视图或服务图中没有数据怎么办？

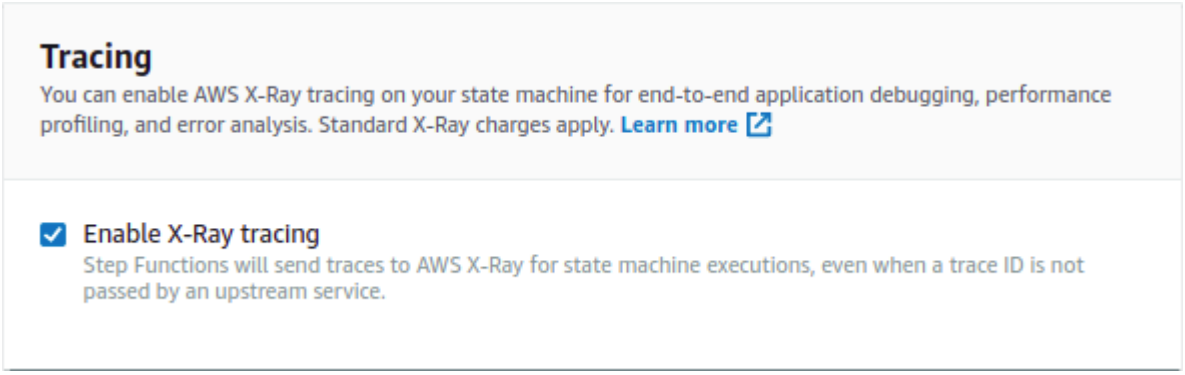
设置和配置

创建状态机时启用 X-Ray 跟踪

通过在指定详细信息页面上选择启用 X-Ray 跟踪，即可在创建新状态机时启用 X-Ray 跟踪。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在选择创作方法页面上，选择适当的选项来创建状态机。如果选择运行示例项目，则无法在创建状态机期间启用 X-Ray 跟踪，并且需要在创建状态机后启用 X-Ray 跟踪。有关在现有状态机中启用 X-Ray 的更多信息，请参阅在现有状态机中启用 X-Ray。

   选择下一步。
3. 在指定详细信息页面，配置状态机。
4. 选择启用 X-Ray 跟踪。



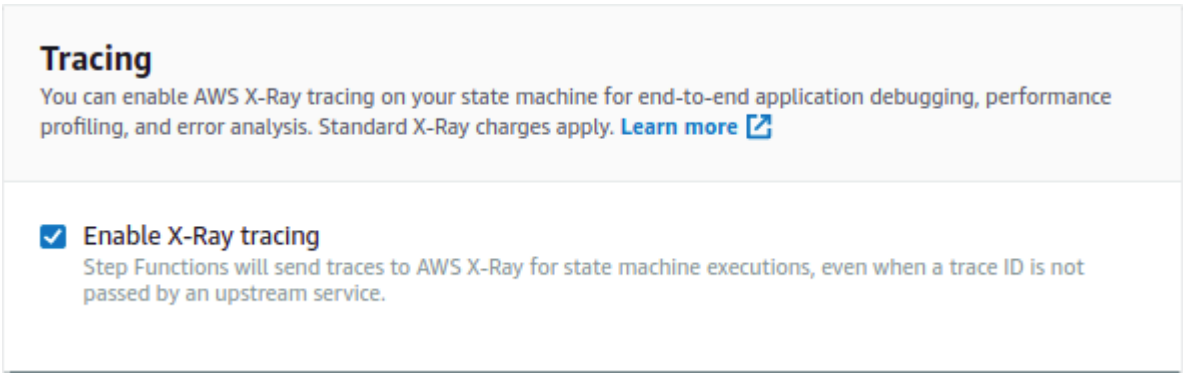
Step Functions 状态机现在将向 X-Ray 发送状态机执行的跟踪。

> **Note**
>
> 如果您选择使用现有 IAM 角色，则应确保允许 X-Ray 写入。有关所需权限的更多信息，请参阅适用于 X-Ray 的 IAM 策略。

在现有状态机中启用 X-Ray

要在现有状态机中启用 X-Ray，请执行以下操作：

1. 在 Step Functions 控制台中，选择要为其启用跟踪的状态机。
2. 选择编辑。
3. 选择启用 X-Ray 跟踪。



您将看到一条通知，告知您可能需要进行其他更改。

> **Note**
>
> 为现有状态机启用 X-Ray 时，必须确保您的 IAM 策略授予了 X-Ray 执行跟踪的足够权限。您可以手动添加一个，也可以生成一个。有关更多信息，请参阅适用于 [适用的 IAM 政策 AWS X-Ray] 的 IAM 策略。

4. （可选）为状态机自动生成新角色，用于包含 X-Ray 权限。
5. 选择保存。

为 Step Functions 配置 X-Ray 跟踪

首次运行启用了 X-Ray 跟踪的状态机时，它将使用默认配置值进行 X-Ray 跟踪。 AWS X-Ray 不会为发送到应用程序的每个请求收集数据。相反，它会为统计上显著数量的请求收集数据。默认情况下，每秒记录第一个请求，并记录所有其他请求的5% 。每秒一个请求是容器。这可确保只要服务正在处理请求，就会每秒至少记录一个跟踪。5% 是对超出容器尺寸的额外请求进行采样的比率。

为避免在您入门时产生服务费用，保守做法是使用默认采样率。您可以配置 X-Ray 以修改默认采样规则并配置基于服务或请求的属性应用采样的其他规则。

例如，您可能想要禁用采样并跟踪所有修改状态或句柄 AWS 账户 或事务的呼叫的请求。对于量非常大的只读调用，例如后台轮询、运行状况检查或连接维护，您采用较低的采样率仍可获取足够的数据来观察出现的任何问题。

要为状态机配置采样规则，请执行以下操作：

1. 前往 X-Ray 控制台。
2. 选择采样。

3. 要创建规则，请选择创建采样规则。

   要编辑规则，请选择规则的名称。

   要删除规则，请选择一条规则并使用操作菜单来删除它。

现有采样规则的某些部分（例如名称和优先级）无法更改。相反，您可以添加或克隆现有规则，进行所需的更改，然后使用新规则。

有关 X-Ray 采样规则以及如何配置各种参数的详细信息，请参阅在 X-Ray 控制台中配置采样规则。

集成上游服务

要将 Step Functions 工作流程（例如快速、同步和标准工作流）的执行与上游服务集成，您需要设置 `traceHeader`。如果您使用的是 API Gateway 中的 HTTP API，则系统会自动为您完成此操作。但是，如果您使用的是 Lambda 函数和/或开发工具包，则需要自己在 `StartExecution` 或 `StartSyncExecution` API 调用中设置 `traceHeader`。

您必须将 `traceHeader` 格式指定为 `\p{ASCII}#`。此外，要让 Step Functions 使用相同的跟踪 ID，必须将格式指定为 `Root={TRACE_ID};Sampled={1 or 0}`。如果您使用的是 Lambda 函数，请将 TRACE_ID 替换为当前分段中的跟踪 ID，如果采样模式为 true，则将“已采样”字段设为 `1`，如果采样模式为 false，则设为 `0`。以这种格式提供跟踪 ID 可确保您获得完整的追踪信息。

下面是使用 Python 编写的示例，展示如何指定 `traceHeader`。

```
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 控制台

通过 AWS X-Ray 控制台，您可以查看应用程序所处理的请求的服务映射和跟踪。当状态机启用 X-Ray 时，您可以访问控制台，查看 X-Ray 收集的详细信息。

有关如何访问状态机执行的 X-Ray 控制台的信息，请参阅查看 X-Ray 控制台。

有关 X-Ray 控制台的详细信息，请参阅 X-Ray 控制台文档。

分段、子分段和跟踪

分段记录有关向状态机发出的请求信息。它包含诸如状态机执行的工作之类的信息，也可能包含有关下游调用信息的子分段。

跟踪会收集单个请求生成的所有分段。

采样

为确保高效跟踪并为应用程序所服务的请求提供代表性样本，X-Ray 应用采样算法来确定跟踪哪些请求。这可以通过编辑采样规则来更改。

指标

对于状态机，X-Ray 将测量调用时间、状态转换时间、Step Functions 的总体执行时间以及执行时间的差异。这些信息可通过 X-Ray 控制台获取。

分析

AWS X-Ray Analytics 控制台是一款用于解释跟踪数据的交互式工具。您可以通过单击与当前跟踪集关联的指标和字段的图表和面板，使用越来越精细的筛选条件细化活动的数据集。这使您可以分析状态机的运行情况，并快速定位和识别性能问题。

有关 X-Ray 分析的详细信息，请参阅与 Analy AWS X-Ray tics 控制台交互

Step Functions 服务集成和 X-Ray

一些与 Step Functions 集成的 AWS 服务 AWS X-Ray 通过向请求添加跟踪标头、运行 X-Ray 守护程序或做出采样决策并将跟踪数据上传到 X-Ray 来提供集成。其他人必须使用 AWS X-Ray SDK 进行检

测。还有一些服务尚未支持 X-Ray 集成。在使用与 Step Functions 集成的服务时，X-Ray 集成是提供完整跟踪数据的必要条件

原生 X-Ray 支持

使用原生 X-Ray 支持的服务集成包括：

- Amazon Simple Notification Service
- Amazon Simple Queue Service
- AWS Lambda
- AWS Step Functions

所需检测

需要 X-Ray 检测的服务集成：

- Amazon Elastic Container Service
- AWS Batch
- AWS Fargate

仅客户端跟踪

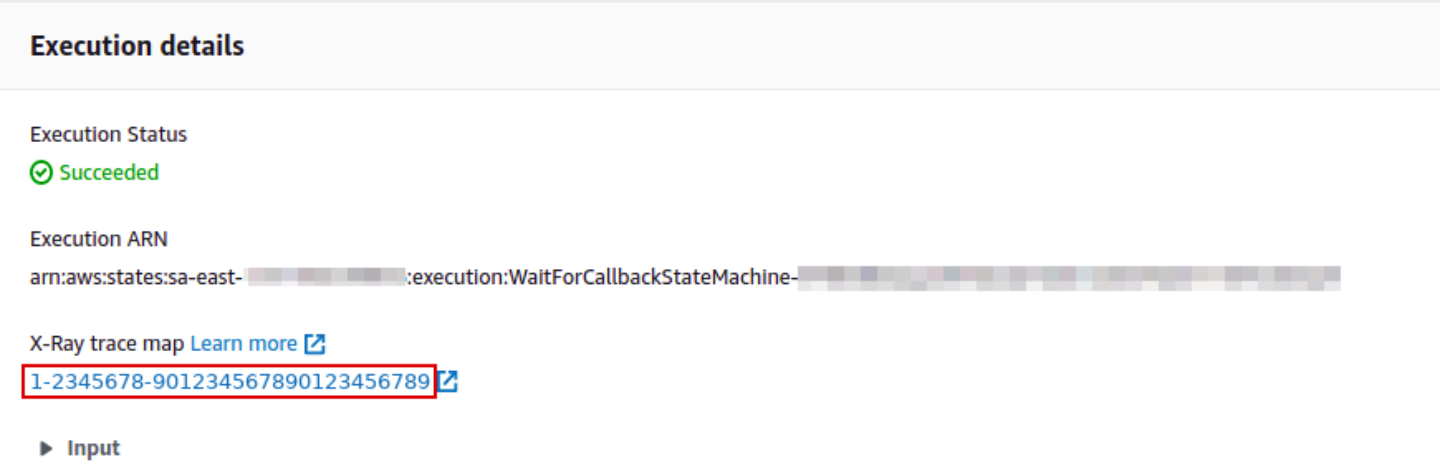
其他服务集成不支持 X-Ray 跟踪。但是，仍然可以收集客户端跟踪：

- Amazon DynamoDB
- Amazon EMR
- Amazon SageMaker
- AWS CodeBuild
- AWS Glue

查看 X-Ray 控制台

X-Ray 从服务以分段形式接收数据。然后，X-Ray 将具有共同请求的分段分组为跟踪。X-Ray 处理跟踪以生成服务图，服务图提供您的应用程序的可视化表示形式。

启动状态机执行后，在执行详细信息部分选择 X-Ray 跟踪视图链接，即可查看其 X-Ray 跟踪。



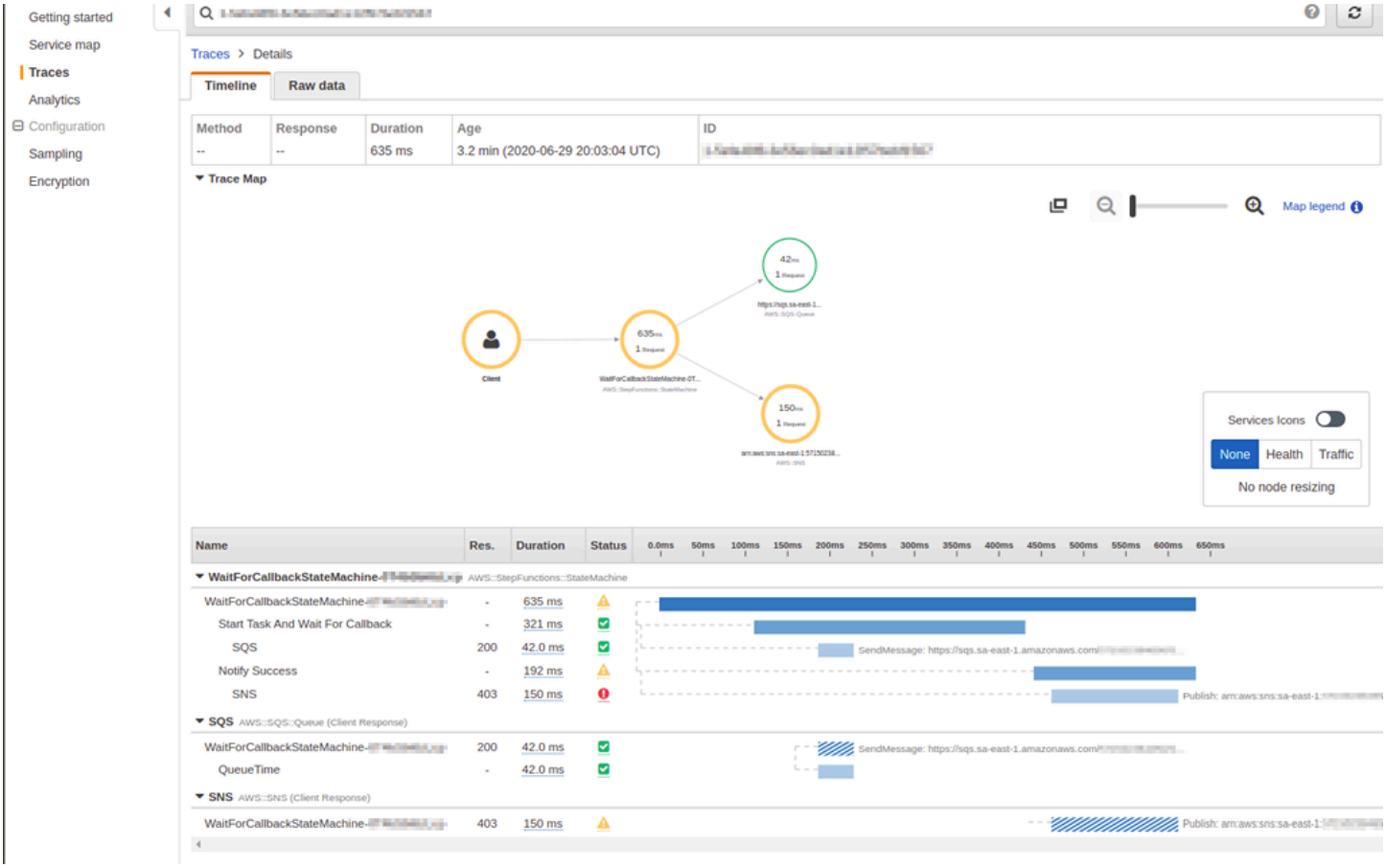
为状态机启用 X-Ray 后，则可以在 X-Ray 控制台中查看其执行跟踪信息。

查看 Step Functions 的 X-Ray 跟踪信息

以下步骤说明启用 X-Ray 并运行执行后，您可以在控制台中看到的信息类型。展示了 回调模式示例（Amazon SQS、Amazon SNS、Lambda） 示例项目的 X-Ray 跟踪。

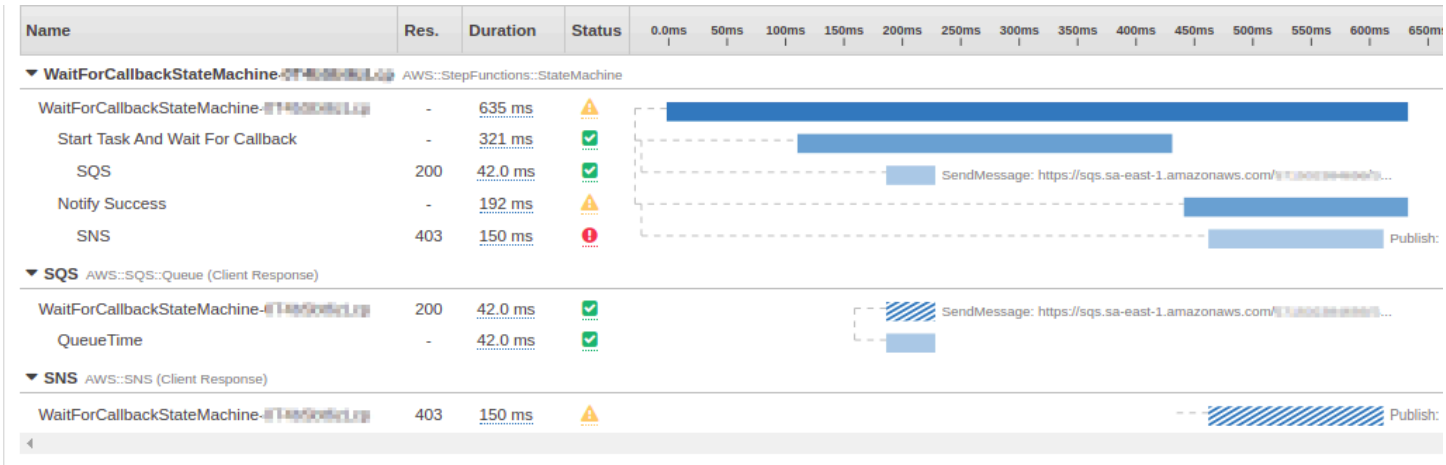
跟踪

执行完成后，您可以导航到 X-Ray 控制台，在在此处可以看到 X-Ray 跟踪页面。这将显示服务图的概述以及状态机的跟踪和分段信息。

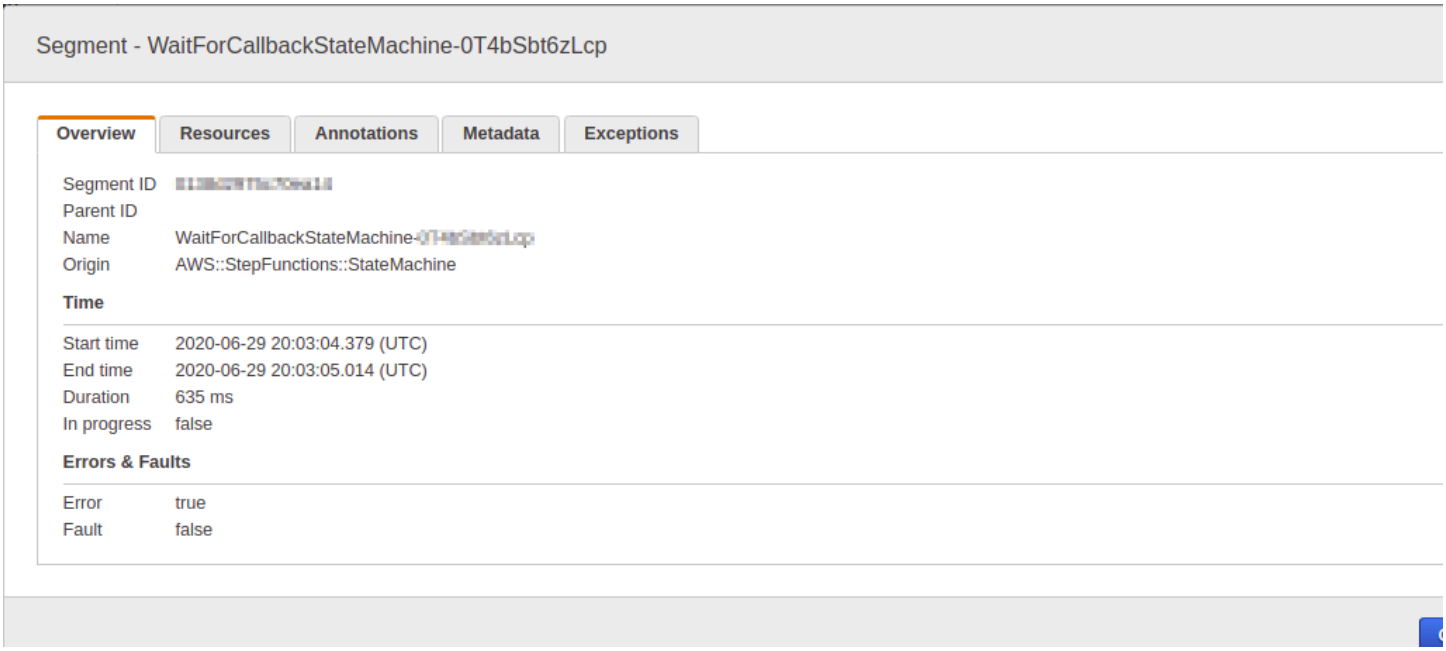


服务图

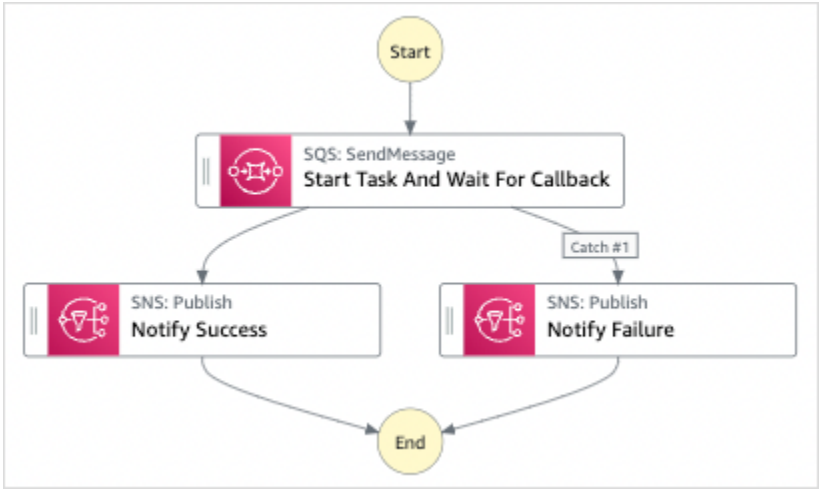
X-Ray 控制台中的服务图可帮助您识别发生错误的服务、存在高延迟连接的服务，或查看请求不成功的跟踪。



在跟踪视图中，您可以选择一个服务节点来查看该节点的请求，或选择两个节点之间的边缘来查看经过该连接的请求。在这里，`WaitForCallBack` 节点已被选中，您可以查看有关其执行和响应状态的其他信息。



您可以看到 X-Ray 服务图如何与状态机相关联。只要支持 X-Ray，Step Functions 调用的每个服务集成都有一个服务图节点。



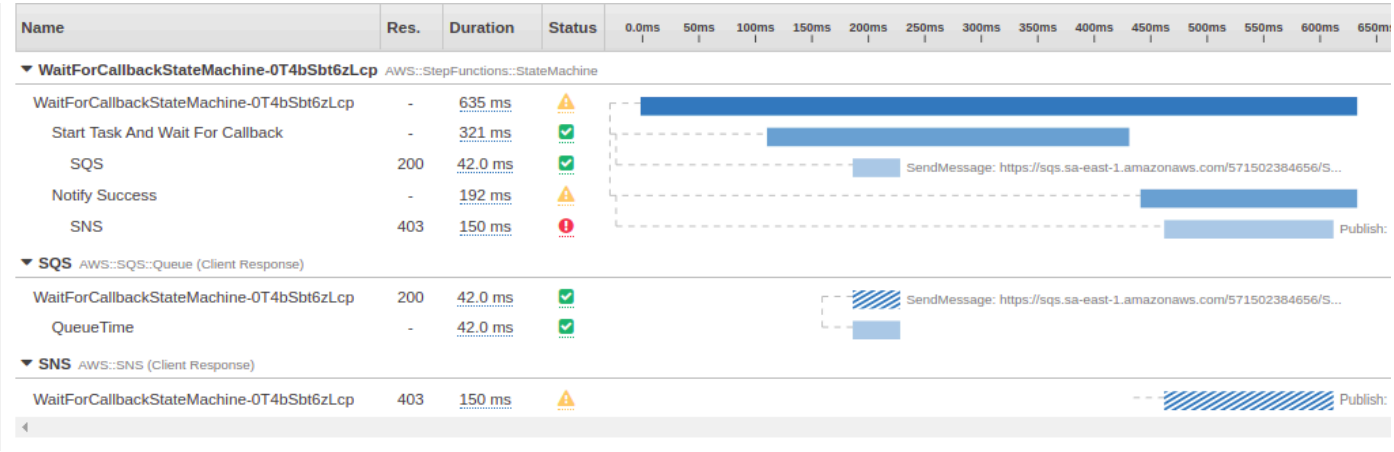
分段和子分段

跟踪是由单个请求生成的片段集合。每个分段提供资源的名称、有关请求的详细信息以及有关所完成工作的详细信息。在跟踪页面上，您可以看到分段，如果展开，还可以看到其相应的子分段。您可以选择一个分段或子分段来查看其详细信息。

选择每个选项卡，查看分段和子分段信息的显示方式。

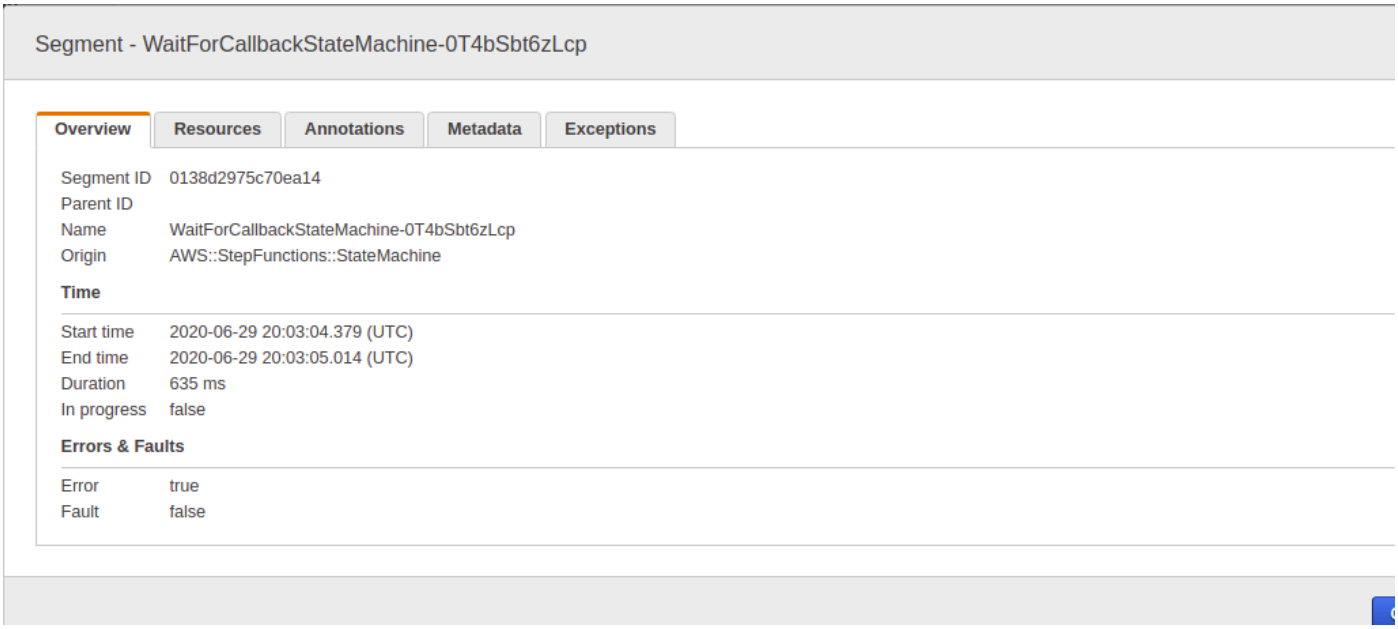
Overview of Segments

此状态机的分段和子分段概览。服务图上的每个节点都有不同的分段。



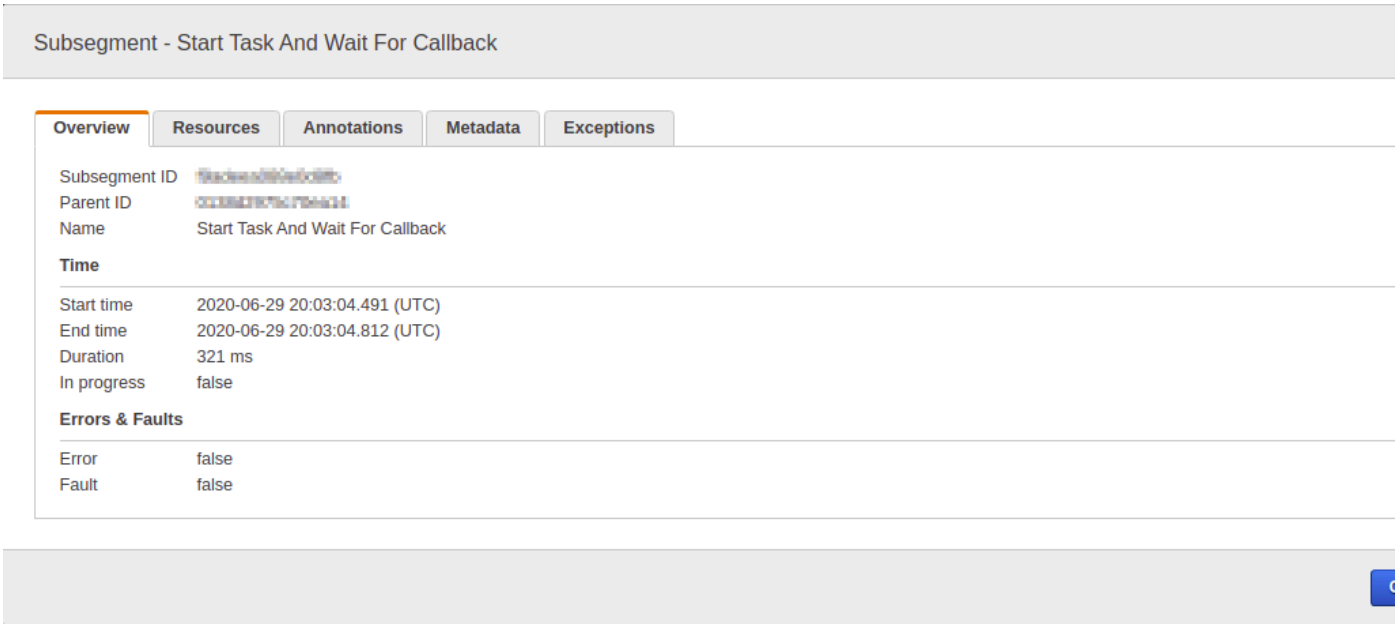
View segment detail

选择分段可提供资源的名称、有关请求的详细信息以及有关所完成工作的详细信息。



View subsegment detail

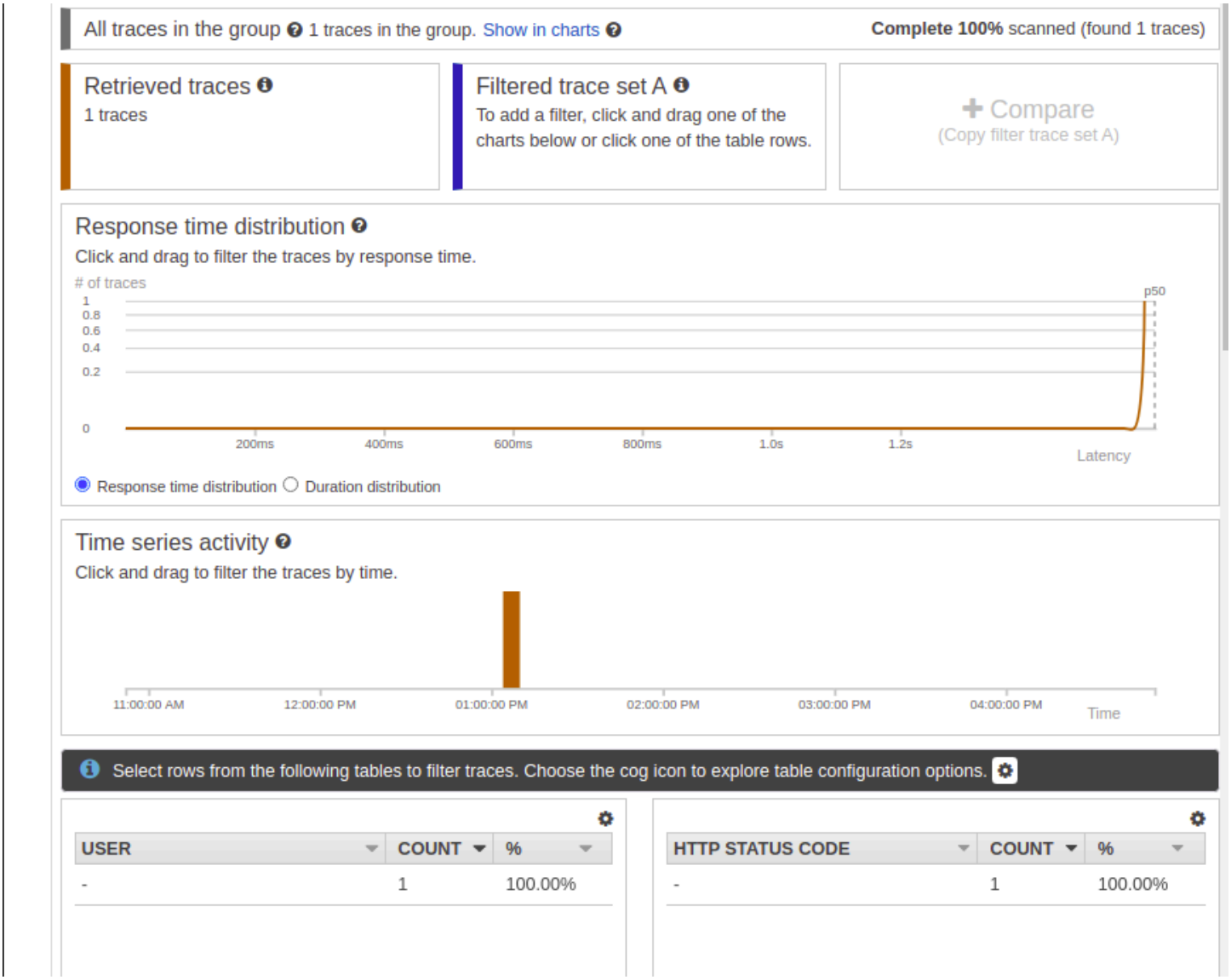
分段可以将关于已完成工作的数据细分为子分段。选择子分段可以查看更精细的时间信息和详细信息。子分段可以包含有关调用 AWS 服务、外部 HTTP API 或 SQL 数据库的其他详细信息。



分析

AWS X-Ray Analytics 控制台是一款用于解释跟踪数据的交互式工具。您可以使用它来更轻松地了解状态机的性能。借助该控制台，您可以通过交互式响应时间图表和时间序列图表探索、分析和直观地显示跟踪。这可以帮助您快速找到性能和延迟问题。

您可以通过单击与当前跟踪集关联的指标和字段的图表和面板，使用越来越精细的筛选条件细化活动的数据集。

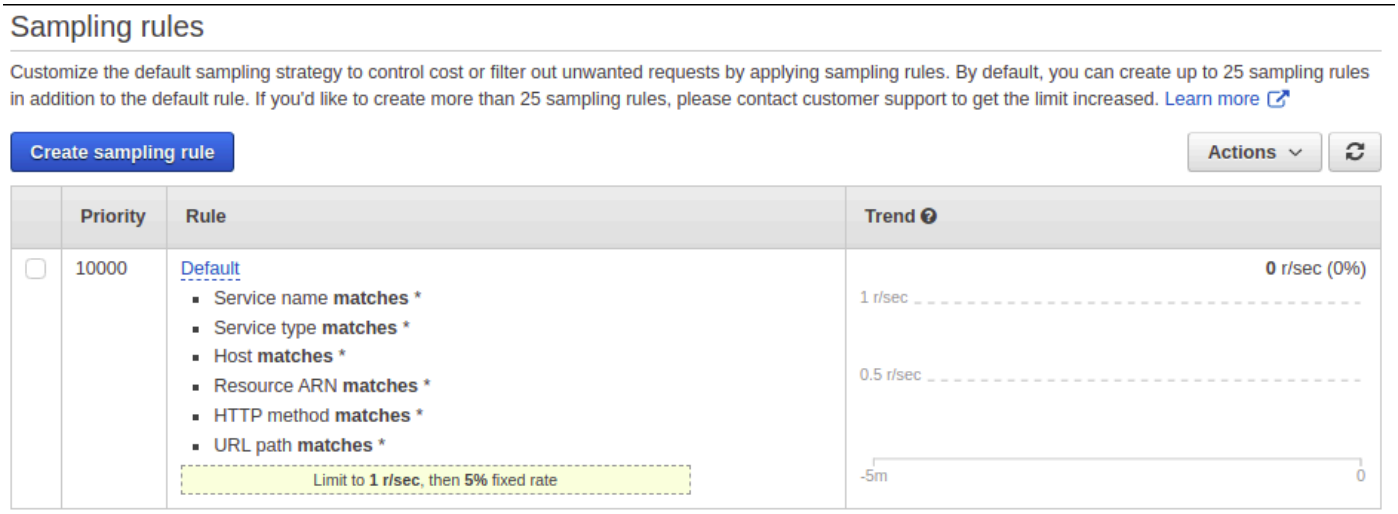


配置

您可以从 X-Ray 控制台配置采样和加密选项。

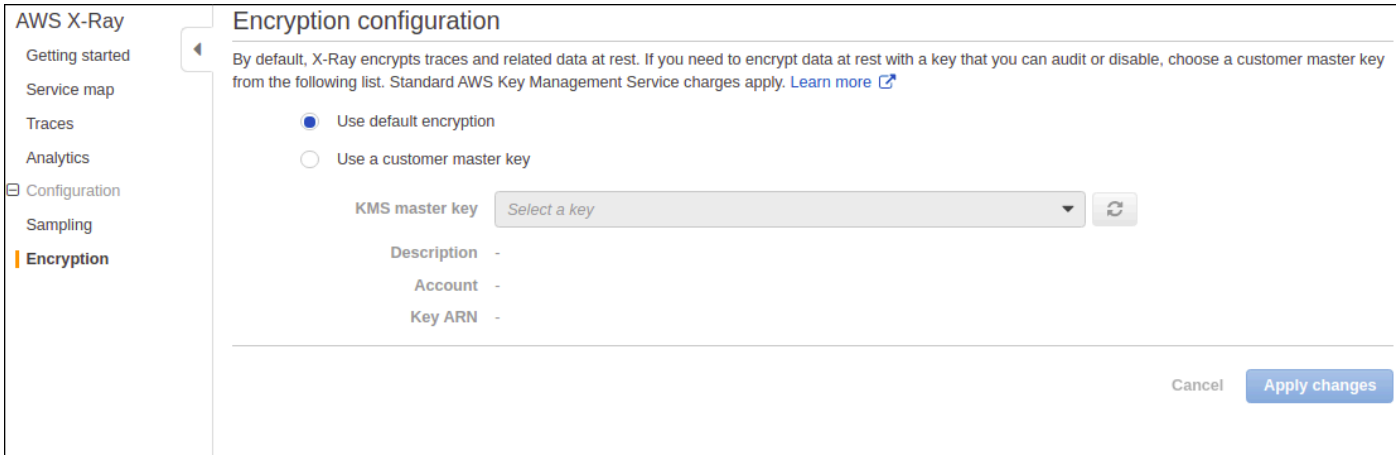
Sampling

选择采样，查看有关采样率和配置的详细信息。您可以更改采样规则来控制记录的数据量，并修改采样行为来满足您的特定要求。



Encryption

选择加密可修改加密设置。您可以使用默认设置，其中 X-Ray 对跟踪和静态日期进行加密，或者，如果需要，您可以选择客户主密钥。标准 AWS KMS 会对后一种情况收费。



如果追踪视图或服务图中没有数据怎么办？

如果您启用了 X-Ray，但在 X-Ray 控制台中看不到任何数据，请检查：

- IAM 角色设置是否正确，允许写入 X-Ray。
- 采样规则是否允许对数据进行采样。
- 由于在应用新创建或修改的 IAM 角色之前可能会有短暂延迟，因此请在几分钟后再次检查跟踪或服务图。

- 如果您在 X-Ray Traces 面板中看到“未找到数据”，请检查您的 IAM 账户设置并确保已 AWS Security Token Service 为目标区域启用该设置。有关更多信息，请参阅 IAM 用户指南 AWS STS AWS 区域中的激活和停用。

配合使用 AWS 用户通知服务和 AWS Step Functions

您可以使用 AWS 用户通知服务来设置交付渠道，以获得有关 AWS Step Functions 事件的通知。当事件与指定的规则匹配时，会收到通知。可以通过多个渠道接收事件通知，包括电子邮件、AWS Chatbot聊天通知或AWS Console Mobile Application推送通知。您还可以在控制台通知中心查看通知。用户通知服务 支持聚合，这可以减少在具体事件期间收到的通知数量。

安全性 AWS Step Functions

本节提供有关 AWS Step Functions 安全和身份验证的信息。

Step Functions 使用 IAM 来控制对其他 AWS 服务和资源的访问。有关 IAM 工作原理的概述，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_access-management.html中的访问管理概述。有关安全凭证的概述，请参阅 中的AWS 安全凭证Amazon Web Services 一般参考。

中的数据保护 AWS Step Functions

分 AWS 担责任模型适用于中的数据保护 AWS Step Functions。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准 (FIPS) 第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括你使用控制台、API 或软件开发工具包 AWS 服务 使用 Step Function AWS s 或其他工具包的情况。 AWS CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

加密 AWS Step Functions

静态加密

Step Functions 始终加密您的静态数据。中的数据使用透明 AWS Step Functions 的服务器端加密进行静态加密。这样可以帮助减少在保护敏感数据时涉及的操作负担和复杂性。通过静态加密，您可以构建符合加密合规性和法规要求的安全敏感型应用程序

传输中加密

Step Functions 加密服务与其他集成 AWS 服务之间的传输中数据（请参阅与其他服务 AWS Step Functions 一起使用）。Step Functions 和集成服务之间传递的所有数据均使用传输层安全性协议 (TLS) 进行加密。

AWS Step Functions中的 Identity and Access Management

访问 AWS Step Functions 需要 AWS 可用于对您的请求进行身份验证的证书。这些证书必须具有访问 AWS 资源的权限，例如从其他 AWS 资源检索事件数据。以下部分提供了有关如何使用 AWS Identity and Access Management (IAM) 和 Step Functions 通过控制访问您的资源的人员以保护这些资源的详细信息：

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 Step Functions 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 Step Functions。

服务用户-如果您使用 Step Functions 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多 Step Functions 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Step Functions中的特征，请参阅 对 AWS Step Functions 身份和访问进行故障排除。

服务管理员-如果您负责公司的 Step Functions 资源，则可能拥有完全访问权限 Step Functions。您的工作是确定您的服务用户应访问哪些 Step Functions 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何将 IAM 与配合使用 Step Functions，请参阅如何 AWS Step Functions 与 IAM 配合使用。

IAM 管理员：如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Step Functions的访问权限的详细信息。要查看您可以在 IAM 中使用的 Step Functions 基于身份的策略示例，请参阅。基于身份的策略示例 AWS Step Functions

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的 多重身份验证和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅 IA M 用户指南中的跨账户资源访问。

- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service (Amazon S3) 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边

界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

访问控制

您可以使用有效的凭证来对自己的请求进行身份验证，但您还必须拥有权限才能创建或访问 Step Functions 资源。例如，您必须有权调用 AWS Lambda与您的 Step Functions 规则关联的亚马逊简单通知服务 (Amazon SNS) 和亚马逊简单队列服务 (Amazon SQS) 目标。

下面几节介绍如何管理 Step Functions 的权限。

- 为状态机创建 IAM 角色
- 为非管理员用户创建精细的 IAM 权限
- 用于 Step Functions 的 Amazon VPC 端点
- 集成服务的 IAM 策略
- 使用分布式 Map 状态的 IAM 策略

的政策行动 Step Functions

支持策略操作	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Step Functions 操作列表，请参阅《服务授权参考》 AWS Step Functions中的 “由定义的资源”。

正在执行的策略操作在操作前 Step Functions 使用以下前缀：

```
states
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "states:action1",
      "states:action2"
         ]
```

要查看 Step Functions 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS Step Functions

的政策资源 Step Functions

支持策略资源	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 `Resource` 或 `NotResource` 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 Step Functions 资源类型及其 ARN 的列表，请参阅《服务授权参考》 AWS Step Functions中定义的操作。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 AWS Step Functions定义的资源。

要查看 Step Functions 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS Step Functions

的策略条件密钥 Step Functions

支持特定于服务的策略条件键	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Step Functions 条件键列表，请参阅《服务授权参考》 AWS Step Functions中的条件密钥。要了解可以使用条件键的操作和资源，请参阅由定义的资源 AWS Step Functions。

要查看 Step Functions 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS Step Functions

输入的 ACL Step Functions

支持 ACL	否

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC with Step Functions

支持 ABAC（策略中的标签）	部分

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证与 Step Functions

支持临时凭证	是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的 切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

的跨服务主体权限 Step Functions

支持转发访问会话 (FAS)	是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详细信息，请参阅转发访问会话。

Step Functions的服务角色

支持服务角色	是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会中断 Step Functions 功能。只有在 Step Functions 提供操作指导时才编辑服务角色。

的服务相关角色 Step Functions

支持服务相关角色	否

服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

如何 AWS Step Functions 与 IAM 配合使用

在使用 IAM 管理访问权限之前 Step Functions，请先了解有哪些 IAM 功能可供使用 Step Functions。

您可以搭配使用的 IAM 功能 AWS Step Functions

IAM 功能	Step Functions 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	否

要全面了解 Step Functions 以及其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

基于身份的策略示例 AWS Step Functions

默认情况下，用户和角色没有创建或修改 Step Functions 资源的权限。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM 策略。

有关由定义的操作和资源类型的详细信息 Step Functions，包括每种资源类型的 ARN 格式，请参阅《服务授权参考》 AWS Step Functions中的操作、资源和条件密钥。

主题

- 策略最佳实践
- 使用 Step Functions 控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 Step Functions 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略 或 工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件 。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 Step Functions 控制台

要访问 AWS Step Functions 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 Step Functions 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 Step Functions 控制台，还需要将 Step Functions *ConsoleAccess*或*ReadOnly* AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

基于身份的策略 Step Functions

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

基于身份的策略示例 Step Functions

要查看 Step Functions 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS Step Functions

内部基于资源的政策 Step Functions

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时

AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅 IAM 用户指南中的跨账户访问 IAM 中的资源。

AWS 的托管策略 AWS Step Functions

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管策略： AWSStepFunctionsConsoleFullAccess

您可以将 AWSStepFunctionsConsoleFullAccess 策略附加到 IAM 身份。

此策略授予###权限，允许用户访问使用 Step Functions 控制台。要获得完整的控制台体验，用户可能还需要该服务可以担任的其他 IAM 角色的 iam: PassRole 权限。

AWS 托管策略： AWSStepFunctionsReadOnlyAccess

您可以将 AWSStepFunctionsReadOnlyAccess 策略附加到 IAM 身份。

此策略授予##权限，允许用户或角色列出和描述状态机、活动、执行、活动 MapRuns、标签以及状态机别名和版本。此策略还授予检查您提供的状态机定义语法的权限。

AWS 托管策略： AWSStepFunctionsFullAccess

您可以将 AWSStepFunctionsFullAccess 策略附加到 IAM 身份。

此策略向用户或角色授予使用 Step Functions API 的##权限。要获得完全访问PassRole权限，用户必须对服务可以担任的至少一个 IAM 角色拥有 iam: 权限。

Step FunctionsAWS 托管策略的更新

查看 Step Functions 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。要获得有关此页面更改的自动提示，请订阅 Step Functions 文档历史记录 页面上的 RSS 源。

更改	描述	日期
AWSStepFunctionsReadOnlyAccess – 对现有策略的更新	Step Functions 添加了新的权限，允许调用 `states:ValidateStateMachineDefinition` API 操作来检查您提供的状态机定义的语法。	2024 年 4 月 25 日
AWSStepFunctionsReadOnlyAccess – 更新了现有策略	Step Functions 添加了新的权限，允许列出和读取与以下内容相关的数据：标签 (ListTagsForResource)、分布式地图（ListMap运行、 DescribeMapRun ）、版本和别名（DescribeStateMachineAlias、ListStateMachineAliases、ListStateMachineVersions ）。	2024年4月2日
Step Functions 已开始跟踪更改	Step Functions 开始跟踪其 AWS 托管策略的更改。	2024年4月2日

为状态机创建 IAM 角色

AWS Step Functions 可以执行代码和访问 AWS 资源（例如调用 AWS Lambda 函数）。为了保持安全性，您必须使用 IAM 角色为 Step Functions 授予对这些资源的访问权限。

本指南Step Functions 教程中的使您可以利用自动生成的 IAM 角色，这些角色对您创建状态机的 AWS 区域有效。但是，您可以为状态机创建自己的 IAM 角色。

在创建状态机使用的 IAM 策略时，策略中应包括您希望状态机承担的权限。您可以使用现有的 AWS 托管策略作为示例，也可以从头开始创建满足您特定需求的自定义策略。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 策略。

要为状态机创建您自己的 IAM 角色，请按照本节中的步骤操作。

在此示例中，您将创建一个有权调用 Lambda 函数的 IAM 角色。

为 Step Functions 创建角色

1. 登录到 IAM 控制台，然后选择 Roles、Create role。
2. 在选择可信实体页面的 AWS 服务下，从列表中选择 Step Functions，然后选择下一步：权限。
3. 在已附加权限政策页面上，选择下一步：审核。
4. 在审核页面上，对于角色名称，输入 `StepFunctionsLambdaRole`，然后选择创建角色。

   IAM 角色显示在角色列表中。

有关 IAM 权限和策略的更多信息，请参阅《IAM 用户指南》中的访问管理。

防止跨服务混淆代理问题

混淆代理问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。这种类型的模拟可能跨账户和跨服务发生。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。

为防止代表感到困惑，我们 AWS 提供了一些工具，可帮助您保护所有服务的数据，这些服务委托人已被授予访问您账户中资源的权限。本节重点介绍针对的跨服务混淆代理预防 AWS Step Functions；但是，您可以在 IAM 用户指南的混乱副手问题部分中了解有关此主题的更多信息。

我们建议在资源策略中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以限制 Step Functions 为其他服务提供的资源访问权限。如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符 (*) 的 `aws:SourceArn` 全局上下文条件键。例如，`arn:aws:states:*:111122223333:*`。

以下是新人策略的示例，它展示了如何将 aws:SourceArn 和 aws:SourceAccount 与 Step Functions 搭配使用来防止出现混乱的副手问题。

```
{
  "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"
           }
        }
     }
  ]
}
```

附加内联策略

Step Functions 可以直接在 Task 状态中控制其他服务。附加内联策略以允许 Step Functions 访问您需要控制的服务的 API 操作。

1. 打开 IAM 控制台，选择角色，搜索您的 Step Functions 角色，然后选择该角色。
2. 选择添加内联策略。
3. 使用可视化编辑器或 JSON 选项卡为您的角色创建策略。

有关 AWS Step Functions 如何控制其他 AWS 服务的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

> **Note**
>
> 有关 Step Functions 控制台创建的 IAM 策略的示例，请参阅集成服务的 IAM 策略。

为非管理员用户创建精细的 IAM 权限

IAM 中的默认托管策略（例如`ReadOnly`）并不能完全涵盖所有类型的 AWS Step Functions 权限。本节将介绍这些不同类型的权限并提供一些示例配置。

Step Functions 具有四种类别的权限。根据您要为用户提供的访问权限，您可以使用这些类别的权限控制访问权限。

服务级别权限

适用于不作用于特定资源的 API 组件。

状态机级权限

适用于对特定状态机执行的所有 API 组件。

执行级权限

适用于对特定执行采用的所有 API 组件。

活动级权限

适用于对特定活动或活动的特殊实例执行的所有 API 组件。

服务级别权限

此权限级别适用于不作用于特定资源的所有 API 操作。这些包括`CreateStateMachine`、`CreateActivityListStateMachines`、`ListActivities`、和`ValidationStateMachineDefinition`。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:ListStateMachines",
        "states:ListActivities",
```

```
        "states:CreateStateMachine",
        "states:CreateActivity",
        "states:ValidationStateMachineDefinition",
      ],
      "Resource": [
        "arn:aws:states:*:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam:::role/my-execution-role"
      ]
    }
  ]
}
```

状态机级权限

此权限级别适用于对特定状态机执行的所有 API 操作。这些 API 操作要求状态机的 Amazon 资源名称 (ARN) 作为请求的一部分，例如 DeleteStateMachine、DescribeStateMachine、StartExecution 和 ListExecutions。

```
{
  "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*"
      ]
```

```
    }
  ]
}
```

执行级权限

此权限级别适用于对特定执行采用的所有 API 操作。这些 API 操作要求执行的 ARN 作为请求的一部分，例如 DescribeExecution、GetExecutionHistory 和 StopExecution。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:DescribeStateMachineForExecution",
        "states:GetExecutionHistory",
        "states:StopExecution"
      ],
      "Resource": [
        "arn:aws:states:*:*:execution:*:ExecutionPrefix*"
      ]
    }
  ]
}
```

活动级权限

此权限级别适用于将对特定活动或活动的特殊实例执行的所有 API 操作。
这些 API 操作要求活动的 ARN 或实例的令牌作为请求的一部分，例如 DeleteActivity、DescribeActivity、GetActivityTask 和 SendTaskHeartbeat。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeActivity",
        "states:DeleteActivity",
        "states:GetActivityTask",
```

```
        "states:SendTaskHeartbeat"
      ],
      "Resource": [
        "arn:aws:states:*:*:activity:ActivityPrefix*"
      ]
    }
  ]
}
```

在工作流程中访问其他资源 AWS 账户 中的资源

Step Functions 提供对工作流程 AWS 账户 中不同配置的资源的跨账户访问权限。使用 Step Functions 服务集成，您可以调用任何跨账户 AWS 资源，即使该资源 AWS 服务 不支持基于资源的策略或跨账户调用。

例如，假设您拥有两个 AWS 账户名为 “开发和测试” 的同一个项目 AWS 区域。使用跨账户存取，“开发”账户中的工作流可以访问“测试”账户中的资源，如 Amazon S3 存储桶、Amazon DynamoDB 表和 Lambda 函数。

> **Important**
>
> IAM 角色和基于资源的策略仅在单个分区内跨账户委派访问权限。例如，假定您在标准 aws 分区的美国西部（加利福尼亚北部）中有一个账户。您在 aws-cn 分区的中国（北京）中也有一个账户。您不能使用中国（北京）的账户中 Amazon S3 基于资源的策略，来允许标准 aws 账户中用户的访问权限。

有关跨账户存取的更多信息，请参阅《IAM 用户指南》中的跨账户策略评估逻辑。

尽管每个 AWS 账户 步骤都可以完全控制自己的资源，但使用 Step Functions，您无需自定义任何代码即可重新组织、交换、添加或删除工作流程中的步骤。即使流程发生变化或应用程序不断发展，您也可以这样做。

您还可以调用嵌套状态机的执行，这样它们就可以在不同的账户中使用。这样做可以有效分离和隔离您的工作流。当您在工作流中使用 `.sync` 服务集成模式访问不同账户中的另一个 Step Functions 工作流时，Step Functions 会使用轮询来消耗您分配的配额。有关更多信息，请参阅 运行作业 (.sync)。

> **Note**
>
> 目前，跨区域 AWS SDK 集成和跨区域 AWS 资源访问在 Step Functions 中不可用。

内容

- 本主题中的关键概念
- 调用跨账户资源
- 教程：访问跨账户资源 AWS
- 跨账户存取 .sync 集成模式

本主题中的关键概念

执行角色

Step Functions 用来运行代码和访问 AWS 资源的 IAM 角色，例如 AWS Lambda 函数的 “调用” 操作。

服务集成

可以在工作流程的某个Task状态内调用的 AWS SDK 集成 API 操作。

源账户

AWS 账户 拥有状态机并已开始执行的。

目标账户

您可以 AWS 账户 向其拨打跨账户的电话。

目标角色

目标账户中的 IAM 角色，状态机在调用目标账户所拥有的资源时会假定该角色。

运行作业 (.sync)

一种用于调用服务的服务集成模式，例如 AWS Batch。它还能使 Step Functions 状态机在进入下一状态前等待作业完成。要指示 Step Functions 应该等待，请在 `Task` 状态定义的 `Resource` 字段中附加 `.sync` 后缀。

调用跨账户资源

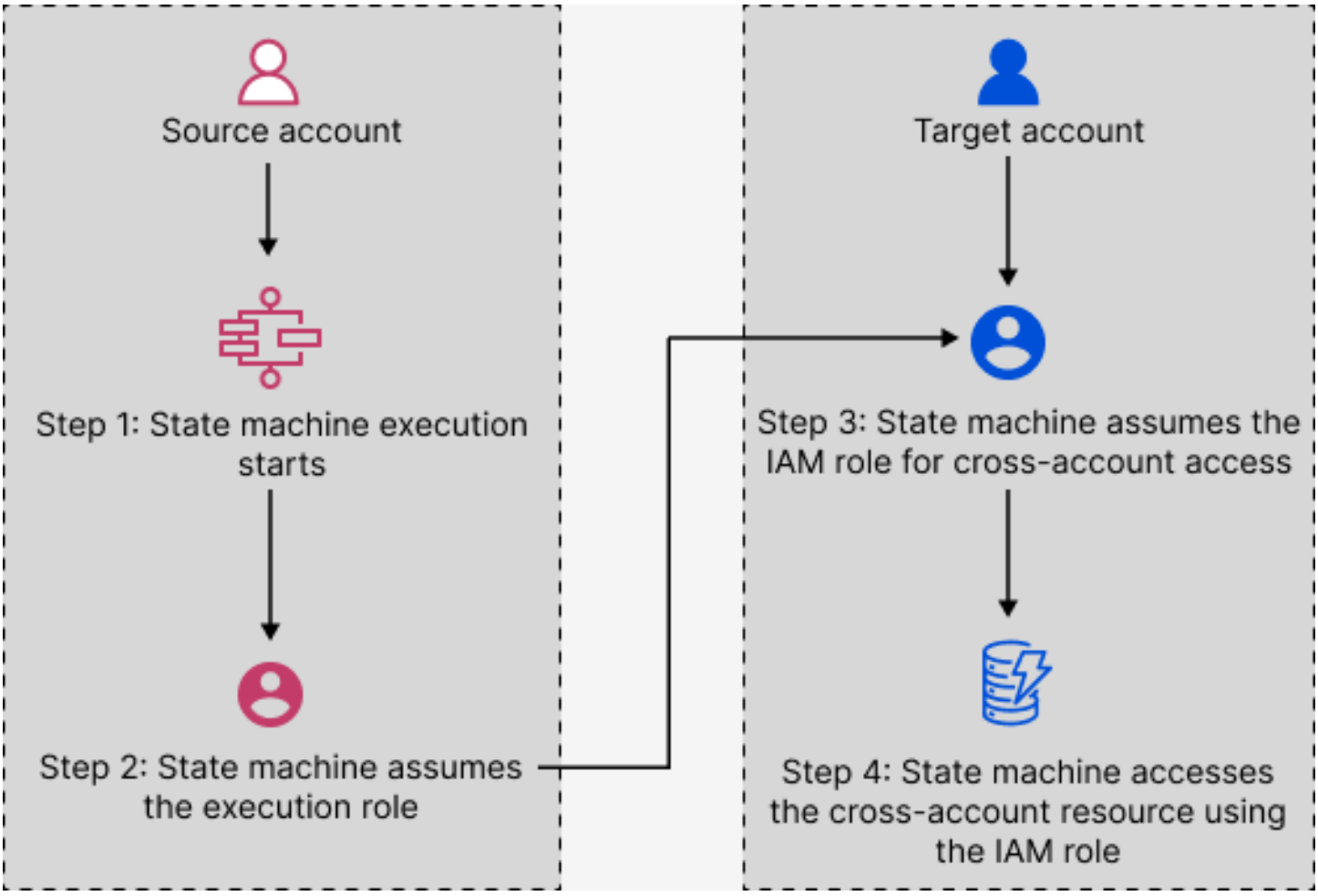
要在工作流中调用跨账户资源，请执行以下操作：

1. 在包含资源的目标账户中创建 IAM 角色。此角色向包含状态机的源账户授予访问目标账户资源的权限。

2. 在 Task 状态的定义中，指定在调用跨账户资源之前由状态机担任的目标 IAM 角色。
3. 修改目标 IAM 角色中的信任策略来允许源账户临时担任此角色。信任策略必须包含源账户中定义的状态机的 Amazon 资源名称（ARN）。此外，还要在目标 IAM 角色中定义调用 AWS 资源的相应权限。
4. 更新源账户的执行角色，使其包含担任目标 IAM 角色所需的权限。

有关示例，请参阅教程：访问跨账户资源 AWS。

> **Note**
>
> 您可以配置状态机承担一个 IAM 角色，以便从多个 AWS 账户访问资源。但是，状态机在给定时间只能承担一个 IAM 角色。



教程：访问跨账户资源 AWS

借助 Step Functions 中的跨账户存取支持，您可以共享不同 AWS 账户配置的资源。在本教程中，我们将引导您完成访问在名为生产的账户中定义的跨账户 Lambda 函数的过程。该函数从名为开发的账户中的状态机调用。在本教程中，开发账户被称为源账户，而生产账户是包含目标 IAM 角色的目标账户。

首先，在 Task 状态定义中，指定状态机在调用跨账户 Lambda 函数前必须承担的目标 IAM 角色。然后，修改目标 IAM 角色中的信任策略，允许源账户临时承担目标角色。此外，要调用 AWS 资源，请在目标 IAM 角色中定义相应的权限。最后，更新源账户的执行角色，指定担任目标角色所需的权限。

您可以配置状态机承担一个 IAM 角色，以便从多个 AWS 账户访问资源。但是，根据 Task 状态的定义，状态机在给定时间只能承担一个 IAM 角色。

> **Note**
>
> 目前，跨区域 AWS SDK 集成和跨区域 AWS 资源访问在 Step Functions 中不可用。

内容

- 先决条件
- 第 1 步：更新 Task 状态定义以指定目标角色
- 第 2 步：更新目标角色的信任策略
- 第 3 步：将所需权限添加到目标角色
- 第 4 步：在执行角色中添加承担目标角色的权限

先决条件

- 本教程以 Lambda 函数为例，演示如何设置跨账户存取。您可以使用任何其他 AWS 资源，但请确保已在其他账户中配置该资源。

> **Important**
>
> IAM 角色和基于资源的策略仅在单个分区内跨账户委派访问权限。例如，假定您在标准 aws 分区的美国西部（加利福尼亚北部）中有一个账户。您在 aws-cn 分区的中国（北京）中也有一个账户。您不能使用中国（北京）的账户中 Amazon S3 基于资源的策略，来允许标准 aws 账户中用户的访问权限。

- 在文本文件中记下跨账户资源的 Amazon 资源名称 (ARN)。本教程稍后将在状态机的 Task 状态定义中提供此 ARN。以下是 Lambda 函数 ARN 的示例：

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

- 确保您已创建状态机需要承担的目标 IAM 角色。

第 1 步：更新 Task 状态定义以指定目标角色

在工作流的 Task 状态下，添加一个 Credentials 字段，其中包含状态机在调用跨账户 Lambda 函数之前必须承担的身份。

以下过程演示了如何访问名为 Echo 的跨账户 Lambda 函数。您可以按照以下步骤调用任何 AWS 资源。

1. 打开 Step Functions 控制台，然后选择创建状态机。
2. 在选择创作方法页面上，选择直观地设计工作流，并保留所有默认选择。
3. 要打开 Workflow Studio，请选择下一步。
4. 在操作选项卡上，将一个 Task 状态拖放到画布上。这将调用使用此 Task 状态的跨账户 Lambda 函数。
5. 在配置页面上，执行以下操作：

   a. 将状态重命名为 **Cross-account call**。

   b. 对于函数名称，选择输入函数名称，然后在框中输入 Lambda 函数 ARN。例如，arn:aws:lambda:us-east-2:111122223333:function:*Echo*。

   c. 对于提供 IAM 角色 ARN，指定目标 IAM 角色 ARN。例如，arn:aws:iam::111122223333:role/LambdaRole。

   > **Tip**
   >
   > 或者，您也可以在包含 IAM 角色 ARN 的状态 JSON 输入中指定现有键值对的参考路径。为此，选择从状态输入中获取运行时的 IAM 角色 ARN。有关使用引用路径指定值的示例，请参见指定 JSONPath 为 IAM 角色 ARN。

6. 选择下一步。
7. 在检查生成的代码页面上，选择下一步。
8. 在 指定状态机设置页面上，指定新状态机的详细信息，例如名称、权限和日志级别。

9. 选择创建状态机。
10. 在文本文件中记下状态机的 IAM 角色 ARN 和状态机 ARN。您需要在目标账户的信任策略中提供这些 ARN。

现在，您的 Task 状态定义应类似于以下定义。

```
{
  "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
    }
  }
}
```

第 2 步：更新目标角色的信任策略

IAM 角色必须存在于目标账户中，并且您必须修改其信任策略以允许源账户暂时承担此角色。此外，您可以控制谁能够担任目标 IAM 角色。

创建信任关系后，源账户中的用户可以使用 AWS Security Token Service (AWS STS) AssumeRoleAPI 操作。此操作提供临时安全证书，允许访问目标账户中的 AWS 资源。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在控制台的导航窗格上，选择角色，然后使用搜索框搜索目标 IAM 角色。例如，*LambdaRole*。
3. 选择信任关系选项卡。
4. 选择编辑信任策略并粘贴以下信任策略。请务必替换 AWS 账户 号码和 IAM 角色 ARN。`sts:ExternalId` 字段进一步控制谁可担任该角色。状态机的名称必须仅包含 AWS Security Token Service `AssumeRole` API 支持的字符。有关更多信息，请参阅 AWS Security Token Service API 参考AssumeRole中的。

```
{
  "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. 保持此窗口处于打开状态，然后继续阅读下一个步骤以进行后续操作。

第 3 步：将所需权限添加到目标角色

IAM 策略中的权限确定是允许还是拒绝特定请求。目标 IAM 角色必须具有调用 Lambda 函数的正确权限。

1. 选择权限选项卡。
2. 选择添加权限，然后选择创建内联策略。
3. 选择 JSON 选项卡，并使用以下权限替换现有内容。请务必替换您的 Lambda 函数 ARN。

```
{
  "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. 选择查看策略。
5. 在检查策略页面上，输入一个权限名称，然后选择创建策略。

第 4 步：在执行角色中添加承担目标角色的权限

Step Functions 不会自动为所有跨账户服务集成生成AssumeRole政策。您必须在状态机的执行角色中添加所需的权限，使其能够在一个或多个 AWS 账户中承担目标 IAM 角色。

1. 访问 https://console.aws.amazon.com/iam/，在 IAM 控制台中打开状态机执行角色。要实现此目的，应按照以下步骤进行：
   a. 在源账户中打开第 1 步中创建的状态机。
   b. 在状态机详细信息页面上，选择 IAM 角色 ARN。
2. 在权限选项卡上，选择添加权限，然后选择创建内联策略。
3. 选择 JSON 选项卡，并使用以下权限替换现有内容。请务必替换您的 Lambda 函数 ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/LambdaRole"  // The target role
 to be assumed
    }
  ]
}
```

4. 选择查看策略。
5. 在检查策略页面上，输入一个权限名称，然后选择创建策略。

跨账户存取 .sync 集成模式

在工作流中使用 .sync 服务集成模式时，Step Functions 会对调用的跨账户资源进行轮询，以确认任务是否完成。这会导致实际任务完成时间与 Step Functions 确认任务完成时间之间出现轻微延迟。目标 IAM 角色需要 .sync 调用所需的权限来完成轮询循环。为此，目标 IAM 角色必须拥有允许源账户承担的信任策略。此外，目标 IAM 角色还需要完成轮询循环所需的权限。

> **Note**
>
> 嵌套快速工作流当前尚不支持 arn:aws:states:::states:startExecution.sync。请改用 arn:aws:states:::aws-sdk:sfn:startSyncExecution。

.sync 调用的信任策略更新

更新目标 IAM 角色的信任策略，如下面的示例所示。sts:ExternalId 字段进一步控制谁可担任该角色。状态机的名称必须仅包含 AWS Security Token Service AssumeRole API 支持的字符。有关更多信息，请参阅 AWS Security Token Service API 参考AssumeRole中的。

```
{
  "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 调用所需的权限

要授予状态机所需的权限，请更新目标 IAM 角色所需的权限。有关更多信息，请参阅 the section called “集成服务的 IAM 策略”。不需要示例策略中的 Amazon EventBridge 权限。例如，要启动状态机，请添加以下权限。

```
{
  "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:*"
      ]
    }
  ]
}
```

用于 Step Functions 的 Amazon VPC 端点

如果您使用亚马逊虚拟私有云（Amazon VPC）托管 AWS 资源，则可以在您的亚马逊 VPC 和 AWS Step Functions 工作流程之间建立连接。您可以将此连接用于您的 Step Functions 工作流，而无需穿越公共 Internet。标准工作流、快速工作流和同步快速工作流都支持 Amazon VPC 端点。

Amazon VPC 允许您在自定义虚拟网络中启动 AWS 资源。可以使用 VPC 控制您的网络设置，例如 IP 地址范围、子网、路由表和网络网关。有关 VPC 的更多信息，请参阅《Amazon VPC 用户指南》。

要将您的 Amazon VPC 连接到 Step Functions，您必须首先定义一个接口 VPC 终端节点，该终端节点允许您将您的 VPC 与其他 AWS 服务连接起来。该端点提供了可靠且可扩展的连接，无需互联网网

关、网络地址转换 (NAT) 实例或 VPN 连接。有关更多信息，请参阅 Amazon VPC 用户指南中的接口 VPC 端点 (AWS PrivateLink)。

创建端点

您可以使用、 AWS Command Line Interface (AWS CLI)、 AWS 软件开发工具包 AWS Management Console、 AWS Step Functions API 或在 VPC 中创建 AWS Step Functions 终端节点 AWS CloudFormation。

有关使用 Amazon VPC 控制台或 AWS CLI创建和配置端点的信息，请参阅 Amazon VPC 用户指南中的创建接口端点。

> **Note**
>
> 在创建端点时，请将 Step Functions 指定为您希望 VPC 连接到的服务。在 Amazon VPC 控制台中，服务名称因 AWS 地区而异。例如，如果您选择美国东部（弗吉尼亚州北部），则标准工作流和快速工作流的服务名称为 com.amazonaws.us-east-1.state，同步快速工作流的服务名称为 com.amazonaws.us-east-1.sync-states。

> **Note**
>
> 无需通过私有 DNS 覆盖开发工具包中的端点即可使用 VPC 端点。但是，如果要覆盖同步快速工作流的开发工具包中的端点，则需要将 `DisableHostPrefixInjection` 配置设置为 `true`。示例（Java 开发工具包 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();
> ```

有关使用创建和配置终端节点的信息 AWS CloudFormation，请参阅用户指南中的 AWS:: EC2:: vpcendPoint 资源。AWS CloudFormation

Amazon VPC 端点策略

要控制对 Step Functions 的连接访问权限，您可以在创建亚马逊 VPC 终端节点时附加 AWS Identity and Access Management (IAM) 终端节点策略。您可以通过附加多个端点策略来创建复杂的 IAM 规则。有关更多信息，请参阅：

- Step Functions 的 Amazon Virtual Private Cloud 端点策略
- 为非管理员用户创建精细的 IAM 权限
- 使用 VPC 端点控制对服务的访问

Step Functions 的 Amazon Virtual Private Cloud 端点策略

您可以为 Step Functions 创建 Amazon VPC 端点策略，并在其中指定以下内容：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

以下示例显示了一个 Amazon VPC 端点策略，该策略允许一个用户创建状态机，并拒绝所有其他用户删除状态机的权限。示例策略还授予所有 用户执行权限。

```
{
    "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": "*"
        }
    ]
}
```

有关创建端点策略的更多信息，请参阅以下内容：

- 为非管理员用户创建精细的 IAM 权限
- 使用 VPC 端点控制对服务的访问

集成服务的 IAM 策略

在 AWS Step Functions 控制台中创建状态机时，Step Functions 会根据状态机定义中使用的资源生成一个 AWS Identity and Access Management (IAM) 策略，如下所示：

- 如果您的状态机使用其中一个经过优化的集成，Step Functions 将为您的状态机创建一个包含必要权限和角色的策略。（例外： MediaConvert 集成需要您手动设置权限 — 请参阅适用的 IAM 政策 AWS Elemental MediaConvert。）
- 如果您的状态机使用其中一个 AWS SDK 集成，则将创建一个具有部分权限的 IAM 角色。之后，您可以使用 IAM 控制台添加任何缺失的角色策略。

以下示例显示 Step Functions 如何根据您的状态机定义生成 IAM 策略。示例代码中的项目（例如 *[[resourceName]]*）将替换为状态机定义中列出的静态资源。如果您有多个静态资源，则在 IAM 角色中每个资源都将有一个条目。

动态资源与静态资源

静态资源直接在状态机的任务状态中定义。在任务状态中包含要直接调用的资源信息时，Step Functions 只为这些资源创建 IAM 角色。

动态资源是传递到您的状态输入并使用路径访问的资源（请参阅路径）。如果要将动态资源传递给任务，Step Functions 将创建一个具有更高特权的策略，该策略指定 `"Resource": "*"`。

使用“运行作业”模式执的任务附加权限

对于使用运行作业模式（以 .sync 结尾的模式），需要额外的权限来监视和接收来自所连接服务的 API 操作的响应。与使用“请求响应”或“等待回调”模式的任务相比，相关策略包含的权限更多。有关同步任务的信息，请参阅 服务集成模式。

> **Note**
>
> 您需要为支持 Run a Job (.sync) 模式的服务集成提供额外权限。

当作业在连接的服务上运行时，Step Functions 使用两种方法来监控作业的状态，即轮询和事件。

轮询需要 `Describe` 或 `Get` API 操作的权限，例如 `ecs:DescribeTasks` 或 `glue:GetJobRun`。如果您的角色中缺少这些权限，则 Step Functions 可能无法确定您的作业状态。这是因为某些 Run a Job (.sync) 服务集成不支持 EventBridge事件，而有些服务只会尽力发送事件。

从 AWS 服务发送到亚马逊 EventBridge 的事件使用托管规则定向到 Step Functions，并且需要`events:PutTargets`、`events:PutRule`和的权限`events:DescribeRule`。如果您的角色中缺少这些权限，则在 Step Functions 得知您的任务已完成之前，可能会有一段延迟。有关 EventBridge 事件的更多信息，请参阅来自 AWS 服务的事件。

> **Note**
>
> 对于同时支持轮询和事件的“运行任务” (.sync)任务，您的任务仍可使用事件正常完成。即使您的角色缺少轮询所需的权限，也可能发生这种情况。在这种情况下，您可能不会立即注意到轮询权限不正确或缺失。在极少数实例下，事件无法传送到 Step Functions 或由 Step Functions 处理，您的执行可能会卡住。要验证您的轮询权限配置是否正确，您可以通过以下方式在没有 EventBridge 事件的环境中运行执行：
>
> - 从中删除托管规则 EventBridge，该规则负责将事件转发到 Step Functions。该托管规则由账户中的所有状态机共享，因此应仅在测试或开发账户中执行此操作，避免对其他状态机造成任何意外影响。通过检查目标服务策略模板中用于 `events:PutRule` 的 `Resource` 字段，可以确定要删除的特定托管规则。下次创建或更新使用该服务集成的状态机时，将重新创建托管规则。有关删除 EventBridge 规则的更多信息，请参阅禁用或删除规则。
> - 使用 Step Functions Local 进行测试，它不支持使用事件来完成运行作业 (.sync) 任务。要使用 Step Functions Local，请假设状态机使用的 IAM 角色。您可能需要编辑信任关系。将 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN` 环境变量设置为假定角色的值，然后使用 java -jar StepFunctionsLocal.jar 启动 Step Functions

Local。最后，使用 AWS CLI 带--endpoint-url参数的状态机创建状态机、开始执行并获取执行历史记录。有关更多信息，请参阅 在本地测试状态机。

如果使用运行作业 (.sync) 模式的任务停止，Step Functions 会尽力尝试取消该任务。这需要 `Cancel`、`Stop`、`Terminate` 或 `Delete` API 操作的权限，例如 `batch:TerminateJob` 或 `eks:DeleteCluster`。如果您的角色中缺少这些权限，Step Functions 将无法取消您的任务，而且当任务继续运行时，可能会产生额外的费用。有关停止任务的更多信息，请参阅运行任务。

用于创建 IAM 角色的策略模板

以下主题包括您使用 Step Functions 为您创建新角色时所用的策略模板。

> **Note**
>
> 查看这些模板，了解 Step Functions 是如何创建您的 IAM 策略的，并举例说明在使用其他 AWS 服务时如何为 Step Functions 手动创建 IAM 策略。有关 Step Functions 服务集成的更多信息，请参阅与其他服务 AWS Step Functions 一起使用。

主题

- 亚马逊 API Gateway 的 IAM 政策
- 亚马逊 Athena 的 IAM 政策
- 适用的 IAM 政策 AWS Batch
- IAM policies for Amazon Bedrock
- 适用的 IAM 政策 AWS CodeBuild
- 亚马逊 DynamoDB 的 IAM 政策
- 适用于 Amazon ECS/ 的 IAM 政策AWS Fargate
- 亚马逊 EKS 的 IAM 政策
- 亚马逊 EMR 的 IAM 政策
- EKS 上的 Amazon EMR 的 IAM 政策
- 适用于 Amazon EMR Serverless 的 IAM 策略
- 适用于亚马逊的 IAM 政策 EventBridge
- 适用的 IAM 政策 AWS Lambda
- 适用的 IAM 政策 AWS Elemental MediaConvert

- 适用的 IAM 政策 AWS Glue
- 适用的 IAM 政策 AWS Glue DataBrew
- 适用于亚马逊的 IAM 政策 SageMaker
- 亚马逊 SNS 的 IAM 政策
- 亚马逊 SQS 的 IAM 政策
- 适用的 IAM 政策 AWS Step Functions
- 适用的 IAM 政策 AWS X-Ray
- 活动或无任务

亚马逊 API Gateway 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

资源：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:[[region]]:[[accountId]]:*"
            ]
        }
    ]
}
```

以下代码示例显示了调用 API Gateway 的资源策略。

```
{
    "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>"
                    ]
                }
            }
        }
    ]
}
```

亚马逊 Athena 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

StartQueryExecution

静态资源

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": [
            "*"
        ]
    }
]
```

```
}
```

动态资源

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

资源

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:stopQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    }
    ]
```

```
}
```

GetQueryExecution

资源

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:getQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    }
    ]
}
```

GetQueryResults

资源

```
{
    "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:::*"
        ]
    }
    ]
}
```

适用的 IAM 政策 AWS Batch

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

由于 AWS Batch 为资源级访问控制提供了部分支持，因此您必须使用。`"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

使用控制台创建状态机时，Step Functions 会自动为状态机创建一个具有所需最低权限的执行角色。这些自动生成的IAM角色对 AWS 区域 您在其中创建状态机的角色有效。

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

我们建议您在创建 IAM 策略时，不要在策略中包含通配符。作为安全最佳实操，应尽可能缩小策略范围。只有在运行时不知道某些输入参数时，才应使用动态策略。

本主题内容

- Amazon Bedrock 与 Step Functions 集成的 IAM 策略示例

Amazon Bedrock 与 Step Functions 集成的 IAM 策略示例

以下部分根据您用于特定基础或预置模型的 Amazon Bedrock API 说明了您需要的 IAM 权限。本部分还包含授予完全访问权限的策略示例。

切记用特定资源信息替换##文本。

- IAM使用访问特定基础模型的策略示例 InvokeModel
- IAM使用访问特定预配置模型的策略示例 InvokeModel
- 要使用的完全访问IAM策略示例 InvokeModel

- 将特定基础模型作为基本模型访问的 IAM 策略示例
- 将特定自定义模型作为基本模型访问的 IAM 策略示例
- 使用 CreateModelCustomizationJob .sync 的完全访问IAM策略示例
- IAM使用 CreateModelCustomizationJob .sync 访问特定基础模型的策略示例
- IAM使用 CreateModelCustomizationJob .sync 访问自定义模型的策略示例
- 使用 CreateModelCustomizationJob .sync 的完全访问IAM策略示例

IAM使用访问特定基础模型的策略示例 InvokeModel

以下是访问`amazon.titan-text-express-v1`使用 InvokeModelAPI 操作命名的特定基础模型的状态机的IAM策略示例。

```
{
    "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使用访问特定预配置模型的策略示例 InvokeModel

以下是访问`c2oi931ulksx`使用 InvokeModelAPI 操作命名的特定预配置模型的状态机的IAM策略示例。

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

```
      "Sid": "InvokeModel1",
      "Action": [
        "bedrock:InvokeModel"
      ],
      "Resource": [
        "arn:aws:bedrock:us-east-2:123456789012:provisioned-model/c2oi931ulksx"
      ]
    }
  ]
}
```

要使用的完全访问IAM策略示例 InvokeModel

以下是状态机的IAM策略示例，该状态机在您使用 InvokeModelAPI 操作时提供完全访问权限。

```
{
    "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/*"
            ]
        }
    ]
}
```

将特定基础模型作为基本模型访问的 IAM 策略示例

以下是状态机使用 CreateModelCustomizationJobAPI 操作访问名amazon.titan-text-express-v1为基础模型的特定基础模型的IAM策略示例。

```
{
    "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"
            ]
        }
    ]
}
```

将特定自定义模型作为基本模型访问的 IAM 策略示例

以下是状态机使用 CreateModelCustomizationJobAPI 操作访问作为基础模型的特定自定义模型的IAM策略示例。

```
{
    "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]]"
            ]
        }
    ]
}
```

使用 CreateModelCustomizationJob .sync 的完全访问IAM策略示例

以下是状态机的IAM策略示例，该状态机在您使用 CreateModelCustomizationJobAPI 操作时提供完全访问权限。

```
{
    "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使用 CreateModelCustomizationJob .sync 访问特定基础模型的策略示例

以下是状态机访问`amazon.titan-text-express-v1`使用 CreateModelCustomizationJob.sync API 操作命名的特定基础模型的IAM策略示例。

```
{
    "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使用 CreateModelCustomizationJob .sync 访问自定义模型的策略示例

以下是状态机使用 CreateModelCustomizationJob.sync API 操作访问自定义模型的IAM策略示例。

```
{
    "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"
            ]
        }
    ]
```

```
}
```

使用 CreateModelCustomizationJob .sync 的完全访问IAM策略示例

以下是状态机的IAM策略示例，该状态机在您使用 CreateModelCustomizationJob.sync API 操作时提供完全访问权限。

```
{
    "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"
            ]
        }
```

```
    ]
}
```

适用的 IAM 政策 AWS CodeBuild

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

资源：

```
{
    "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

静态资源

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]]"
      ]
    }
  ]
}
```

动态资源

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

静态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
```

```
    }
  ]
}
```

动态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

BatchDeleteBuilds

静态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

动态资源

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

BatchGetReports

静态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:report-group/[[reportName]]"
      ]
    }
  ]
}
```

动态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
```

```
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:report-group/*"
      ]
    }
  ]
}
```

StartBuildBatch

静态资源

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]]"
            ]
        }
    ]
}
```

动态资源

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

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
```

```
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

RetryBuildBatch

静态资源

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]]"
            ]
        }
    ]
}
```

动态资源

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

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

亚马逊 DynamoDB 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:[[region]]:[[accountId]]:table/[[tableName]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

有关所有 DynamoDB API 操作的 IAM 策略的更多信息，请参阅《Amazon DynamoDB 开发人员指南》中的 DynamoDB 的 IAM 策略。此外，有关 PartiQL for DynamoDB 的 IAM 策略的信息，请参阅《Amazon DynamoDB 开发人员指南》中的 PartiQL for DynamoDB 的 IAM 策略。

适用于 Amazon ECS/ 的 IAM 政策AWS Fargate

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

由于在提交任务之前 `TaskId` 的值始终是未知的，因此 Step Functions 会创建具有更高特权的 `"Resource": "*"` 策略。

> **Note**
>
> 尽管有 `"*"` IAM 策略，但您只能停止由 Step Functions 启动的 Amazon Elastic Container Service (Amazon ECS) 任务。

Run a Job (.sync)

静态资源

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

```
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:[[region]]:
[[accountId]]:task-definition/[[taskDefinition]]:[[revisionNumber]]"
            ]
        },
        {
            "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"
            ]
        }
    ]
}
```

动态资源

```
{
    "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)

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:[[region]]:
[[accountId]]:task-definition/[[taskDefinition]]:[[revisionNumber]]"
            ]
        }
    ]
}
```

动态资源

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": "*"
        }
    ]
}
```

如果您计划的 Amazon ECS 任务需要使用任务执行角色、任务角色或任务角色覆盖，则必须将每个任务执行角色、任务角色或任务角色覆盖的`iam:PassRole`权限添加到调用实体的 Ev CloudWatch ents IAM 角色（在本例中为 Step Functions）中。

亚马逊 EKS 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

`CreateCluster`

资源

```
{
    "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

资源

```
{
    "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

资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DeleteCluster",
                "eks:DescribeCluster"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:cluster/ExampleCluster"
            ]
        }
    ]
}
```

DeleteNodegroup

资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DeleteNodegroup",
                "eks:DescribeNodegroup"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:nodegroup/ExampleCluster/
ExampleNodegroup/*"
            ]
        }
    ]
}
```

有关将 Amazon EKS 与 Step Functions 搭配使用的更多信息，请参阅使用 Step Functions 调用 Amazon EKS。

亚马逊 EMR 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

addStep

静态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
```

```
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

动态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
}
```

cancelStep

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

createCluster

静态资源

```
{
  "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

静态资源

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

modifyInstanceFleetByName

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
```

```
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

modifyInstanceGroupByName

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

terminateCluster

静态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

动态资源

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
```

```
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
}
```

EKS 上的 Amazon EMR 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

`CreateVirtualCluster`

资源

```
{
    "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`

静态资源

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]]"
      ]
    }
  ]
}
```

动态资源

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`

静态资源

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]]"
          ]
        }
      }
    }
  ]
}
```

动态资源

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]]"
          ]
        }
      }
    }
  ]
}
```

适用于 Amazon EMR Serverless 的 IAM 策略

使用控制台创建状态机时，Step Functions 会自动为状态机创建一个具有所需最低权限的执行角色。这些自动生成的IAM角色对 AWS 区域 您在其中创建状态机的角色有效。

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

我们建议您在创建 IAM 策略时，不要在策略中包含通配符。作为安全最佳实操，应尽可能缩小策略范围。只有在运行时不知道某些输入参数时，才应使用动态策略。

此外，管理员用户在向非管理员用户授予运行状态机的执行角色时应谨慎行事。如果要自行创建策略，我们建议在执行角色中加入 passRole 策略。我们还建议在执行角色中添加 `aws:SourceARN` 和 `aws:SourceAccount` 上下文密钥。

本主题内容

- EMR Serverless 与 Step Functions 集成的 IAM 策略示例

EMR Serverless 与 Step Functions 集成的 IAM 策略示例

- 的 IAM 策略示例 CreateApplication
- 的 IAM 策略示例 StartApplication
- 的 IAM 策略示例 StopApplication
- 的 IAM 策略示例 DeleteApplication
- 的 IAM 策略示例 StartJobRun
- 的 IAM 策略示例 CancelJobRun

的 IAM 策略示例 CreateApplication

以下是带有状态的状态机的 IAM 策略示例。 CreateApplication 任务状态

> **Note**
>
> 在您的账户中创建有史以来第一个应用程序时，您需要在 IAM 策略中指定 CreateServiceLinkedRole 权限。此后，便无需再添加此权限。有关信息 CreateServiceLinkedRole，请参阅 https://docs.aws.amazon.com/IAM/latest/APIReference/CreateServiceLinkedRole中的。

以下策略的静态资源和动态资源相同。

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"
                }
            }
        }
    ]
}
```

的 IAM 策略示例 StartApplication

静态资源

以下是当您使用带有状态的状态机时静态资源的 IAM 策略示例。 StartApplication 任务状态

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]]"
            ]
        }
    ]
}
```

动态资源

以下是当您使用带有状态的状态机时动态资源的 IAM 策略示例。 StartApplication [任务状态](#)

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/*"
            ]
        }
    ]
}
```

的 IAM 策略示例 StopApplication

静态资源

以下是当您使用带有状态的状态机时静态资源的 IAM 策略示例。 StopApplication 任务状态

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]]"
             ]
        }
    ]
}
```

动态资源

以下是当您使用带有状态的状态机时动态资源的 IAM 策略示例。 StopApplication [任务状态](#)

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/*"
            ]
        }
    ]
}
```

的 IAM 策略示例 DeleteApplication

静态资源

以下是当您使用带有状态的状态机时静态资源的 IAM 策略示例。 DeleteApplication 任务状态

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]]"
            ]
        }
    ]
}
```

动态资源

以下是当您使用带有状态的状态机时动态资源的 IAM 策略示例。 DeleteApplication [任务状态](#)

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/*"
             ]
        }
    ]
}
```

的 IAM 策略示例 StartJobRun

静态资源

以下是当您使用带有状态的状态机时静态资源的 IAM 策略示例。 StartJobRun 任务状态

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"
                }
            }
        }
    ]
```

```
}
```

动态资源

以下是当您使用带有状态的状态机时动态资源的 IAM 策略示例。 StartJobRun 任务状态

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"
                }
            }
        }
    ]
}
```

的 IAM 策略示例 CancelJobRun

静态资源

以下是当您使用带有状态的状态机时静态资源的 IAM 策略示例。 CancelJobRun 任务状态

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]]"
            ]
        }
    ]
}
```

动态资源

以下是当您使用带有状态的状态机时动态资源的 IAM 策略示例。 CancelJobRun [任务状态](#)

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/*"
            ]
        }
    ]
}
```

适用于亚马逊的 IAM 政策 EventBridge

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

PutEvents

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:123456789012:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:PutEvents"
            ],
            "Resource": "arn:aws:events:*:*:event-bus/*"
        }
    ]
}
```

有关与 Step Functi EventBridge ons 配合使用的更多信息，请参阅 EventBridge 使用 Step Functions 调用。

适用的 IAM 政策 AWS Lambda

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

AWS Step Functions 根据您的状态机定义生成 IAM 策略。对于具有两个调用`function1`和的 AWS Lambda 任务状态的状态机`function2`，必须使用具有这两个函数`lambda:Invoke`权限的策略。

如以下示例所示。

```
{
    "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]]"
            ]
        }
    ]
}
```

适用的 IAM 政策 AWS Elemental MediaConvert

以下示例模板显示了如何 AWS Step Functions 要求您根据状态机定义中的资源设置 IAM 策略。您可以使用 IAM 控制台添加任何缺少的角色策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

由于 MediaConvert 为资源级访问控制提供了部分支持，因此您必须使用。`"Resource": "*"`

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mediaconvert:CreateJob",
                "mediaconvert:GetJob",
                "mediaconvert:CancelJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForMediaConvertJobRule"
            ]
        }
    ]
```

```
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mediaconvert:CreateJob"
            ],
            "Resource": "*"
        }
    ]
}
```

适用的 IAM 政策 AWS Glue

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

AWS Glue 没有基于资源的控制。

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": "*"
        }
    ]
}
```

适用的 IAM 政策 AWS Glue DataBrew

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

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/*"
            ]
        }
    ]
}
```

适用于亚马逊的 IAM 政策 SageMaker

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

> **Note**
>
> 对于这些示例，*[[roleArn]]*请参阅 IAM 角色的 Amazon 资源名称 (ARN)，该 SageMaker 角色用于访问模型工件和 docker 镜像，以便在 ML 计算实例上部署，或用于批量转换任务。有关更多信息，请参阅 Amazon SageMaker 角色。

`CreateTrainingJob`

静态资源

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"
```

```
        }
      }
    }
  ]
}
```

动态资源

.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 当您创建与集成的状态机`CreateTransformJob`时，不会自动创建策略 SageMaker。您必须根据以下 IAM 示例之一将内联策略附加到创建的角色。

静态资源

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"
        }
      }
    }
  ]
}
```

动态资源

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"
        }
      }
    }
  ]
}
```

亚马逊 SNS 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:[[region]]:[[accountId]]:[[topicName]]"
            ]
        }
    ]
}
```

基于路径，或发布到 *TargetArn* 或 *PhoneNumber* 的资源

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

```
                "sns:Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

亚马逊 SQS 的 IAM 政策

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

静态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": [
                "arn:aws:sqs:[[region]]:[[accountId]]:[[queueName]]"
            ]
        }
    ]
}
```

动态资源

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

适用的 IAM 政策 AWS Step Functions

对于为单个嵌套工作流执行调用 `StartExecution` 的状态机，应使用 IAM 策略限制该状态机的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:stateMachine:[[stateMachineName]]"
            ]
        }
    ]
}
```

有关更多信息，请参阅下列内容：

- 与其他服务 AWS Step Functions 一起使用
- 将参数传递给服务 API
- 将 AWS Step Functions 执行作为集成服务进行管理

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]]"
                ]
            }
        ]
    }
```

有关嵌套工作流执行的更多信息，请参阅从 Task 状态启动工作流执行。

适用的 IAM 政策 AWS X-Ray

以下示例模板展示了如何根据状态机定义中的资源 AWS Step Functions 生成 IAM 策略。有关更多信息，请参阅 集成服务的 IAM 策略 和 服务集成模式。

要启用 X-Ray 跟踪，您需要具有适当权限的 IAM 策略以允许跟踪。如果您的状态机使用其他集成服务，则可能需要其他 IAM 策略。有关特定服务集成，请参阅 IAM 政策。

当创建启用了 X-Ray 跟踪的状态机时，会自动创建 IAM 策略。

> **Note**
>
> 如果为现有状态机启用 X-Ray 跟踪，必须确保添加的策略有足够的权限来启用 X-Ray 跟踪。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

有关将 Step Functions 与 X-Ray 搭配使用的更多信息，请参阅 AWS X-Ray 和 Step Functions。

活动或无任务

对于仅具有 `Activity` 任务或根本没有任务的状态机，请使用拒绝访问所有操作和资源的 IAM 策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

有关使用 `Activity` 任务的更多信息，请参阅活动。

使用分布式 Map 状态的 IAM 策略

当您使用 Step Functions 控制台创建工作流时，Step Functions 可以根据工作流定义中的资源自动生成 IAM 策略。这些策略包括允许状态机角色调用分布式 Map 状态 的 `StartExecution` API 操作所需的最低权限。这些策略还包括 Step Functions 访问 AWS 资源（例如 Amazon S3 存储桶和对象以及 Lambda 函数）所需的最低权限。我们建议在您的 IAM 策略中仅包含这些必需的权限。例如，如果您的工作流包含分布式模式下的 `Map` 状态，则将策略范围缩小到包含您的数据集的特定 Amazon S3 存储桶和文件夹。

> **Important**
>
> 如果您在分布式 Map 状态 输入中指定了 Amazon S3 存储桶和对象或前缀，并将参考路径指向现有键值对，请务必更新工作流程的 IAM 策略。将策略范围缩小到运行时该路径解析到的存储桶和对象名称。

本主题内容：

- 运行分布式 Map 状态的 IAM 策略示例
- redriving分布式 Map 的 IAM 策略示例
- 从 Amazon S3 数据集读取数据的 IAM 策略示例

- 将数据写入 Amazon S3 存储桶的 IAM 策略示例

运行分布式 Map 状态的 IAM 策略示例

当您在工作流中包含分布式 Map 状态 时，Step Functions 需要适当的权限才能允许状态机角色为分布式 Map 状态 调用 StartExecution API 操作。

以下 IAM 策略示例授予您的状态机角色运行分布式 Map 状态 所需的最低权限。

> **Note**
>
> 确保将 *stateMachineName* 替换为使用分布式 Map 状态 的状态机的名称。例如，arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*。

```
{
  "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:*"
    }
  ]
}
```

redriving分布式 Map 的 IAM 策略示例

您可以通过redriving父工作流，重新启动未成功的子工作流执行。redriven父工作流会redrives所有未成功的状态，包括分布式 Map。确保您的执行角色具有允许其在父工作流上调用 `RedriveExecution` API 操作所需的最低权限。

下面的 IAM 策略示例授予状态机角色redriving分布式 Map 状态 所需的最低权限。

> **Note**
>
> 确保将 *stateMachineName* 替换为使用分布式 Map 状态 的状态机的名称。例如，arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
}
```

从 Amazon S3 数据集读取数据的 IAM 策略示例

以下 IAM 策略示例授予使用 ListObjectsV2 和 GetObjectAPI 操作访问您的 Amazon S3 数据集所需的最低权限。

Example Amazon S3 对象作为数据集的 IAM 策略

以下示例显示了一个 IAM 策略，该策略可授予访问名为 *myBucket* 的 Amazon S3 存储桶的 *processImages* 中组织的对象的最低权限。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example 将 CSV 文件作为数据集的 IAM 政策

以下示例显示一个 IAM 策略，该策略授予可授予访问名为 *ratings.csv* 的 CSV 文件的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example Amazon S3 清单作为数据集的 IAM 策略

以下示例显示了一个 IAM 策略，可授予访问 Amazon S3 清单报告的最低权限。

```
{
    "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/*"
            ]
        }
    ]
}
```

将数据写入 Amazon S3 存储桶的 IAM 策略示例

下面的 IAM 策略示例授予使用 PutObject API 操作将子工作流执行结果写入 Amazon S3 存储桶中名为 *csvJobs* 的文件夹所需的最低权限。

```
{
    "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 加密的 Amazon S3 存储桶的 IAM 权限

分布式 Map 状态 使用多部份内容上传将子工作流执行结果写入 Amazon S3 存储桶。如果使用 AWS Key Management Service (AWS KMS) 密钥对存储桶进行加密，则还必须在 IAM 策略中包含对密钥执行 `kms:Decrypt`、`kms:Encrypt` 和 `kms:GenerateDataKey` 操作的权限。这些权限是必需的，因为 Amazon S3 必须在完成分段上传之前解密并读取加密的文件段中的数据。

下面的 IAM 策略示例对用于加密 Amazon S3 存储桶的密钥的 `kms:Decrypt`、`kms:Encrypt` 和 `kms:GenerateDataKey` 操作授予了权限。

```
{
  "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"
    ]
  }
}
```

有关更多信息，请参阅 AWS 知识中心 中的用 AWS KMS key加密将大型文件上传到 Amazon S3 。

如果您的 IAM 用户或角色与 AWS 账户 相同KMS key，则您必须对密钥策略拥有这些权限。如果您的 IAM 用户或角色属于与 KMS key 不同的账户，您必须在密钥政策和 IAM 用户或角色中具有这些权限。

基于标签的策略

Step Functions 支持基于标签的策略。例如，您可能会限制对下面这样的所有 Step Functions 资源的访问：在这些资源包含的标签中，具有键 `environment` 和值 `production`。

```
{
    "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"}
            }
        }
    ]
}
```

此策略将使用 Deny 来拒绝对已标记为 environment/production 的所有资源的以下功能：删除状态机或活动、停止执行以及添加或删除新的标签。

对于基于标签的授权，状态机执行资源（如下例所示）会继承与状态机关联的标签。

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

当您调用DescribeExecution或其他 API 并在其中指定执行资源 ARN 时，Step Functions 会在执行基于标签的授权时使用与状态机关联的标签来接受或拒绝请求。这有助于在状态机级别允许或拒绝对状态机执行的访问。

有关标记的更多信息，请参阅以下内容：

- Step Functions 中的标记
- 使用 IAM 标签控制访问

对 AWS Step Functions 身份和访问进行故障排除

使用以下信息可帮助您诊断和修复在使用 Step Functions 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Step Functions 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我的 Step AWS 账户 Functions 资源

我无权在 Step Functions 中执行操作

如果您收到一个错误，指明您无权执行某个操作，则必须更新策略以允许您执行该操作。

当 `mateojackson` 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `states:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 states:GetWidget on resource: my-example-widget
```

在此情况下，Mateo 的策略必须更新以允许其使用 `states:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Step Functions。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Step Functions 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我的 Step AWS 账户 Functions 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Step Functions 是否支持这些特征，请参阅 如何 AWS Step Functions 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅 IA M 用户指南中的跨账户资源访问。

日志记录和监控

有关登录和监控的信息 AWS Step Functions，请参阅一日记账记录和监控节。

的合规性验证 AWS Step Functions

AWS Step Functions 作为多个合规计划的一部分，第三方审计师对安全性和 AWS 合规性进行评估。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

有关特定合规计划范围内的 AWS 服务列表，请参阅合规计划范围内的AWS 服务按合规计划。有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您使用 Step Functions 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用的法律法规。 AWS 提供以下资源来帮助满足合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在上部署以安全性和合规性为重点的基准环境的步骤。 AWS
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规性 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 标准的应用程序。
- AWS 合AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。

- AWS Security Hub— 此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

韧性在 AWS Step Functions

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础架构外，Step Functions 还提供多项功能来帮助支持您的数据弹性和备份需求。

中的基础设施安全 AWS Step Functions

作为一项托管服务 AWS Step Functions ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用 Step Functions 通过网络进行访问。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

您可以从任何网络位置调用 AWS API 操作，但Step Functions不支持基于资源的访问策略，其中可能包括基于源 IP 地址的限制。您还可以使用 Step Functions 策略来控制来自特定 Amazon Virtual Private Cloud (Amazon VPC) 端点或特定 VPC 的访问。实际上，这可以将对给定Step Functions资源的网络访问与网络中的特定 VPC 隔离开来。 AWS

中的配置和漏洞分析 AWS Step Functions

配置和 IT 控制由您（我们的客户）共同 AWS 负责。有关更多信息，请参阅责任 AWS 共担模型。

将工作负载从 Step F AWS Data Pipeline unctions 迁移

AWS 该 AWS Data Pipeline 服务于 2012 年推出。当时，客户想要一种服务，以允许他们使用各种计算选项在不同数据来源之间移动数据。随着时间的推移，数据传输需求发生了改变，因此满足这些需求的解决方案也随之改变。现在，您可以选择最符合您的业务要求的解决方案。例如，您可以执行以下任一操作：

- 使用 Step Functions 来编排多个 AWS 服务 之间的工作流程。
- 使用 Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 来管理 Apache Airflow 的工作流程编排。
- AWS Glue 用于运行和编排 Apache Spark 应用程序。

您可以将的典型用例迁移到 Step F AWS Data Pipeline unctions 或 Amazon MWAA。 AWS Glue您选择的选项因 AWS Data Pipeline 上的当前工作负载而定。本主题介绍如何从迁移到 Step F AWS Data Pipeline unctions。

主题

- 从 AWS Data Pipeline 迁移工作负载
- Step Functions 和 AWS Data Pipeline 之间的概念映射
- Step Functions 示例项目
- 定价比较

从 AWS Data Pipeline 迁移工作负载

Step Functions 是一项无服务器编排服务，可让您为业务关键型应用程序构建工作流。借助 Step Functions Workflow Studio，您可以构建工作流，并将其与 250 多个 AWS 服务中的 11,000 多个 API 操作集成。这 AWS 服务 包括亚马逊 EMR 和亚马逊 DynamoDB 等。 AWS Lambda您还可以使用 Step Functions 来编排数据处理管道、处理错误以及处理底层 AWS 服务的节流限制。您可以使用 AWS Glue 创建处理和发布机器学习模型、编排微服务以及处理提取、转换、加载 (ETL) 工作流。您还可以为需要人际互动的应用程序创建长时间运行的自动化工作流。

Step Functions 是由 AWS 提供的一项完全托管的服务。这意味着 AWS 会管理各种任务，例如维护基础架构、修补工作线程和管理操作系统版本更新。

当您的用例符合以下条件时，我们建议您从 Step Functions 迁移 AWS Data Pipeline 到 Step Functions：

- 您更喜欢无服务器、高度可用的工作流编排服务。
- 您需要一种按单个任务执行的粒度收费的解决方案。
- 您的工作负载涉及为其他多个工作负载编排任务 AWS 服务，例如 Amazon EMR、Lambda 或 DynamoD AWS Glue B。
- 您需要一个带有 drag-and-drop 可视化设计器的低代码解决方案来创建工作流程。该解决方案不需要学习陌生、复杂的编程概念。
- 你需要一项与超过 250 个 API 操作集成的服务 AWS 服务 ，涵盖超过 11,000 个 API 操作。此服务还必须与外部的自定义服务和活动集成 AWS。

Step Functions 和 AWS Data Pipeline 之间的概念映射

AWS Data Pipeline 和 Step Functions 有一些共同的概念。例如，要定义工作流程，可以在两者 AWS Data Pipeline 和 Step Functions 中使用 JSON 格式。在 Step Functions 中，您可以使用 Amazon States Language，这是一种基于 JSON 的结构化语言。您可以使用 Amazon States Language (ASL) 定义工作流，并在工作流的文本和可视化表示之间切换。这种基于 JSON 的格式有助于简化在源控制工具中存储工作流的过程。它还能帮助您管理工作流的多个版本、控制其访问权限或使用 CI/CD 方法自动编排。

下表描述了两种服务中使用的主要概念之间的映射。左侧的 “数据管道概念” 列出了中的概念 AWS Data Pipeline，而右侧的 “Step Functions 概念” 列出了 Step Functions 中的等效概念。

Data pipeline 概念	Step Functics 概念
管线	工作流
管道定义	Amazon States Language (ASL)
活动	状态 和 任务状态
实例	执行
Attempts	捕获器和重试器
管道计划	• 使用 Amazon EventBridge 计划程序执行任务 • 通过 EventBridge 管道触发的事件
管道表达式和函数	• 内置函数

Data pipeline 概念	Step Functics 概念
	• 使用服务集成的 Lambda 函数

Step Functions 示例项目

有关 Step Functions 的简介，请参阅以下视频：

服务编排入门 AWS Step Functions

以下列表概述了一些示例项目，这些项目使用 Step Functions 实现了最常见的 AWS Data Pipeline 用例。您可以使用这些示例项目作为参考，从 Step Functions 迁移 AWS Data Pipeline 到 Step Functions。您也可以将它们用作样板，根据自己的用例构建自己的工作流并与支持的 AWS 服务集成。

- 管理 Amazon EMR 任务
- 在 Amazon EMR Serverless 上运行数据处理任务
- 正在运行 Hive/Pig/Hadoop 任务
- 查询大型数据集（亚马逊 Athena、亚马逊 S3、亚马逊 SN AWS Glue S）
- 使用 Amazon Redshift 运行 ETL/ELT 工作流程
- 编排爬虫 AWS Glue
- 使用 Step Functions 运行 Shell 脚本

要了解有关 Step Functions 的更多信息，请参阅以下主题和资源：

- Step Functions 教程
- Step Functions 的示例项目
- AWS Step Functions 研讨会

定价比较

AWS Data Pipeline 按管道数量及其使用水平定价。您每天运行一次以上（高频率）的活动每项活动每月花费 1 美元。您每天运行一次或以下（低频率）的活动每项活动每月花费 0.6 美元。非活跃管道的价格为每条管道 1 美元。有关定价的更多信息，请参阅 AWS Data Pipeline 定价页面。

Step Functions 有两种类型的工作流：标准工作流和快速工作流。每种工作流类型都有不同的定价模式。这种比较基于标准工作流程，因为它最符合中的常见用例 AWS Data Pipeline。标准工作流的定价为每 1000 个状态转换 0.025 美元。不活动的状态机无需付费；您只需为使用的状态机付费。有关定价的更多信息，请参阅 AWS Step Functions 定价页面。

故障排除

如果您在使用 Step Functions 时遇到问题，请参阅以下故障排除资源。

主题

- 一般故障排除
- 服务集成故障排除
- 活动故障排除
- 快速工作流故障排除

一般故障排除

我无法创建状态机。

与状态机关联的 IAM 角色可能没有足够的权限。检查 IAM 角色的权限，包括 AWS 服务集成任务、X-Ray 和 CloudWatch 日志记录的权限。`.sync` 任务状态需要额外的权限。

我无法使用 JsonPath 来引用上一个任务的输出。

JsonPath 的 JSON 密钥必须以 `.$` 结尾。这意味着 JsonPath 只能在键值对中使用。如果想在其他地方使用 JsonPath（比如数组），则可以使用内置函数。例如，可以使用类似以下代码的内容：

任务 A 的输出：

```
{
    "sample": "test"
}
```

任务 B：

```
{
    "JsonPathSample.$": "$.sample"
}
```

> **Tip**
>
> 使用 Step Functions 控制台中的数据流模拟器来测试 JSON 路径语法，以更好地了解在状态下如何操作数据，并查看数据在状态之间是如何传递的。

状态转换出现延迟。

对于标准工作流，状态转换的数量是有限制的。当超过状态转换限制时，Step Functions 会延迟状态转换，直到配额存储桶填满。可以通过查看 CloudWatch 指标页面 执行指标 部分中的 `ExecutionThrottled` 指标来监控状态转换限制节流。

启动新的标准工作流执行时，会执行失败并出现 **`ExecutionLimitExceeded`** 错误。

Step Functions 对每个 AWS 区域的每个 AWS 账户的开放执行次数限制为 100 万次。如果超过此限制，Step Function 会抛出 `ExecutionLimitExceeded` 错误。此限制不适用于快速工作流。您可以使用《Amazon CloudWatch 用户指南》中的以下 CloudWatch Metrics 数学来估算开放执行次数：`ExecutionsStarted - (ExecutionsSucceeded + ExecutionsTimedOut + ExecutionsFailed + ExecutionsAborted)`。

一个处于并行状态的分支出现故障，导致整个执行失败。

这是一个预期行为。为避免在使用并行状态时遇到故障，请将 Step Functions 配置为捕获每个分支抛出的错误。

服务集成故障排除

在下游服务中，我的作业已经完成，但在 Functions 中，任务状态仍为“进行中”或延迟完成。

对于 `.sync` 服务集成模式，Step Functions 使用 EventBridge 规则、下游 API 或两者的组合来检测下游作业状态。对于某些服务，Step Functions 不会创建 EventBridge 规则进行监控。例如，对于 AWS Glue 服务集成，Step Functions 不使用 EventBridge 规则，而是进行 `glue:GetJobRun` 调用。由于 API 调用的频率，下游任务完成时间与 Step Functions 任务完成时间存在差异。Step Functions 需要 IAM 权限才能管理 EventBridge 规则和调用下游服务。有关执行角色权限不足如何影响任务完成的详细信息，请参阅使用“运行作业”模式执的任务附加权限。

我想从嵌套状态机执行中返回 JSON 输出。

Step Functions 有两个同步服务集成：`startExecution.sync` 和 `startExecution.sync:2`。两者都将等待嵌套状态机完成，但它们返回不同的 `Output` 格式。您可以使用 `startExecution.sync:2` 在 `Output` 下返回 JSON 输出。

我无法从另一个账户调用 Lambda 函数。

通过跨账户支持访问 Lambda 函数

如果您所在区域提供 AWS 资源的跨账户存取，请使用以下方法从另一个帐户调用 Lambda 函数。

要在工作流中调用跨账户资源，请执行以下操作：

1. 在包含资源的目标账户中创建 IAM 角色。此角色向包含状态机的源账户授予访问目标账户资源的权限。
2. 在 `Task` 状态的定义中，指定在调用跨账户资源之前由状态机担任的目标 IAM 角色。
3. 修改目标 IAM 角色中的信任策略来允许源账户临时担任此角色。信任策略必须包含源账户中定义的状态机的 Amazon 资源名称（ARN）。此外，还要在目标 IAM 角色中定义调用 AWS 资源的相应权限。
4. 更新源账户的执行角色，使其包含担任目标 IAM 角色所需的权限。

有关示例，请参阅教程：访问跨账户资源 AWS。

> **Note**
>
> 您可以配置状态机承担一个 IAM 角色，以便从多个 AWS 账户访问资源。但是，状态机在给定时间只能承担一个 IAM 角色。

有关指定跨账户资源的 `Task` 状态定义示例，请参阅 Task 状态的“凭证”字段示例。

在没有跨账户支持的情况下访问 Lambda 函数

如果您所在区域不提供 AWS 资源的跨账户存取，请使用以下方法从另一个帐户调用 Lambda 函数。

在 `Task` 状态的 `Resource` 字段中，使用 `arn:aws:states:::lambda:invoke` 并传递参数中的 `FunctionArn`。与状态机关联的 IAM 角色必须拥有调用跨账户 Lambda 函数的正确权限：`lambda:invokeFunction`。

```
{
   "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
      }
   }
}
```

我无法看到从 `.waitForTaskToken` 状态传递的任务令牌。

您必须在 `Task` 状态的 `Parameters` 字段中传递任务令牌。例如，您可以使用与以下代码类似的内容。

```
{
   "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**
>
> 您可以尝试在任何 API 操作中使用 `.waitForTaskToken`。不过，有些 API 没有合适的参数。

活动故障排除

我的状态机执行卡在活动状态。

在使用 GetActivityTask API 操作轮询任务标记之前，活动任务状态不会启动。最佳实操是添加任务级超时，用于避免执行卡住。有关更多信息，请参阅使用超时避免执行卡顿。

如果您的状态机在 ActivityScheduled 事件中卡住，这表明您的活动工作线程实例集存在问题或规模不足。您应该监控 `ActivityScheduleTime` CloudWatch 指标，并在该时间增加时设置警报。不过，要想超时处理任何卡住的状态机执行（其中 `Activity` 状态没有转换到 `ActivityStarted` 状态），可在状态机级别定义超时。为此，请在状态机定义的开头指定一个 `TimeoutSeconds` 字段，该字段位于 `States` 字段之外。

我的活动工作线程在等待任务令牌时超时。

工作线程使用 GetActivityTask API 操作来检索具有指定活动 ARN 的任务，该活动 ARN 计划由运行中的状态机执行。`GetActivityTask` 会启动长时间轮询，因此服务会保持 HTTP 连接打开，并在任务可用时立即做出响应。服务在响应之前保留请求的最长时间为 60 秒。如果在 60 秒内没有任务可用，轮询将返回一个带空字符串的 `taskToken`。要避免此类超时，请在 AWS 开发工具包或用于进行 API 调用的客户端中配置一个超时至少 65 秒的客户端套接字。

快速工作流故障排除

我的应用程序在收到 `StartSyncExecution` API 调用的响应之前超时。

在用于进行 API 调用的 AWS 开发工具包或客户端中配置客户端套接字超时。要接收响应，超时值必须高于快速工作流执行的持续时间。

我无法查看执行历史记录，无法解决快速工作流故障。

快速工作流不会在 AWS Step Functions 中记录执行历史。您必须开启 CloudWatch 日志记录。日志记录开启后，您可以使用 CloudWatch Logs Insights 查询来查看快速工作流的执行情况。如果在执行选项卡中选择启用按钮，还可以在 Step Functions 控制台上查看快速工作流的执行历史记录。有关更多信息，请参阅在 Step Functions 控制台上查看和调试执行。

根据持续时间列出执行情况：

```
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
```

列出失败和取消的执行：

```
fields ispresent(execution_arn) as isRes | filter type in ["ExecutionFailed",
 "ExecutionAborted", "ExecutionTimedOut"]
```

相关信息

下表列出了在您使用此服务时可能为您提供帮助的相关资源。

资源	描述
AWS Step Functions API 参考	API 操作、参数和数据类型的说明，以及该服务返回的错误的列表。
AWS Step Functions 命令行参考	可用于使用 AWS Step Functions 的 AWS CLI 命令的说明。
Step Functions 产品信息	提供有关 Step Functions 信息的主要 Web 页面。
开发论坛	由开发人员组成的社区形式的论坛，他们可以在这里讨论与 Step Functions 和其他 AWS 服务有关的技术问题。
AWS Support 信息	提供有关 AWS Support 信息的主要 Web 页面，它是一种一对一、快速响应的支持渠道，可帮助您在 AWS 基础设施服务上构建和运行应用程序。

最近发布的特征

下表列出了提供新 Step Functions 特征的地区。

推出日期	特征名称	区域可用性
2023 年 11 月 26 日	调用公共 HTTPS 端点并测试各个状态	• 美国东部（弗吉尼亚北部）- us-east-1 • 美国西部（俄勒冈）- us-west-2 • 美国东部（俄亥俄）- us-east-2 • 欧洲（爱尔兰）- eu-west-1 • 欧洲（法兰克福）- eu-central-1 • 欧洲（斯德哥尔摩）- eu-north-1 • 亚太地区（悉尼）- ap-southeast-2 • 亚太地区（东京）- ap-northeast-1 • 亚太地区（新加坡）- ap-southeast-1
2023 年 11 月 15 日	Redrive执行	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。
2023 年 10 月 12 日	Amazon EMR Serverless 的优化集成	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。

推出日期	特征名称	区域可用性
2023 年 9 月 7 日	增强的错误处理	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。
2023 年 8 月 31 日	Workflow Studio 增强功能带来简化的创作体验	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。
2023 年 6 月 22 日	版本与别名功能	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。
2023 年 6 月 16 日	全新 AWS 开发工具包集成	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。
2022 年 12 月 1 日	利用分布式 Map 状态 为数据处理编排大规模并行工作流	有关可使用此特征的 AWS 区域的完整列表，请参阅标题为按区域划分的 AWS 服务页面上区域下拉列表中的选项。

文档历史记录

本部分列出了《AWS Step Functions 开发人员指南》的主要更改。

更改	描述	更改日期
更新	AWS 托管策略更新-新权限：`states:ValidateStateMachineDefinition` 添加了有关检查您提供的状态机语法的新权限的信息。要了解更多信息，请参阅AWS 的托管策略 AWS Step Functions。	2024 年 4 月 29 日
新功能	Step Functions 增加了优化的集成 AWS Elemental MediaConvert AWS Elemental MediaConvert 提供广播级视频和音频文件转码，客户可以使用适合其媒体工作流程的代码自动进行转码。通过针对in的优化集成 MediaConvert，现在可以使用低代码可视化工具 Workflow Studio 进行编排。 AWS Step Functions 要了解更多信息，请参阅使用 Ste p Funct AWS Elemental MediaConvert ions 进行管理的文档。	2024 年 4 月 12 日
更新	AWS 托管策略更新-更新现有策略：`AWSStepFunctionsReadOnlyAccess` 添加了有关标签、分布式地图、版本和别名的新只读权限的信息。要了解更多信息，请参阅AWS 的托管策略 AWS Step Functions。	2024年4月2日
更新	Step Functions 增加了对开放工作流程指标的支持 有了开放的工作流程指标，您现在可以在账户层面查看正在进行的标准工作流程数量以及打开的工作流程限制。您可以管理所有工作流程中的工作负载，无论它们是如何启动的，以确保工作流顺畅运行。您可以设置 CloudWatch 警报以监控工作流程，并在接近极限时主动接收警报。收到警报后，	2024 年 2 月 29 日

更改	描述	更改日期
	您可以通过采取诸如停止特定工作流程或请求提高限额之类的操作来有效地管理工作流程。 开放式工作流指标 CloudWatch 可用于标准工作流程，无需额外配置。要了解更多信息，请参阅执行指标。	
更新	服务集成新增和更新。有关新增和更新的 AWS SDK 集成列表，请参阅更改支持的 AWS SDK 集成的日志。有关服务的完整列表，请参阅支持的 AWS SDK 服务集成。	2024 年 1 月 18 日
新功能	使用 Application Composer 中的 Workflow Studio 利用 AWS CloudFormation 模板构建无服务器工作流。有关更多信息，请参阅 使用 Application Composer 中的 Workflow Studio。	2023 年 11 月 27 日
新功能	Step Functions 现在可让您使用新的 Test State API 直接调用公共 HTTPS 端点并测试各个状态。有关更多信息，请参阅： • 调用第三方 API • 使用 TestState API 测试状态	2023 年 11 月 26 日
新功能	Step Functions 现在与 Amazon Bedrock 相集成。有关更多信息，请参阅以下主题： • 使用 Step Functions 调用 Amazon Bedrock • Amazon Bedrock 的 IAM 权限 • 使用 Amazon Bedrock 执行 AI 提示链接 • 与其他服务 AWS Step Functions 一起使用	2023 年 11 月 26 日
新功能	Step Functions 现在可让您从故障点redrive标准类型的工作流程执行。有关更多信息，请参阅 Redriving执行 和 Redriving Map Run。	2023 年 11 月 15 日

更改	描述	更改日期
仅文档更新	发布了一个新主题，介绍如何使用 Amazon EventBridge Scheduler按计划运行状态机。有关更多信息，请参阅 将 Amazon EventBridge 调度器与 AWS Step Functions 结合使用。	2023 年 10 月 16 日
新功能	Step Functions 现在与 Amazon EMR Serverless 相集成。有关更多信息，请参阅以下主题： • 使用 Step Functions 调用 Amazon EMR Serverless • 运行 EMR Serverless 作业 • Step Functions 的优化集成 • 与其他服务 AWS Step Functions 一起使用	2023 年 10 月 12 日
仅文档更新	添加了关于使用 Amazon EventBridge Scheduler按计划运行状态机的信息。有关更多信息，请参阅 使用 EventBridge 调度器。	2023 年 10 月 5 日
更新	重组并更新了分布式 Map 状态 主题，使其清晰简洁，并为新用户创建了清晰的流程图。有关更多信息，请参阅 使用分布式模式下的 Map 状态编排大规模并行工作负载。	2023 年 10 月 6 日
修复	修复了教程中使用 AWS CDK v2 的代码示例。有关更多信息，请参阅 使用 AWS CDK 为 Step Functions 创建一个 Lambda 状态机。	2023 年 9 月 19 日
更新	添加了有关 Step Functions 引入增强的错误处理功能的信息，以清晰地识别错误并以更强的控制力实施重试。有关更多信息，请参阅 Fail 和 出错后重试。	2023 年 9 月 7 日
更新	Step Functions 为 Workflow Studio 添加了增强功能，简化工作流的编写体验。有关更多信息，请参阅 AWS Step Functions 工作流程工作室。	2023 年 8 月 31 日
仅文档更新	添加了关于 `ExecutionsStarted` 指标报告的实际指标计数加倍的信息。有关更多信息，请参阅 报告计数的指标。	2023 年 7 月 25 日

更改	描述	更改日期
仅文档更新	Step Functions 新增了两个示例项目，演示了分布式 Map 状态 的以下常见用例： • 处理 CSV 文件 • 处理 Amazon S3 存储桶中的数据	2023 年 7 月 17 日
仅文档更新	发布了关于使用 Terraform 部署状态机的新主题。有关更多信息，请参阅 使用 Terraform 部署状态机。	2023 年 7 月 5 日
仅文档更新	更新了以下过程以匹配对 Amazon EventBridge 界面的更改。 • 将 Step Functions 事件路由到 EventBridge • 启动状态机执行以响应 Amazon S3 事件	2023 年 6 月 26 日
新功能	Step Functions 现在可以创建多个状态机版本和别名功能，从而在部署无服务器工作流时提高弹性。有关更多信息，请参阅 使用版本与别名功能管理持续部署。	2023 年 6 月 22 日
仅文档更新	改进了 `TimeoutSeconds` 和 `HeartbeatSeconds` 字段的描述，说明了它们之间的区别。有关更多信息，请参阅 Task 状态字段。	2023 年 6 月 22 日
仅文档更新	发布了一个新章节，介绍了如何将通常作为 Parallel 和 Map 状态结果返回的数组进行展平处理。有关更多信息，请参阅 展平由数组组成的数组。	2023 年 6 月 20 日
更新	Step Functions 增加了七个 AWS 服务 和 468 个新的 API 操作，从而扩大了对 AWS SDK 集成的支持。有关更多信息，请参阅 支持的 AWS SDK 服务集成 和 更改支持的 AWS SDK 集成的日志。	2023 年 6 月 16 日
仅文档更新	发布了一个新主题，其中列出了最近推出的 AWS 区域 Step Functions 功能的可用内容。有关更多信息，请参阅 最近发布的特征。	2023 年 6 月 16 日

更改	描述	更改日期
仅文档更新	Step Functions 现在包括一个关于 AWS 用户通知服务的部分 AWS 服务 ，它充当中 AWS 通知的中心位置 AWS Management Console。有关更多信息，请参阅 配合使用 AWS 用户通知服务和 Step Functions。	2023 年 5 月 4 日
仅文档更新	添加了一个新章节，说明将子工作流执行结果写入使用 AWS Key Management Service (AWS KMS) 密钥加密的 Amazon S3 存储桶所需的权限。有关更多信息，请参阅 AWS KMS key 加密的 Amazon S3 存储桶的 IAM 权限。	2023 年 4 月 29 日
仅文档更新	添加了一个新主题，介绍数据流模拟器特征。有关更多信息，请参阅 数据流模拟器。	2023 年 4 月 14 日
配额更新	添加了有关每个账户中开放 Map Run 的默认配额为 1000 的信息。有关更多信息，请参阅 与账户相关的配额。	2023 年 4 月 5 日
仅文档更新	添加了一个主题，描述何时将 AWS Data Pipeline 工作负载迁移到 Step Functions。本主题还提供了解释如何执行迁移的示例列表。有关更多信息，请参阅 将工作负载从 Step F AWS Data Pipeline unctions 迁移。	2023 年 3 月 30 日
仅文档更新	添加了关于分布式 Map 状态 下的 X-Ray 跟踪不可用的注释。有关更多信息，请参阅 AWS X-Ray 和 Step Functions 。	2023 年 3 月 21 日
仅文档更新	添加了有关 Step Functions 如何处理基于标签的授权信息。有关更多信息，请参阅 Step Functions 中的标记 和 基于标签的策略。	2023 年 3 月 15 日
仅文档更新	添加了有关 Step Functions 如何解析在分布式 Map 状态中用作输入的 CSV 文件的信息。有关更多信息，请参阅 Amazon S3 存储桶中的 CSV 文件。	2023 年 3 月 14 日
仅文档更新	添加了有关 Step Functions 如何处理运行作业 (.sync) 模式的跨账户调用的信息。有关更多信息，请参阅运行作业 (.sync)。	2023 年 3 月 1 日

更改	描述	更改日期
仅文档更新	将已完成的工作流执行的历史记录保留期从 90 天缩短至 30 天。有关更改保留期的更多信息，请参阅执行保障和与状态机执行相关的配额。	2023 年 2 月 21 日
更新	Step Functions 通过添加了 35 项 AWS 服务和 1100 个新的 API 操作，扩展了对 AWS SDK 集成的支持。有关更多信息，请参阅 支持的 AWS SDK 服务集成 和 更改支持的 AWS SDK 集成的日志。	2023 年 2 月 17 日
仅文档更新	发布了入门教程系列，指导您完成使用 Step Functions 创建信用卡申请工作流的过程。有关更多信息，请参阅 入门 AWS Step Functions。	2022 年 12 月 30 日
新功能	Step Functions 增加了使用新的 Map 状态分布式模式编排大规模并行工作流以进行数据处理的支持。有关更多信息，请参阅 使用分布式模式下的 Map 状态编排大规模并行工作负载。	2022 年 12 月 1 日
新功能	Step Functions 现在支持访问在其他账户中配置的跨账户 AWS 资源。有关更多信息，请参阅 • 在工作流程中访问其他资源 AWS 账户 中的资源 • 教程：访问跨账户资源 AWS • Task state	2022 年 11 月 18 日
更新	Step Functions 现在为查看和调试快速工作流执行提供了全新的控制台体验。有关更多信息，请参阅： • 控制台中的标准和快速工作流执行 • 在 Step Functions 控制台上查看和调试执行	2022 年 10 月 18 日
更新	为 Amazon EMR 优化服务集成添加了支持，以便在使用 `addStep` 和 `addStep.sync` API 时选择性地指定 `ExecutionRoleArn` 参数。有关更多信息，请参阅 使用 Step Functions 调用 Amazon EMR。	2022 年 9 月 20 日

更改	描述	更改日期
仅文档更新	添加了一个新主题，其中提供了有关在使用 Step Functions 构建无服务器工作流时优化成本的建议。有关更多信息，请参阅 使用快速工作流实现成本优化。	2022 年 9 月 15 日
更新	Step Functions 增加了对 14 个新的内置函数的支持，用于执行数据处理任务，如数组操作、数据编码和解码、哈希计算、JSON 数据操作、数学函数运算和唯一标识符生成。 仅文档更新： 根据帮助您执行的数据处理任务类型，将所有现有和新引入的内置函数分为以下几类： • 数组的内置函数 • 用于数据编码和解码的内置函数 • 用于哈希计算的内置函数 • 用于 JSON 数据操作的内置函数 • 用于数学运算的内置函数 • 用于字符串操作的内置函数 • 用于生成唯一标识符的内置函数 • 用于泛型操作的内置函数 有关更多信息，请参阅 内置函数。	2022 年 8 月 31 日
更新	Step Functions 又添加了三项 AWS 服务 — AWS Billing Conductor Amazon GameSparks、和Amazon Pinpoint SMS and Voice V2，从而扩大了对 AWS SDK 集成的支持。有关更多信息，请参阅 更改支持的 AWS SDK 集成的日志。	2022 年 7 月 26 日
仅文档更新	添加了一个新主题，其中包含对 Step Functions 支持的 AWS SDK 集成所做的所有更新的摘要。有关更多信息，请参阅 更改支持的 AWS SDK 集成的日志。	2022 年 7 月 26 日

更改	描述	更改日期
仅文档更新	AWS Step Functions 《开发人员指南》现在包含有关专为 Express Workflows 发布的执行指标的详细信息。有关更多信息，请参阅 快速工作流的执行指标。	2022 年 6 月 9 日
更新	Step Functions 控制台增强功能 Step Functions 控制台中的执行详细信息页面经过重新设计，其中包括以下强功能。 • 能够一目了然地识别执行失败的原因。 • 两种新的状态机可视化模式 – 表格视图和事件视图。您还能使用这些视图进行筛选，来进查看感兴趣的信息。此外，您还可以根据事件时间戳对事件视图的内容进行排序。 • 在图表视图模式中，使用下拉列表在 Map 状态的不同迭代之间切换，或者在表格视图模式的树视图中切换 Map 状态。 • 查看工作流程中每个状态的深入信息，包括完整的输入和输出数据传输路径，以及 Task 或 Parallel 状态的重试尝试次数。 • 其他增强功能包括复制状态机的执行 Amazon 资源名称、查看状态机总转换次数以及以 JSON 格式导出执行详情。 仅文档更新 添加了一个新主题来解释执行详细信息信息页面中显示的各类信息。此外，还添加了一个教程，展示如何检查这些信息。有关更多信息，请参阅： • 在 Step Functions 控制台上查看和调试执行 • 教程：使用 Step Functions 控制台检查状态机执行	2022 年 5 月 9 日

更改	描述	更改日期
更新	Step Functions 现在提供了一种防止混淆代理安全问题的方法，当一个实体（服务或账户）被另一个实体强迫执行操作时，就会出现该问题。有关更多信息，请参阅： • 防止跨服务混淆代理问题	2022 年 5 月 2 日

更新	• Step Functions 又增加了 21 项 AWS 服务，从而扩大了对 AWS SDK 集成的支持。有关更多信息，请参阅：支持的 AWS SDK 服务集成。
	• 仅文档更新：
	  • 添加了在异常中存在的所有异常前缀的列表，这些异常前缀是在您错误地与 Step Functions 执行 AWS SDK 服务集成时生成的。有关更多信息，请参阅：支持的 AWS SDK 服务集成。
	  • 为支持的 AWS SDK 集成添加了所有不支持的 API 操作的列表。有关更多信息，请参阅：支持服务的不支持 API 操作。
	  • 添加了现已弃用的所有受支持的 AWS SDK 集成的列表。有关更多信息，请参阅：已弃用的 AWS SDK 服务集成。
		2022 年 4 月 19 日

新功能	Step Functions Local 现在支持 AWS SDK 集成和模拟服务集成。有关更多信息，请参阅： • 使用模拟服务集成	2022 年 1 月 28 日
新功能	AWS Step Functions 现在支持使用创建 Amazon API Gateway REST API，将同步快速状态机作为后端集成 AWS Cloud Development Kit (AWS CDK)。有关更多信息，请参阅： • 使用同步 Express 状态机创建 API Gateway REST API 使用 AWS CDK	2021 年 12 月 10 日

更改	描述	更改日期
更新	Step Functions 增加了三个新的示例项目，演示了 Step Functions 与 Amazon Athena 升级后的控制台的集成。有关更多信息，请参阅： • 执行多个查询（Amazon Athena、Amazon SNS） • 查询大型数据集（亚马逊 Athena、亚马逊 S3、亚马逊 SN AWS Glue S） • 保持数据最新（亚马逊 Athena、Amazon S3、）AWS Glue	2021 年 11 月 22 日
新功能	Step Functions 已为同步快速工作流添加 Amazon VPC 端点支持。有关更多信息，请参阅： • 用于 Step Functions 的 Amazon VPC 端点	2021 年 11 月 15 日
更新	AWS Step Functions 添加了三个新的示例项目，演示了如何使用 Step Functions AWS Batch 集成。有关更多信息，请参阅： • 分散一份 AWS Batch 工作 • AWS Batch 用 Lambda • 使用 Step Functions 和 AWS Batch 进行错误处理	2021 年 10 月 14 日
新功能	AWS Step Functions 添加了 AWS SDK 集成，允许您使用所有 200 多项 AWS 服务的 API 操作。有关更多信息，请参阅： • AWS SDK 服务集成 • 使用 AWS 软件开发工具包服务集成收集 Amazon S3 存储桶信息	2021 年 9 月 30 日
新功能	AWS Step Functions 添加了可视化工作流设计器，即 AWS Step Functions 工作流工作室。有关更多信息，请参阅： • AWS Step Functions 工作流程工作室 • 学习使用 AWS Step Functions 工作流程工作室	2021 年 6 月 17 日

更改	描述	更改日期
更新	AWS Step Functions 已在 CodeBuild 集成中添加了四个新 API `StartBuildBatch` `StopBuildBatch` `DeleteBuildBatch` 、、`RetryBuildBatch` 和。有关更多信息，请参阅： • AWS CodeBuild 使用 Step Functions 调用	2021 年 6 月 4 日
新功能	AWS Step Functions 现已与 Amazon 集成 EventBridge。有关更多信息，请参阅： • EventBridge 使用 Step Functions 调用 • Step Functions 和 适用于亚马逊的 IAM 政策 EventBridge 的 IAM 策略 • 演示如何将自定义事件发送到 EventBridge 的示例项目	2021 年 5 月 14 日
更新	AWS Step Functions 添加了一个新的示例项目，该项目展示了如何使用 Step Functions 和 Amazon Redshift 数据 API 来运行 ETL/ELT 工作流程。有关更多信息，请参阅： • 使用 Amazon Redshift 运行 ETL/ELT 工作流（Lambda、Amazon Redshift 数据 API）	2021 年 4 月 16 日
新功能	AWS Step Functions 控制台中有一个新的数据流模拟器。有关更多信息，请参阅： • Step Functions 控制台	2021 年 4 月 8 日
新功能	AWS Step Functions 现在在 EKS 上与 Amazon EMR 集成。有关更多信息，请参阅： • 使用以下方式在 EKS 上致电 Amazon EMR AWS Step Functions	2021 年 3 月 29 日

更改	描述	更改日期
更新	AWS Toolkit for Visual Studio Code 和 AWS CloudFormation添加了对状态机定义的 YAML 支持 有关更多信息，请参阅： • 定义格式支持 • AWS Toolkit for Visual Studio Code	2021 年 3 月 4 日
新功能	AWS Step Functions 现在与集成 AWS Glue DataBrew。有关更多信息，请参阅： • 使用 Step Functions 管理 AWS Glue DataBrew 作业 • 什么是 AWS Glue DataBrew？ 在 DataBrew 开发者指南中。	2021 年 1 月 8 日
新功能	AWS Step Functions 同步 Express 工作流程现已推出，让您可以轻松地编排微服务。有关更多信息，请参阅： • 同步和异步快速工作流 • 演示如何调用同步快速工作流 的示例项目 • StartSyncExecutionAPI 文档。	2020 年 11 月 24 日
新功能	AWS Step Functions 现已与亚马逊 API Gateway 集成。有关更多信息，请参阅： • 使用 Step Functions 调用 API Gateway • Step Functions 和 亚马逊 API Gateway 的 IAM 政策 的 IAM 策略 • 演示如何调用 API Gateway 的示例项目	2020 年 11 月 17 日
新功能	AWS Step Functions 现在已与亚马逊 Elastic Kubernetes Service 集成。有关更多信息，请参阅： • 使用 Step Functions 调用 Amazon EKS • Step Functions 和 亚马逊 EKS 的 IAM 政策 的 IAM 策略 • 演示如何管理 Amazon EKS 集群 的示例项目	2020 年 11 月 16 日

更改	描述	更改日期
新功能	AWS Step Functions 现已与亚马逊 Athena 集成。有关更多信息，请参阅： • 使用 Step Functions 调用 Athena • Step Functions 和 亚马逊 Athena 的 IAM 政策 的 IAM 策略 • 演示如何启动 Athena 查询 的示例项目	2020 年 10 月 22 日
新功能	AWS Step Functions 现在支持使用跟踪 end-to-end 工作流程 AWS X-Ray，使您可以全面了解状态机执行情况，并使分析和调试分布式应用程序变得更加容易。有关更多信息，请参阅： • AWS X-Ray 和 Step Functions • Step Functions 和 适用的 IAM 政策 AWS X-Ray 的 IAM 策略 • AWS Step Functions API 引用 • TracingConfiguration	2020 年 9 月 14 日

更改	描述	更改日期

更新	AWS Step Functions 现在支持作为 UTF-8 编码字符串的有效载荷大小不超过 256 KB 的数据。这样，您就可以在标准和快速工作流中处理更大的有效负载。
	
	无需更改现有的状态机即可使用更大的有效负载。但是，您需要更新到最新版本的 Step Functions SDK 和 Local Runner 才能使用更新后的 API。有关更多信息，请参阅：
	
	• 配额
	• the section called “使用 Amazon S3 ARN 而不是传递大量有效负载”
	• States.DataLimitExceeded
	• the section called “CloudWatch Logs 有效负载”
	• the section called “EventBridge 有效载荷”
	• AWS Step Functions API 引用
	
	  • CloudWatchEventsExecutionDataDetails
	  • HistoryEventExecutionDataDetails
	  • GetExecutionHistory
	  • ActivityScheduledEventDetails
	  • ActivitySucceededEventDetails
	  • CloudWatchEventsExecutionDataDetails
	  • ExecutionSucceededEventDetails
	  • LambdaFunctionScheduledEventDetails
	  • ExecutionSucceededEventDetails
	  • StateEnteredEventDetails
	  • StateExitedEventDetails
	  • TaskSubmittedEventDetails
	  • TaskSucceededEventDetails
		2020 年 9 月 3 日

更改	描述	更改日期

更新	Amazon States Language 已更新，如下所示：
	• 已添加 选项规则
	  • 空比较运算符，IsNull。针对 JSON 空值进行 IsNull 测试，可用于检测先前状态的输出是否为空。
	  • 添加了另外四个新运算符 IsBoolean、 IsNumeric、IsString 和 IsTimestamp。
	  • 使用 IsPresent 运算符测试字段是否存在。IsPresent 可用于防止有人尝试访问不存在的密钥时出现 States.Runtime 错误。
	  • 通配符模式匹配，支持与包含一个或多个通配符的模式进行字符串比较。
	  • 支持比较运算符的两个变量之间的比较。
	• 现在可以从状态输入中动态提供 Task 状态下的超时值和信号检测值，而无需使用 TimeoutSecondsPath 和 HeartbeatSecondsPath 字段提供固定值。有关更多信息，请参阅 任务状态 状态。
	• 新的 ResultSelector 字段提供了一种在应用 ResultPath 状态之前操作状态结果的方法。ResultSelector 字段是 Map、Parallel 和 任务状态 状态下的可选字段。
	• 内置函数 已添加以允许在没有 Task 状态的情况下进行基本操作。内置函数可以在 Parameters 和 ResultSelector 字段中使用。
		2020 年 8 月 13 日

更新	AWS Step Functions 现在支持亚马逊 SageMaker CreateProcessingJob API 调用。有关更多信息，请参阅： • SageMaker 使用 Step Functions 进行管理 • 预处理数据并训练机器学习模型，一个演示 CreateProcessingJob 的示例项目。	2020 年 8 月 4 日

更改	描述	更改日期
新功能	AWS Step Functions 现在受支持 AWS Serverless Application Model，可以更轻松地将工作流程编排集成到您的无服务器应用程序中。有关更多信息，请参阅： • AWS Step Functions 和 AWS SAM • AWS::Serverless::StateMachine • AWS SAM 策略模板	2020 年 5 月 27 日
新功能	AWS Step Functions 为嵌套 Step Functions 执行引入了新的同步调用。新调用 `arn:aws:states:::states:startExecution.sync:2` 会返回 JSON 对象。原有调用 `arn:aws:states:::states:startExecution.sync` 可以继续使用，它会返回 JSON 转义的字符串。有关更多信息，请参阅： • 将 AWS Step Functions 执行作为集成服务进行管理	2020 年 5 月 19 日
新功能	AWS Step Functions 现在与集成 AWS CodeBuild。有关更多信息，请参阅： • 与其他服务 AWS Step Functions 一起使用 • AWS CodeBuild 使用 Step Functions 调用 • Step Functions 的优化集成	2020 年 5 月 5 日
新功能	AWS Toolkit for Visual Studio Code 现已支持 Step Functions，无需离开代码编辑器即可更轻松地创建和可视化基于状态机的工作流。	2020 年 3 月 31 日
更新	现在，您可以为标准工作流程配置对 Amazon CloudWatch 日志的日志记录。有关更多信息，请参阅： • 使用 CloudWatch Logs 进行日志记录	2020 年 2 月 25 日

更改	描述	更改日期
新功能	AWS Step Functions 现在无需公有 IP 地址即可直接从亚马逊虚拟私有云 (VPC) 进行访问。有关更多信息，请参阅： • 用于 Step Functions 的 Amazon VPC 端点	2019 年 12 月 23 日

更改	描述	更改日期

新功能	快速工作流是一种新的工作流类型，适用于大批量事件处理工作负载，例如 IoT 数据摄取、流数据处理和转换以及移动应用程序后端。
	
	有关详细信息，请查看以下新主题和更新主题。
	
	• 标准和快速工作流
	  • 执行保障
	• 与其他服务 AWS Step Functions 一起使用
	  • Step Functions 的优化集成
	• 处理来自 Amazon SQS（快速工作流）的大批量消息
	• 选择性检查点示例（快速工作流）
	• 配额
	  • 配额
	• 使用 CloudWatch Logs 进行日志记录
	• AWS Step Functions API 引用
	  • CreateStateMachine
	  • UpdateStateMachine
	  • DescribeStateMachine
	  • DescribeStateMachineForExecution
	  • StopExecution
	  • DescribeExecution
	  • GetExecutionHistory
	  • ListExecutions
	  • ListStateMachines
	  • StartExecution
	  • CloudWatchLogsLogGroup
	  • LogDestination
	  • LoggingConfiguration
		2019 年 12 月 3 日

更改	描述	更改日期
新功能	AWS Step Functions 现已与 Amazon EMR 集成。有关更多信息，请参阅： • 与其他服务 AWS Step Functions 一起使用 • 使用 Step Functions 调用 Amazon EMR • Step Functions 的优化集成	2019 年 11 月 19 日

更新	AWS Step Functions 已经发布了 Ste AWS p Functions 数据科学 SDK。有关更多信息，请参阅下列内容。
	• GitHub 上的项目
	• 开发工具包文档
	• 以下示例笔记本可在SageMaker控制台和相关GitHub 项目中找到。
	  • `hello_world_workflow.ipynb`
	  • `machine_learning_workflow_abalone.ipynb`
	  • `training_pipeline_pytorch_mnist.ipynb`
		2019 年 11 月 7 日

更新	Step Functions 现在支持更多适用于亚马逊的 API 操作 SageMaker，并包括两个新的示例项目来演示该功能。有关更多信息，请参阅下列内容。 • SageMaker 使用 Step Functions 进行管理 • 与其他服务 AWS Step Functions 一起使用 • 训练机器学习模型 • 调整机器学习模型	2019 年 10 月 3 日

更改	描述	更改日期
新特征	Step Functions 支持通过调用 `StartExecution` 作为集成服务 API 来启动新的工作流程执行。请参阅： • 从 Task 状态启动工作流执行 • 将 AWS Step Functions 执行作为集成服务进行管理 • 与其他服务 AWS Step Functions 一起使用 • 用于启动 Step Functions 工作流执行的 IAM 策略	2019 年 8 月 12 日
新功能	Step Functions 包括将任务令牌传递给集成服务，并暂停执行，直到使用 `SendTaskSuccess` 或 `SendTaskFailure` 返回该任务令牌的功能。请参阅： • 服务集成模式 • 等待具有任务令牌的回调 • 回调模式示例（Amazon SQS、Amazon SNS、Lambda） • Step Functions 的优化集成 • 部署示例人员批准项目 • 服务集成指标 Step Functions 现在提供了一种直接在状态定义的 `"Parameters"` 字段中访问有关当前执行的动态信息的方法。请参阅： • Context 对象 • 将上下文对象节点作为参数传递	2019 年 5 月 23 日
新功能	Step Functions 支持执行状态更改 CloudWatch 事件，请参阅： • EventBridge Step Functions 的（CloudWatch 事件）执行状态发生了变化 • Amazon CloudWatch 活动用户指南	2019 年 5 月 8 日

更改	描述	更改日期
新功能	Step Functions 支持使用标签的 IAM 权限。有关更多信息，请参阅： • Step Functions 中的标记 • 基于标签的策略	2019 年 3 月 5 日
新功能	Step Functions Local 现已推出。您可以在本地计算机上运行 Step Functions 以进行测试和开发。Step Functions Local 可以 Java 应用程序或 Docker 映像的方式下载。请参阅 在本地测试状态机。	2019 年 2 月 4 日
新功能	AWS Step Functions 现已在北京和宁夏区域推出。请参阅 支持的 区域。	2018 年 1 月 15 日
新功能	Step Functions 支持资源标记以帮助跟踪您的成本分配。您可以在详细信息页面上或通过 API 操作来标记状态机。请参阅 Step Functions 中的标记。	2019 年 1 月 7 日
新功能	AWS Step Functions 现已在欧洲（巴黎）和南美洲（圣保罗）地区推出。请参阅 支持的 区域。	2018 年 12 月 13 日
新功能	AWS Step Functions 现已在欧洲（斯德哥尔摩）地区推出。有关受支持的区域的列表，请参阅支持的 区域。	2018 年 12 月 12 日
新特征	Step Functions 现在与某些 AWS 服务集成。您现在可以从 Amazon States Language 中的任务状态直接调用这些集成服务的 API 并将参数传递给它们。有关更多信息，请参阅： • 与其他服务 AWS Step Functions 一起使用 • 将参数传递给服务 API • Step Functions 的优化集成	2018 年 11 月 29 日

更改	描述	更改日期
更新	改进了任务状态文档中 `TimeoutSeconds` 和 `HeartbeatSeconds` 的描述。请参阅 任务状态。	2018 年 10 月 24 日
更新	改善了对最大执行历史记录大小限制的描述，并提供了一个指向相关最佳实操主题的链接。 • 与状态机执行相关的配额 • 避免达到历史记录的配额	2018 年 10 月 17 日
更新	在 AWS Step Functions 文档中添加了新教程：请参阅启动状态机执行以响应 Amazon S3 事件。	2018 年 9 月 25 日
更新	从限制文档中删除了条目 Step Functions 控制台中显示的最大执行数。请参阅 配额。	2018 年 9 月 13 日
更新	在 AWS Step Functions 文档中添加了有关在轮询活动任务时缩短延迟的最佳实践主题。请参阅 避免轮询活动任务时发生延迟。	2018 年 8 月 30 日
更新	改进了有关活动和活动工作人员 AWS Step Functions 的主题。请参阅 活动。	2018 年 8 月 29 日
更新	改进了有关 CloudTrail 集成的 AWS Step Functions 主题。请参阅 使用录制 API 调用 AWS CloudTrail。	2018 年 8 月 7 日
更新	在 AWS CloudFormation 教程中添加了 JSON 示例。请参阅 使用 AWS CloudFormation为 Step Functions 创建一个 Lambda 状态机。	2018 年 23 月 6 日
更新	增加了有关处理 Lambda 服务错误的新主题。请参阅 处理 Lambda 服务异常。	2018 年 6 月 20 日
新功能	AWS Step Functions 现已在亚太地区（孟买）地区推出。有关受支持的区域的列表，请参阅支持的 区域。	2018 年 6 月 28 日

更改	描述	更改日期
新功能	AWS Step Functions 现已在 AWS GovCloud （美国西部）地区推出。有关受支持的区域的列表，请参阅支持的 区域。有关在 AWS GovCloud （美国西部）地区使用 Step Functions 的信息，请参阅AWS GovCloud (US)。	2018 年 6 月 28 日
更新	改进了有关 `Parallel` 状态错误处理的文档。请参阅 错误处理。	2018 年 6 月 20 日
更新	改进了有关 Step Functions 中输入和输出处理的文档。了解如何使用 `InputPath` 、`ResultPath` 和 `OutputPath` 控制 JSON 在工作流程、状态和任务中的流动。请参阅： • Step Functions 中的输入和输出处理 • ResultPath	2018 年 6 月 7 日
更新	改进了 Parallel 状态的代码示例。请参阅 Parallel。	2018 年 4 月 6 日
新功能	现在，您可以在中监控 API 和服务指标 CloudWatch。请参阅 使用监控 Step Functions CloudWatch。	2018 年 5 月 25 日
更新	`StartExecution` 、`StopExecution` 和 `StateTransition` 现在在以下区域中提高了限制： • 美国东部（弗吉尼亚州北部） • 美国西部（俄勒冈州） • 欧洲地区（爱尔兰） 有关更多信息，请参阅 配额。	2018 年 5 月 16 日
新功能	AWS Step Functions 现已在美国西部（加利福尼亚北部）和亚太地区（首尔）地区推出。有关受支持的区域的列表，请参阅支持的 区域。	2018 年 5 月 5 日
更新	更新过程和图像，以便与界面更改相符。	2018 年 4 月 25 日

更改	描述	更改日期
更新	添加了新的教程，展示如何启动新执行以继续您的工作。请参阅 将长时间运行的工作流执行作为新执行继续执行。本教程介绍了一种设计模式，有助于避免一些服务限制。请参阅 避免达到历史记录的配额。	2018 年 4 月 19 日
更新	通过添加有关状态机的概念信息，改进了状态文档的介绍。请参阅 状态。	2018 年 3 月 9 日
更新	除了 HTML、PDF 和 Kindle 之外，《AWS Step Functions 开发者指南》还可在上找到 GitHub。要留下反馈，请选择右上角的 GitHub图标。 	2018 年 3 月 2 日
更新	增加了一个描述与 Step Functions 相关的其他资源的主题。 请参阅 相关信息。	2018 年 2 月 20 日

新功能	• 当您创建新的状态机时，您必须确认 AWS Step Functions 将创建一个 IAM 角色，此角色可用于访问您的 Lambda 函数。
	• 更新了以下教程以反映状态机创建工作流程中的微小更改：
	  • 创建使用 Lambda 的 Step Functions 状态机
	  • 使用 Step Functions 创建活动状态机
	  • 使用状态机处理 Step Functions 函数错误情形
	  • 使用 Lambda 迭代循环
		2018 年 2 月 19 日

更新	添加了一个主题，介绍使用 Ruby 编写的示例活动工作线程。可以使用此实现直接创建 Ruby 活动工作线程，也可以将其作为一种设计模式来使用其他语言创建活动工作线程。 请参阅 使用 Ruby 编写的示例活动工作线程。	2018 年 2 月 6 日

更改	描述	更改日期
更新	添加了一个新教程，介绍使用 Lambda 函数迭代计数的设计模式。 请参阅 创建使用 Lambda 的 Step Functions 状态机。	2018 年 1 月 31 日
更新	更新了 IAM 权限的相关内容，加入了 `DescribeStateMachineForExecution` 和 `UpdateStateMachine` API 的说明。 请参阅 为非管理员用户创建精细的 IAM 权限。	2018 年 1 月 26 日
更新	增加了新推出的区域：加拿大（中部）、亚太地区（新加坡）。 请参阅 支持的 区域。	2018 年 1 月 25 日
更新	更新了教程和过程，以反映 IAM 允许您选择 Step Functions 作为角色。	2018 年 1 月 24 日
更新	添加了一个新的最佳实操 主题：建议不要在状态间传递大量负载。 请参阅 使用 Amazon S3 ARN 而不是传递大量有效负载。	2018 年 1 月 23 日
更新	更正了过程，以匹配更新后的“创建状态机”界面： • 创建使用 Lambda 的 Step Functions 状态机 • 使用 Step Functions 创建活动状态机 • 使用状态机处理 Step Functions 函数错误情形	2018 年 1 月 17 日

更改	描述	更改日期
新功能	您可以使用示例项目 快速预置状态机和所有相关的 AWS 资源。请参阅Step Functions 的示例项目， 可用示例项目包括： • 任务状态投票 (Lambda,) AWS Batch • 任务计时器（Lambda、Amazon SNS） **Note** 这些示例项目和相关文档将取代描述实现同一功能的教程。	2018 年 1 月 11 日
更新	添加了最佳实操 部分，其中包含有关避免执行卡顿的信息。请参阅 Step Functions 的最佳实操。	2018 年 1 月 5 日
更新	增加了有关重试对定价的影响的注释： **Note** 重试被视为一种状态转换。有关状态转换如何影响计费的信息，请参阅 Step Functions 定价。	2017 年 12 月 8 日
更新	增加了与资源名称相关的信息： **Note** Step Functions 允许您为状态机、执行和活动创建名称，以及包含非 ASCII 字符的标签。这些非 ASCII 名称不适用于亚马逊。 CloudWatch为确保您可以跟踪 CloudWatch 指标，请选择仅使用 ASCII 字符的名称。	2017 年 12 月 6 日

更改	描述	更改日期
更新	改进了安全概述信息并且添加了有关精细 IAM 权限的主题。请参阅安全性 AWS Step Functions和为非管理员用户创建精细的 IAM 权限。	2017 年 11 月 27 日
新功能	您可以更新现有状态机。请参见更新您的状态机。	2017 年 11 月 15 日
更新	添加了一个用于阐明 Lambda.Unknown 错误的注释并链接到了以下部分中的 Lambda 文档： • 错误名称 • 第 3 步：创建使用 Catch 字段的状态机 Note Lambda 中未处理的错误在错误输出中报告为 Lambda.Unknown 。其中包括 out-of-memory 错误和函数超时。您可以匹配 Lambda.Unknown 、States.ALL 或 States.TaskFailed 来处理这些错误。当 Lambda 达到最大调用次数，会出现 Lambda.TooManyRequestsException 错误。有关 Lambda 函数错误的更多信息，请参阅《AWS Lambda 开发者指南》中的错误处理和自动重试。	2017 年 10 月 17 日
更新	更正和阐明了 IAM 说明并更新了所有教程中的屏幕截图。	2017 年 10 月 11 日

更改	描述	更改日期

更新	• 为状态机执行结果添加了新的屏幕截图以体现 Step Functions 控制台中的更改。重新编写了以下教程中的 Lambda 说明以体现 Lambda 控制台中的更改：
	  • 创建使用 Lambda 的 Step Functions 状态机
	  • 创建作业状态轮询器
	  • 创建任务计时器
	  • 使用状态机处理 Step Functions 函数错误情形
	• 在以下部分中更正和阐明了有关状态机创建的信息：
	  • 使用 Step Functions 创建活动状态机
		2017 年 10 月 6 日

更新	重新编写了以下部分中的 IAM 说明以体现 IAM 控制台中的更改： • 为状态机创建 IAM 角色 • 创建使用 Lambda 的 Step Functions 状态机 • 创建作业状态轮询器 • 创建任务计时器 • 使用状态机处理 Step Functions 函数错误情形 • 使用 API Gateway 创建 Step Functions API	2017 年 10 月 5 日
更新	重新编写了状态机数据部分。	2017 年 9 月 28 日
新功能	在 Step Functions 可用的所有区域中增加了与 API 操作限制相关的局限性。	2017 年 9 月 18 日
更新	• 更正和阐明了所有教程中有关启动新执行的信息。 • 更正和阐明了与账户相关的配额部分中的信息。	2017 年 9 月 14 日

更改	描述	更改日期
更新	重新编写了以下教程以体现 Lambda 控制台中的更改： • 创建使用 Lambda 的 Step Functions 状态机 • 使用状态机处理 Step Functions 函数错误情形 • 创建作业状态轮询器	2017 年 8 月 28 日
新功能	Step Functions 已在欧洲地区（伦敦）推出。	2017 年 23 月 8 日
新功能	状态机的可视化工作流让您可以放大、缩小图表以及将图表居中。	2017 年 8 月 21 日
新功能	**Important** 一个执行无法使用另一执行的名称 90 天。 当您使用相同的名称进行多个`StartExecution` 调用时，新的执行不会运行。 有关更多信息，请参阅《AWS Step Functions API 参考》中 `StartExecution` API 操作的 `name` 请求参数。	2017 年 8 月 18 日
更新	在 使用 API Gateway 创建 Step Functions API 教程中增加了有关传递状态机 ARN 的替代方法的信息。	2017 年 17 月 8 日
更新	添加了新的创建任务状态轮询器 教程。	2017 年 8 月 10 日
新功能	• Step Functions 会发出`ExecutionThrottled` CloudWatch 指标。有关更多信息，请参阅 使用监控 Step Functions CloudWatch。 • 增加了与状态限制相关的配额部分。	2017 年 8 月 3 日
更新	更新了 第 1 步：为 API Gateway 创建 IAM 角色部分中的说明。	2017 年 18 月 7 日

更改	描述	更改日期
更新	更正和阐明了Choice部分中的信息。	2017 年 6 月 23 日
更新	在以下教程中添加了有关使用其他 AWS 账户下资源的信息： • 创建使用 Lambda 的 Step Functions 状态机 • 使用 AWS CloudFormation为 Step Functions 创建一个 Lambda 状态机 • 使用 Step Functions 创建活动状态机 • 使用状态机处理 Step Functions 函数错误情形	2017 年 6 月 22 日
更新	更正和阐明了以下部分中的信息： • 使用状态机处理 Step Functions 函数错误情形 • 状态 • Step Functions 中的错误处理	2017 年 6 月 21 日
更新	重新编写了所有教程以匹配 Step Functions 控制台更新。	2017 年 6 月 12 日
新功能	Step Functions 现已在亚太地区（悉尼）推出。	2017 年 6 月 8 日
更新	已重构Amazon States Language 部分。	2017 年 6 月 7 日
更新	更正和阐明了使用 Step Functions 创建活动状态机部分中的信息。	2017 年 6 月 6 日
更新	更正了使用 Retry 和使用 Catch 的状态机示例部分中的代码示例。	2017 年 6 月 5 日
更新	使用 AWS 文档标准重组了本指南。	2017 年 5 月 31 日

更改	描述	更改日期
更新	更正和阐明了Parallel部分中的信息。	2017 年 5 月 25 日
更新	将“路径和筛选条件”部分合并到了 Step Functions 中的输入和输出处理部分。	2017 年 5 月 24 日
更新	更正和阐明了使用监控 Step Functions CloudWatch部分中的信息。	2017 年 5 月 15 日
更新	更新了使用 Step Functions 创建活动状态机教程中的 `GreeterActivities.java` 工作线程代码。	2017 年 5 月 9 日
更新	在 什么是 AWS Step Functions？ 部分增加了介绍性视频。	2017 年 4 月 19 日
更新	更正和阐明了以下教程中的信息： • 创建使用 Lambda 的 Step Functions 状态机 • 使用 Step Functions 创建活动状态机 • 使用状态机处理 Step Functions 函数错误情形	2017 年 4 月 19 日
更新	在创建使用 Lambda 的 Step Functions 状态机和使用状态机处理 Step Functions 函数错误情形教程中添加了有关 Lambda 模板的信息。	2017 年 4 月 6 日
更新	将“最大输入或结果数据大小”限制更改为“任务、状态或执行的最大输入或结果数据大小”(32768 个字符)。有关更多信息，请参阅 与任务执行相关的配额。	2017 年 3 月 31 日
新功能	• Step Functions 支持通过将 Step Functions 设置为亚马逊 CloudWatch 事件目标来执行状态机。	2017 年 3 月 21 日
新功能	• Step Functions 允许将 Lambda 函数错误处理作为首选错误处理方法。 • 更新了使用状态机处理 Step Functions 函数错误情形教程和Step Functions 中的错误处理部分。	2017 年 3 月 16 日

更改	描述	更改日期
新功能	Step Functions 现已在欧洲地区（法兰克福）推出。	2017 年 3 月 7 日
更新	在目录中重新组织了主题，并更新了以下教程： • 创建使用 Lambda 的 Step Functions 状态机 • 使用 Step Functions 创建活动状态机 • 使用状态机处理 Step Functions 函数错误情形	2017 年 2 月 23 日
新功能	• Step Functions 控制台的状态机页面包含复制到新的和删除按钮。 • 更新了屏幕截图以匹配控制台更改。	2017 年 2 月 23 日
新功能	• Step Functions 支持使用 API Gateway 创建 API。 • 增加了使用 API Gateway 创建 Step Functions API教程。	2017 年 2 月 14 日
新功能	• Step Functions 支持与集成 AWS CloudFormation。 • 增加了使用 AWS CloudFormation为 Step Functions 创建一个 Lambda 状态机教程。	2017 年 2 月 10 日
更新	阐明了 `ResultPath` 和 `OutputPath` 字段与 `Parallel` 状态相关时的行为。	2017 年 2 月 6 日
更新	• 在教程中阐明了状态机命名限制。 • 更正了一些代码示例。	2017 年 1 月 5 日
更新	更新了 Lambda 函数示例以使用最新的编程模型。	2016 年 12 月 9 日
初始版本	的初始版本 AWS Step Functions.	2016 年 12 月 1 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。