The art of the state: Coordinating services using AWS Step Functions

Gabe Hollombe

Senior Developer Advocate
Amazon Web Services

aws SUMMIT ONLINE

reliable, automated way of orchestrating very complex queries and processes between all our distributed systems, saved time and money more productivity and agility. easier when discussing the solution with nontechnical stakeholders

The Guardian

"AWS Step Functions gives us a **reliable, automated way of orchestrating very complex queries and processes between all our distributed systems**," Brown says. "We **saved time and money** by making it easy for our developers to build applications using AWS Lambda functions, giving them **more productivity and agility**. We also get a visual representation of the logic for each workflow, which makes it **easier when discussing the solution with nontechnical stakeholders** at the company."

Paul Brown

Senior Developer Manager

What we'll cover in this session

Background and challenges around working with distributed services

Introducing the concept of orchestration

Service orchestration made easy using state machines

AWS Step Functions: State machines in the cloud

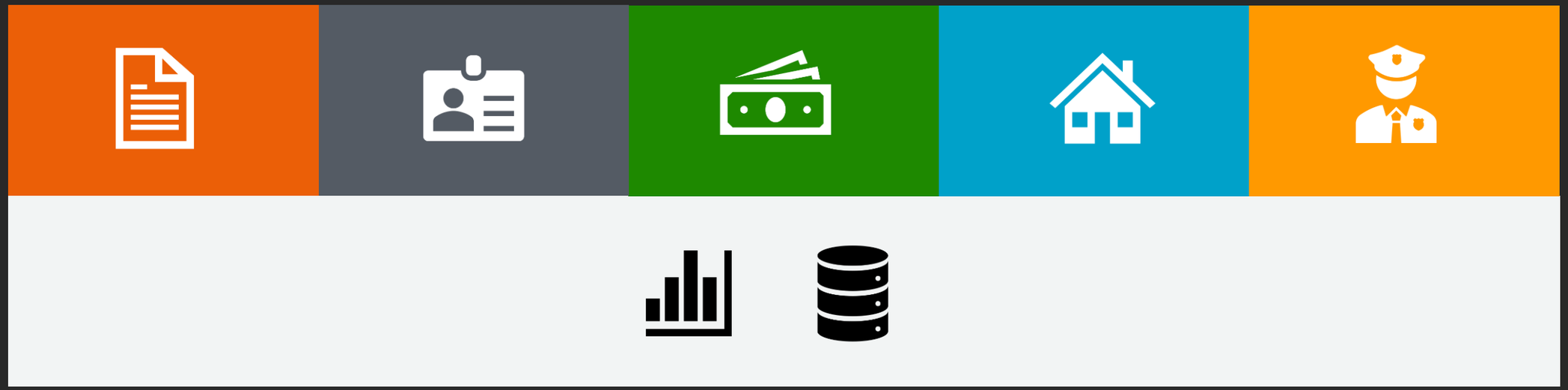
Examples from the real world

Where to learn more

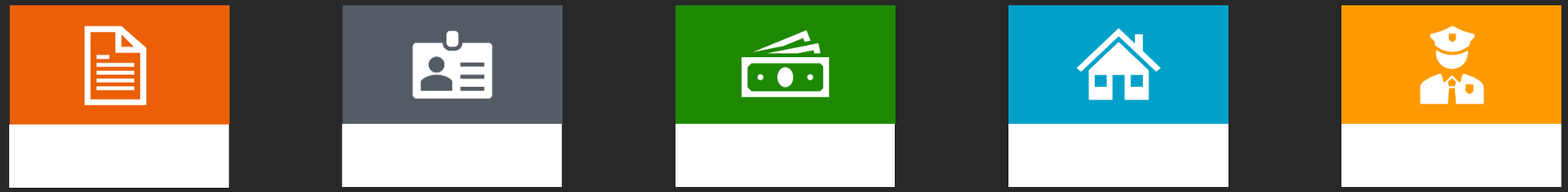
Getting things done

aws SUMMIT ONLINE

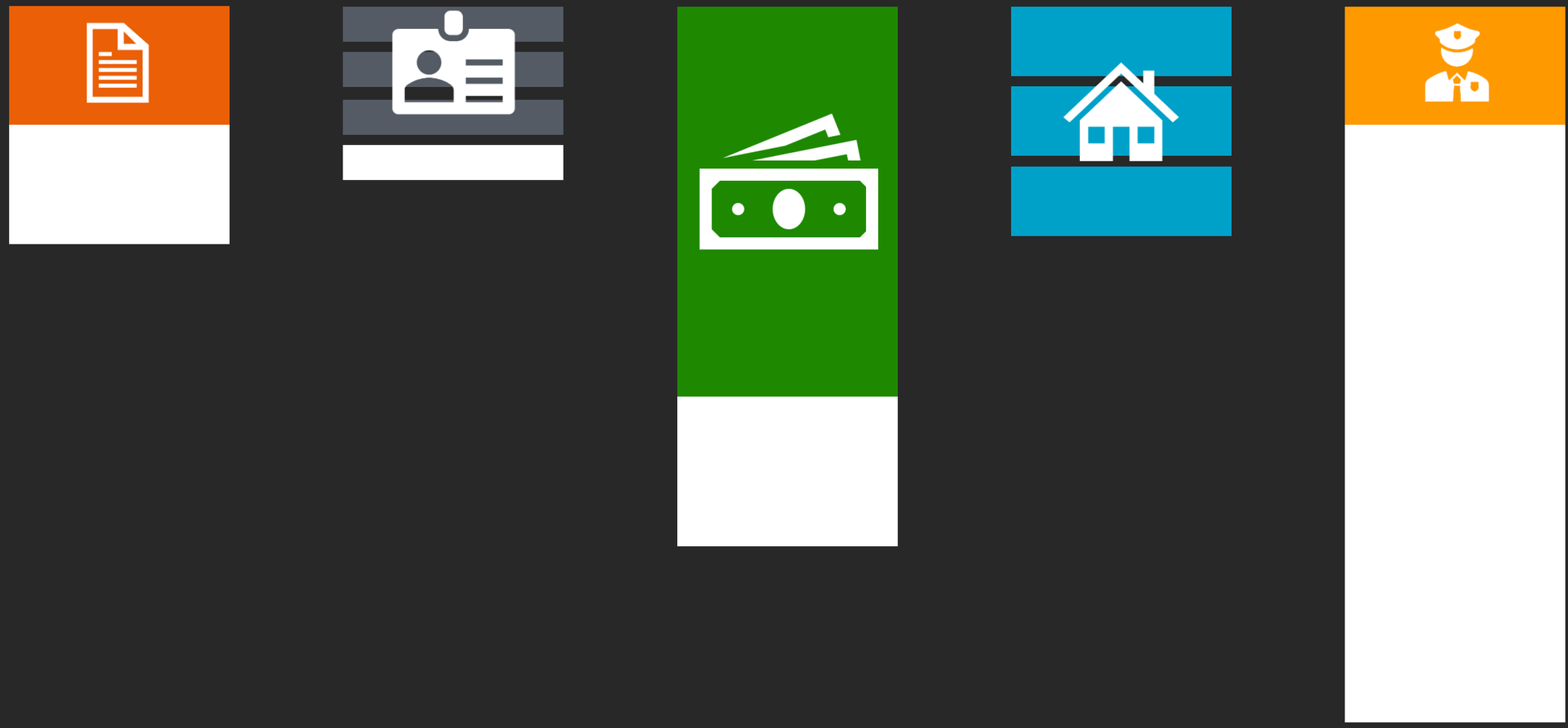
In a monolith, everything gets deployed together



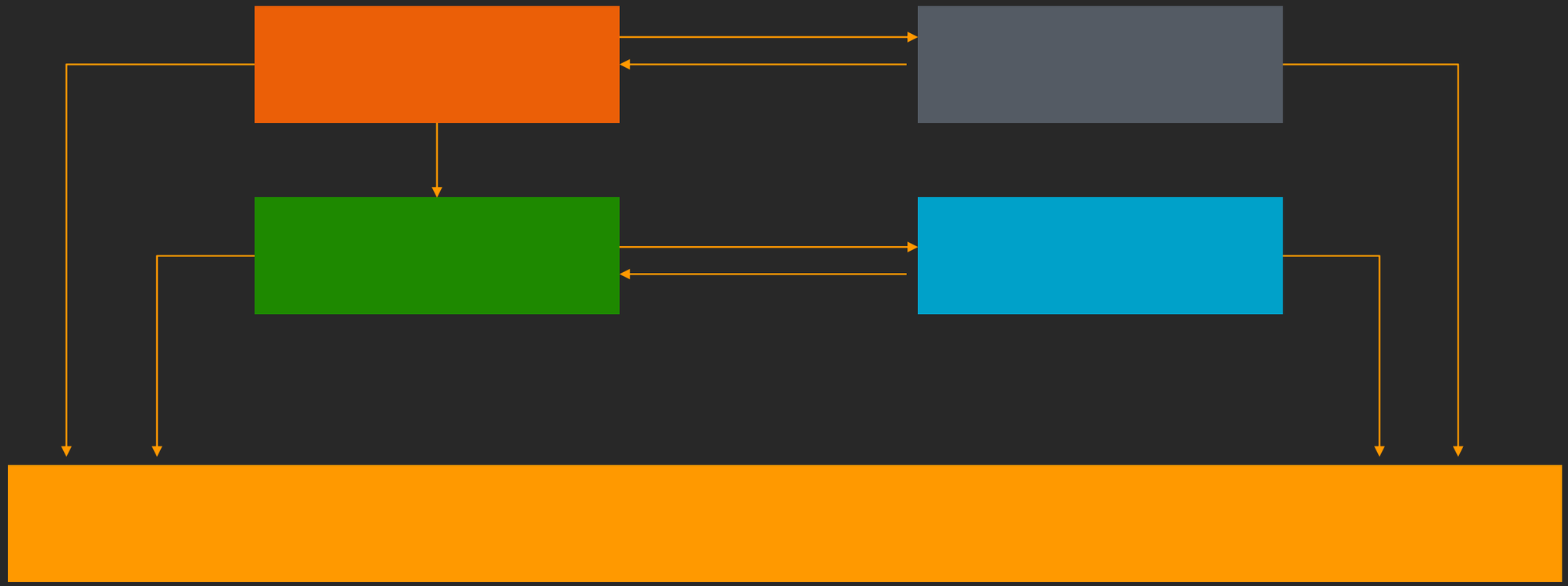
With microservices, we split the work between multiple systems



Microservices can give us increased agility and scalability

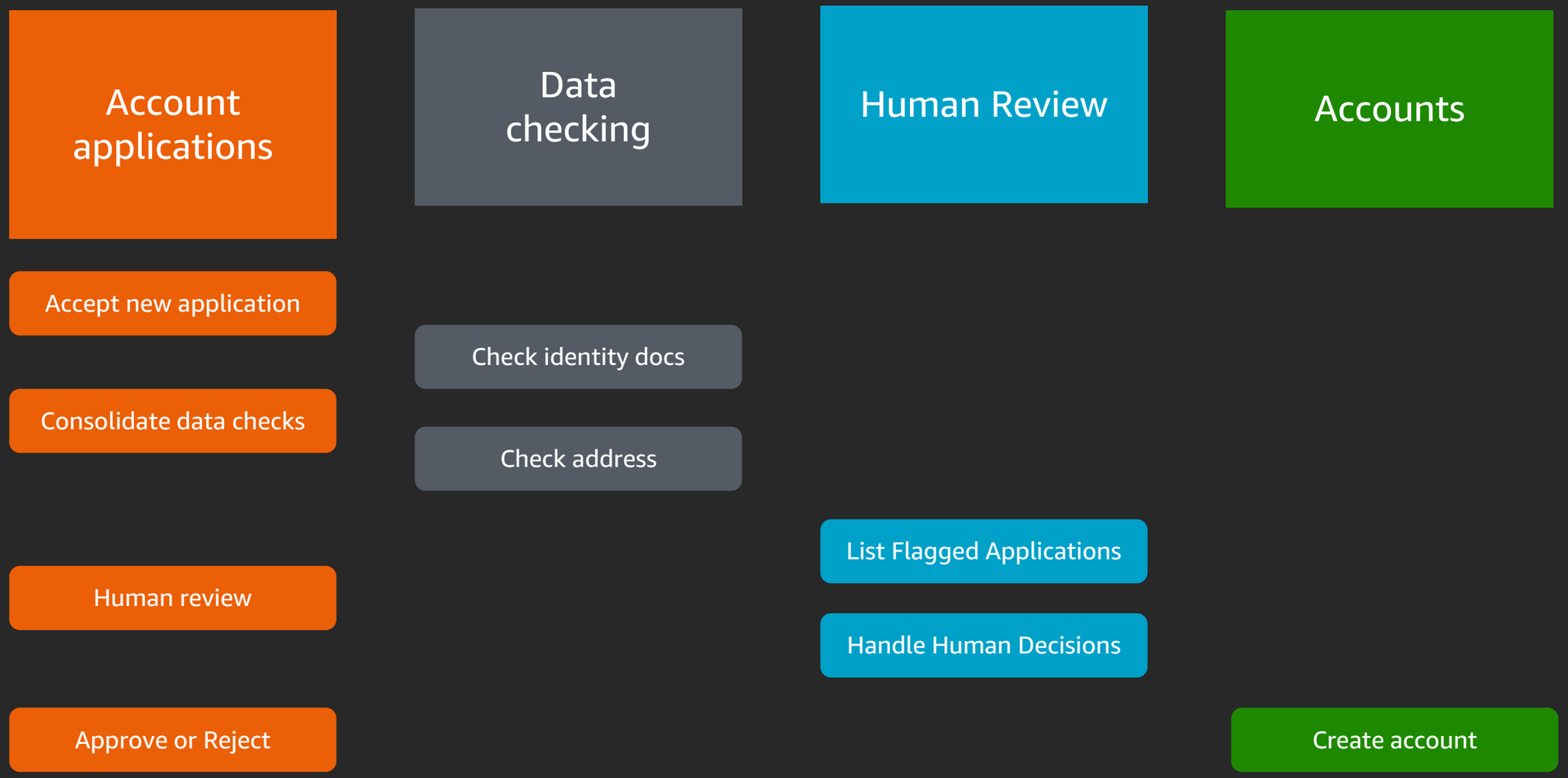


But distributed systems can be harder to coordinate and debug

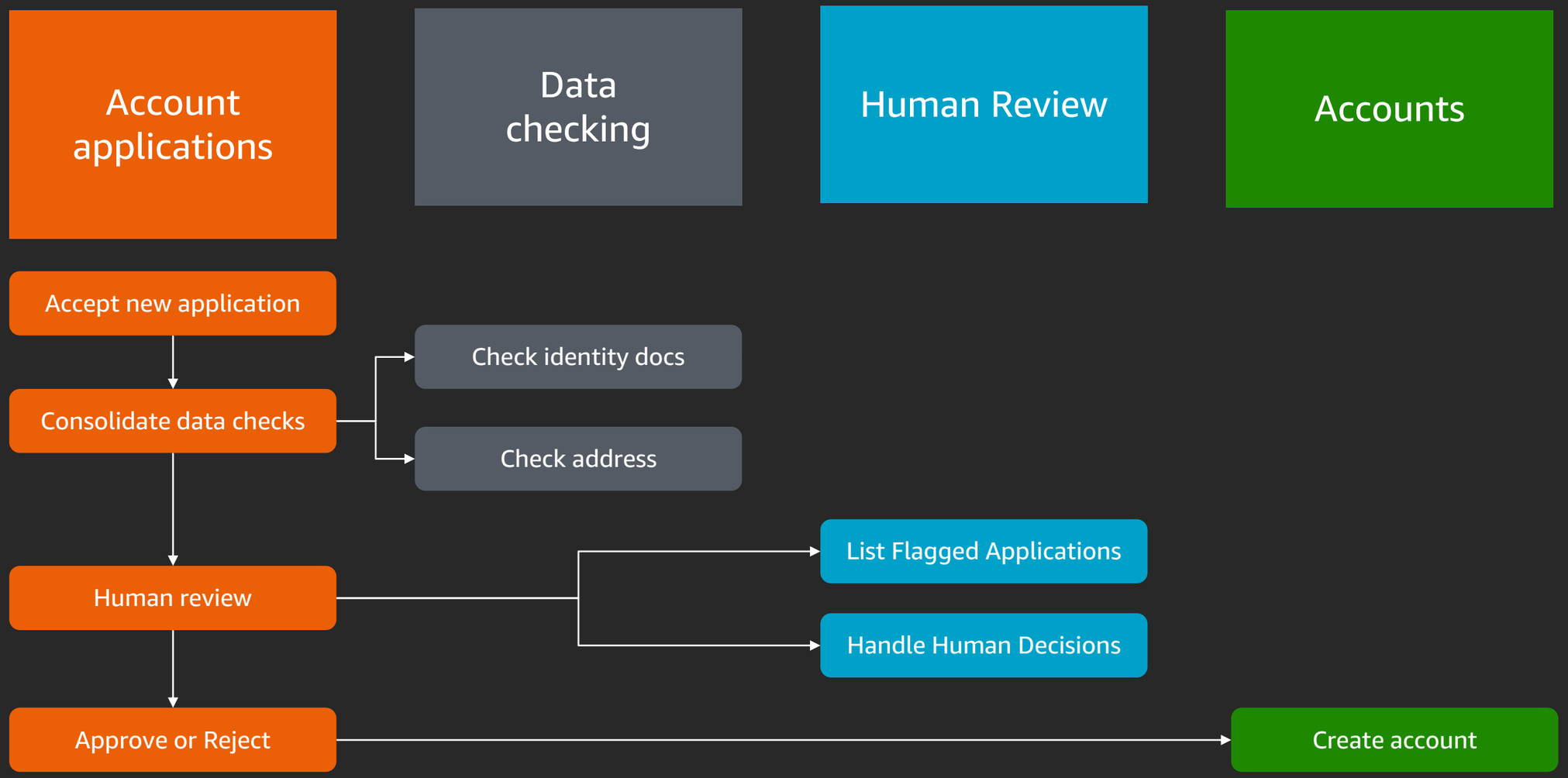


Introducing orchestration

Here's a simplified banking system

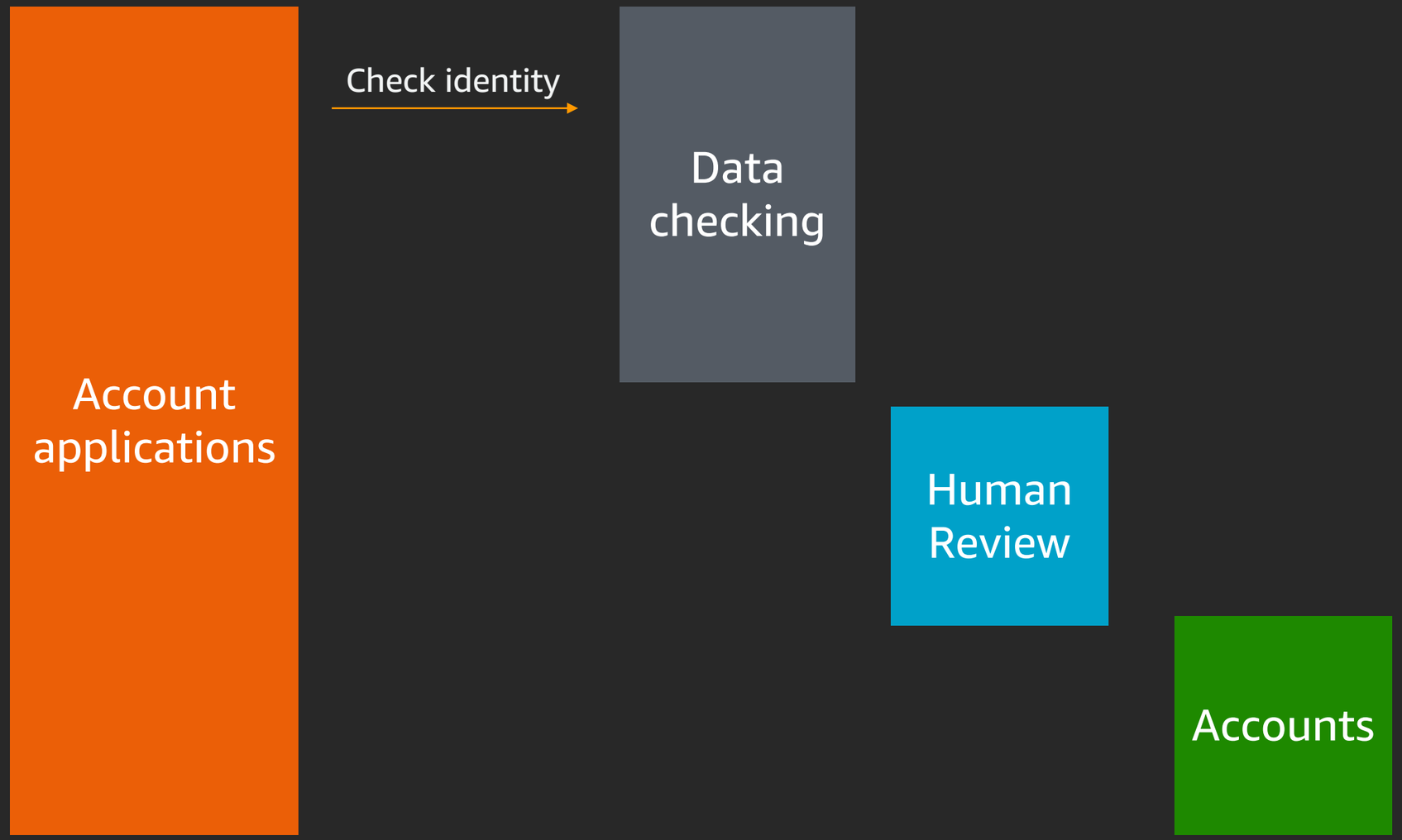


Processing a new account application requires some coordination

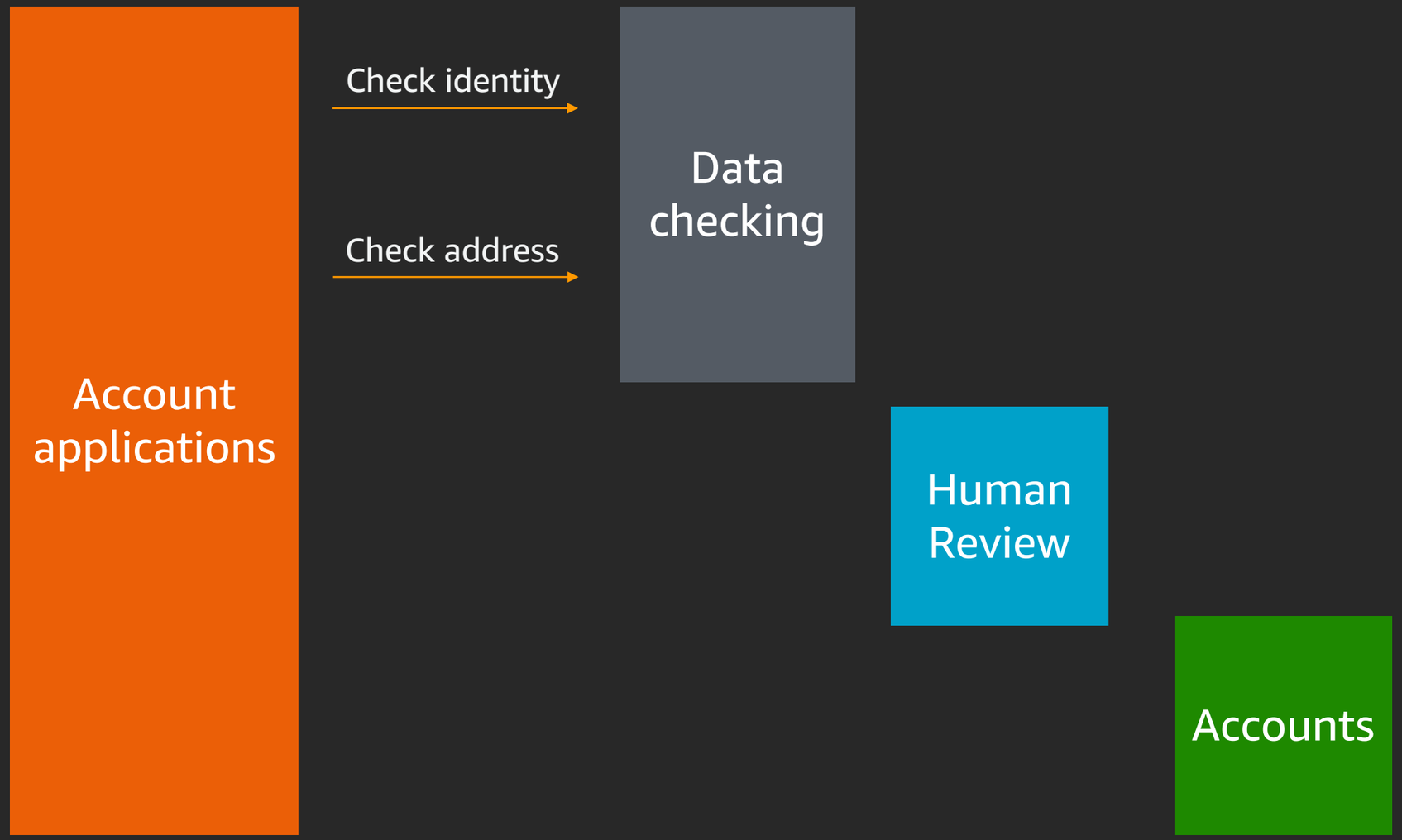


Orchestration: one process manages workflow state and calls appropriate services in turn

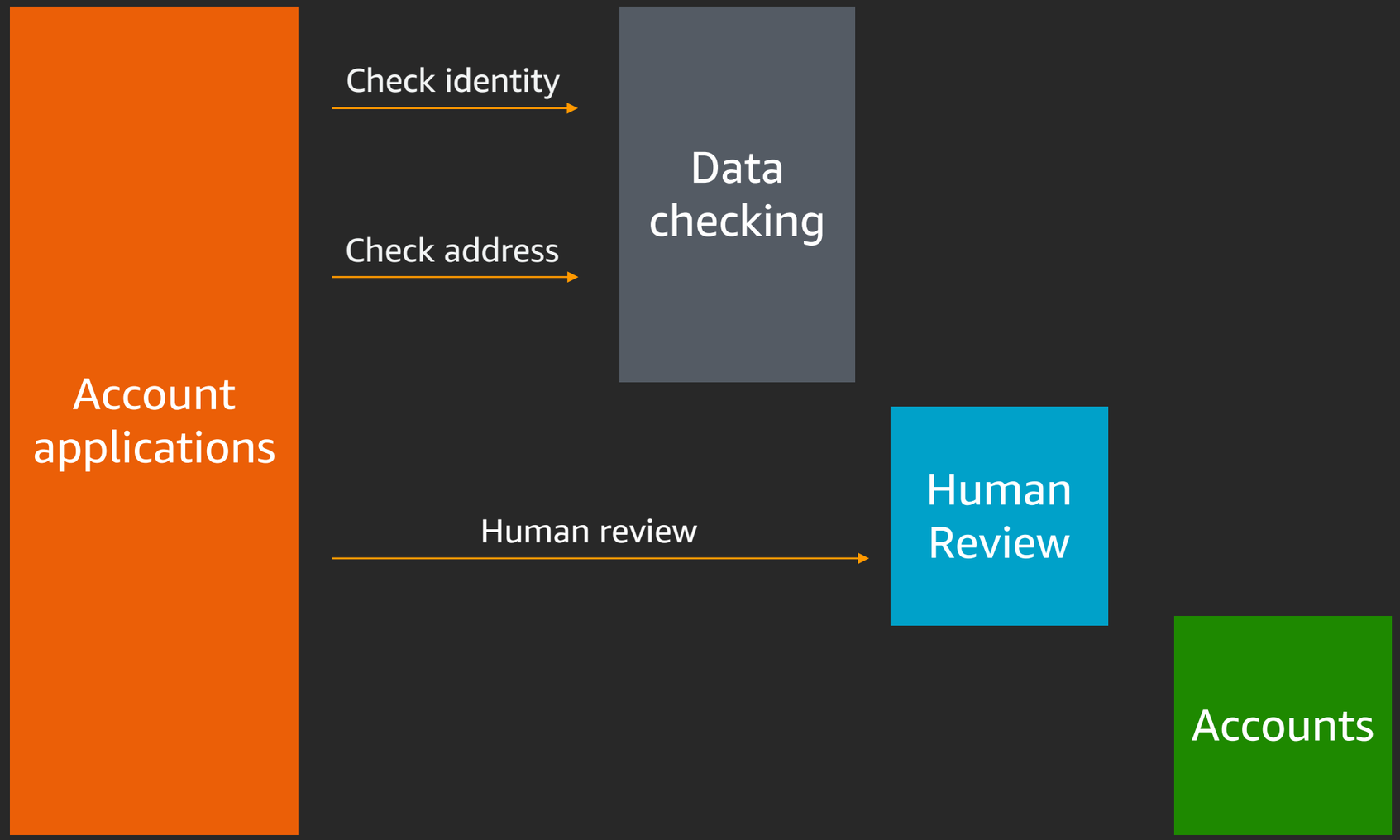
Orchestration: one process manages workflow state and calls appropriate services in turn



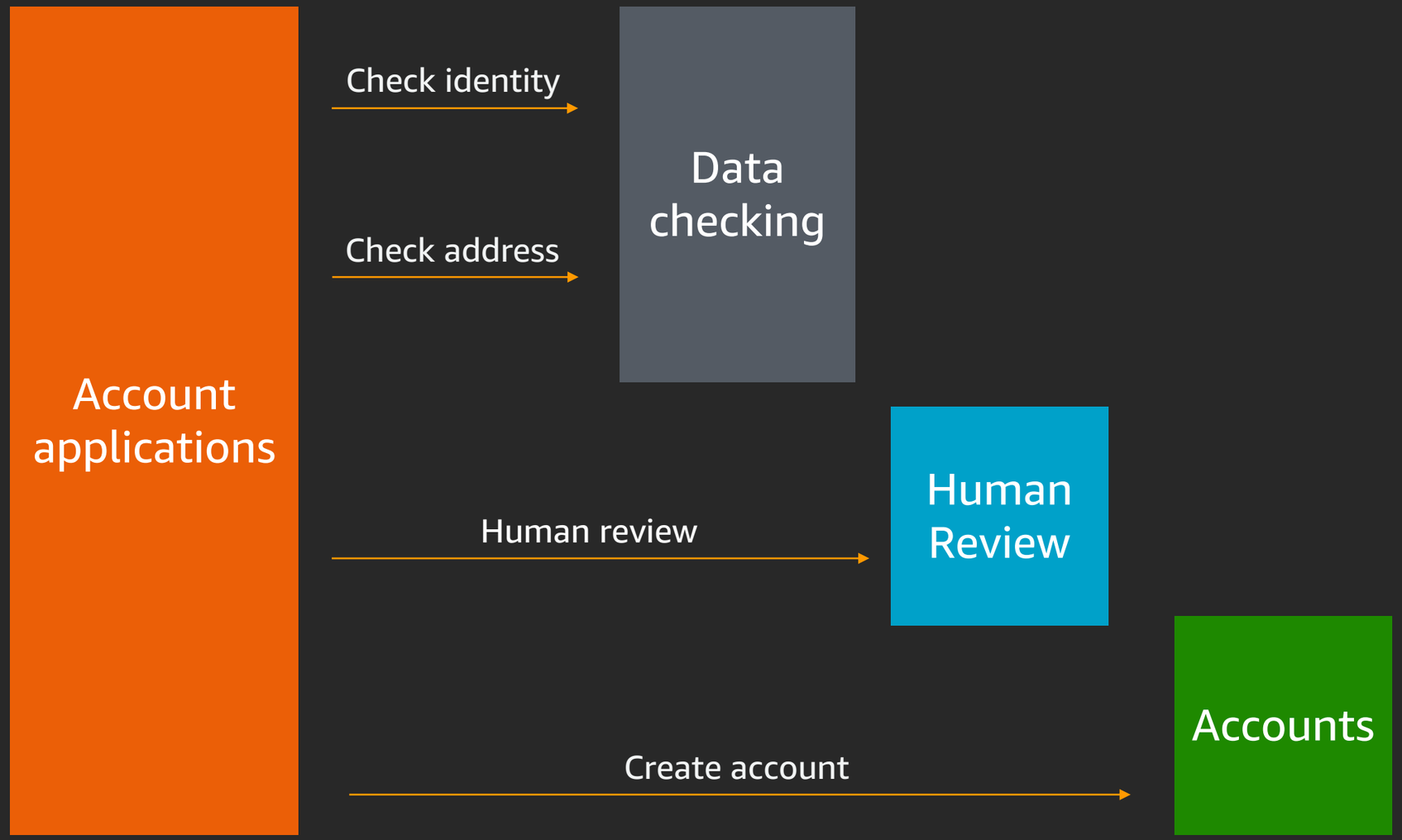
Orchestration: one process manages workflow state and calls appropriate services in turn



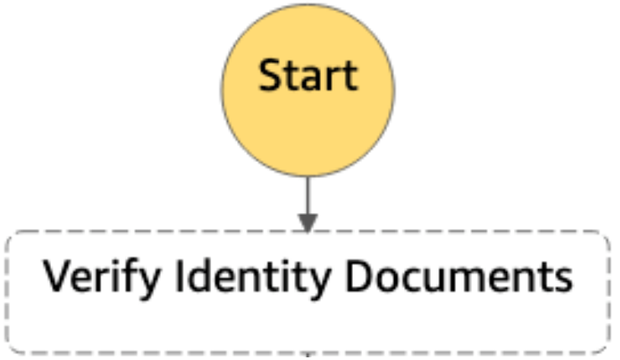
Orchestration: one process manages workflow state and calls appropriate services in turn

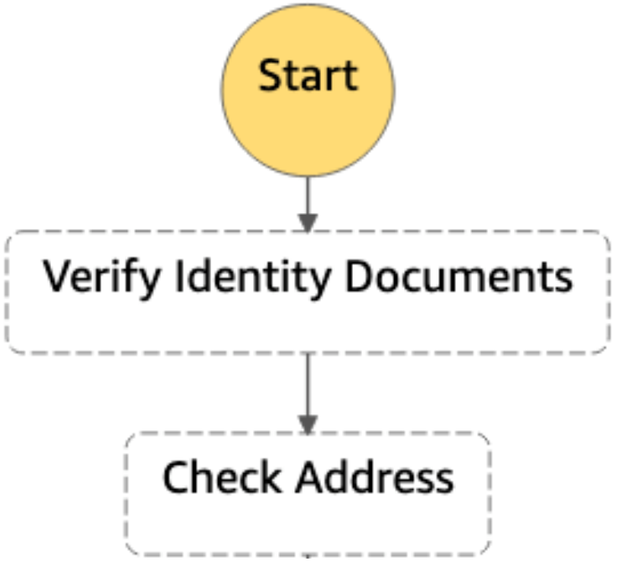


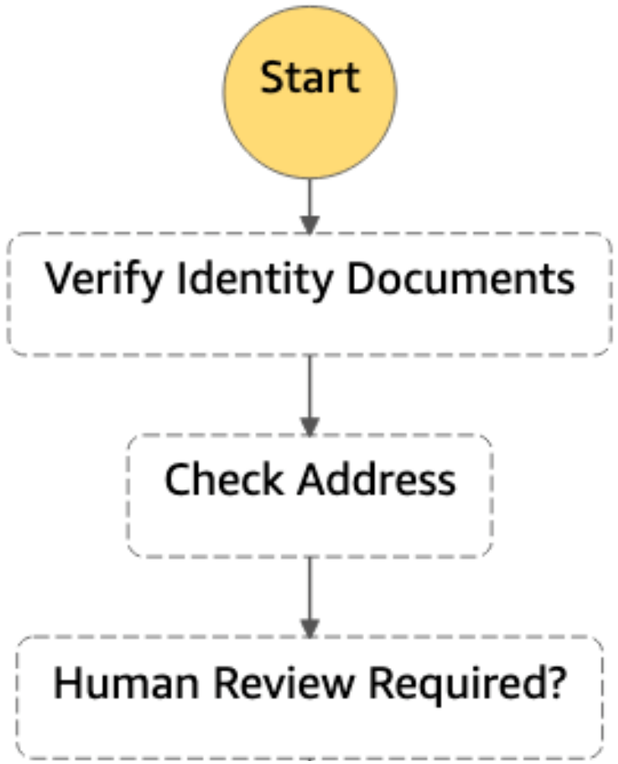
Orchestration: one process manages workflow state and calls appropriate services in turn

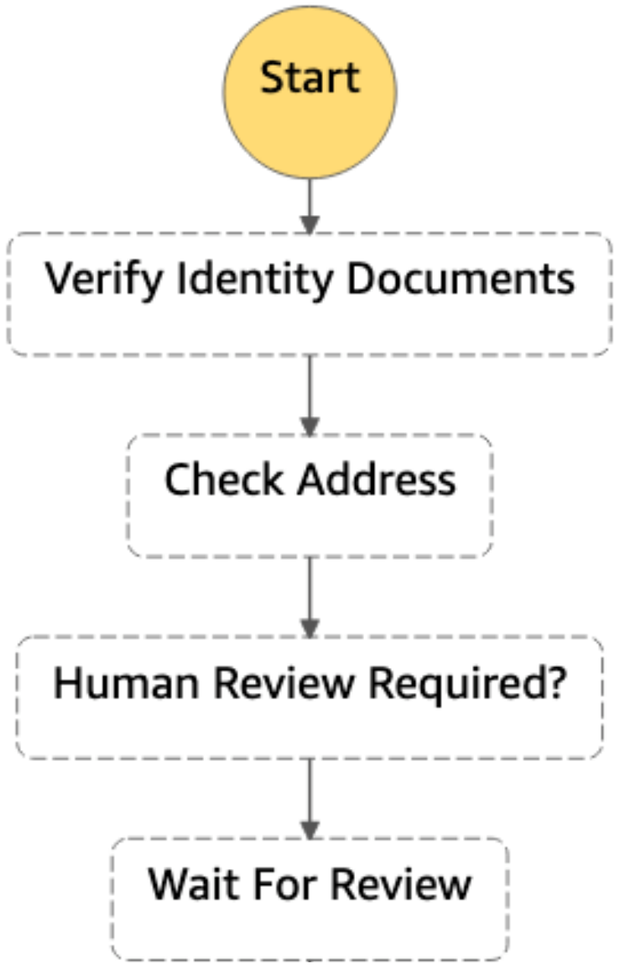


Example orchestration
Processing new bank account applications

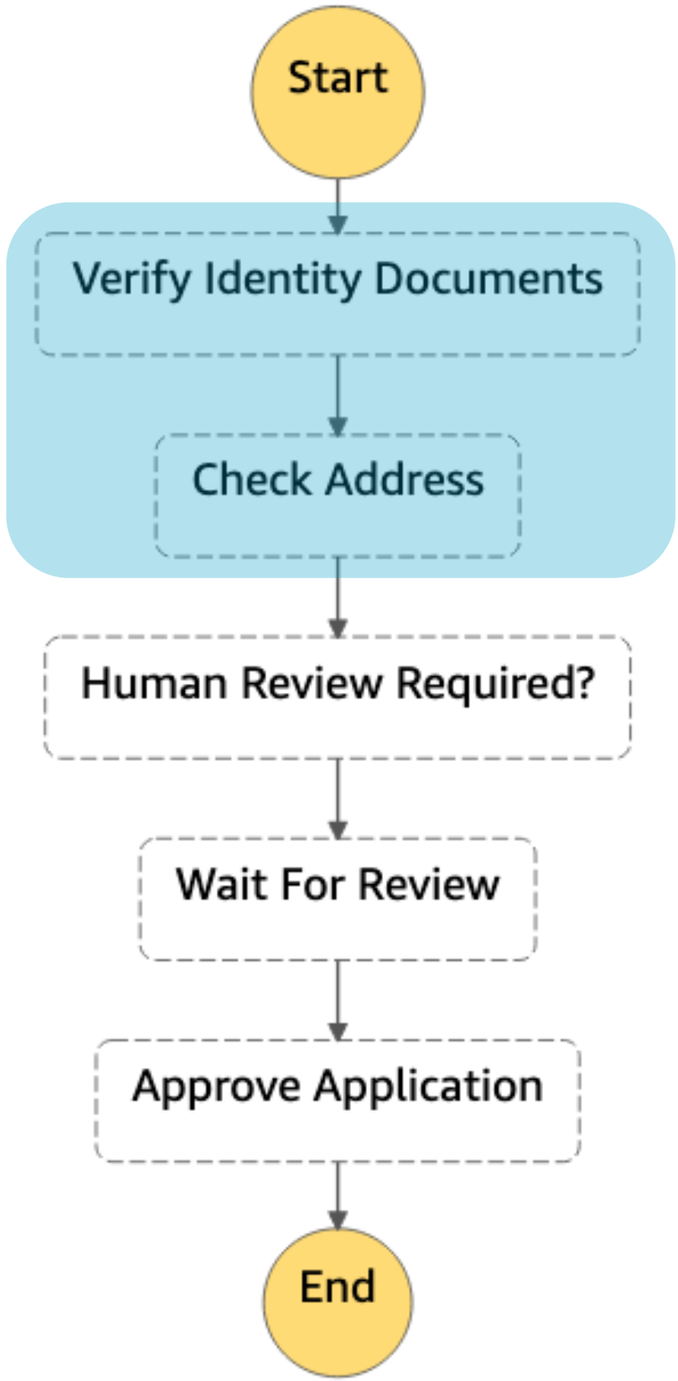

Start
Verify Identity Documents

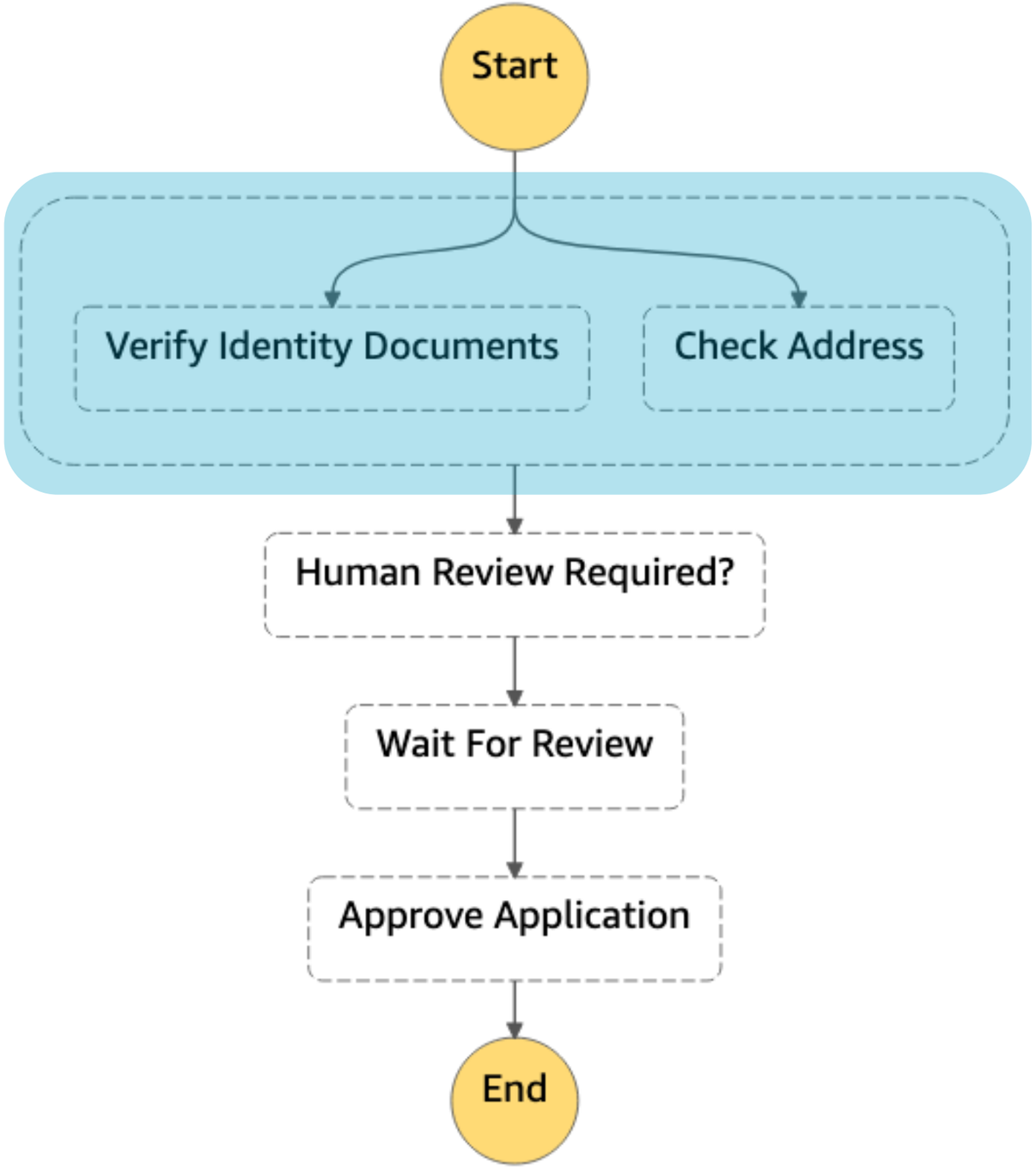

Start
Verify Identity Documents
Check Address

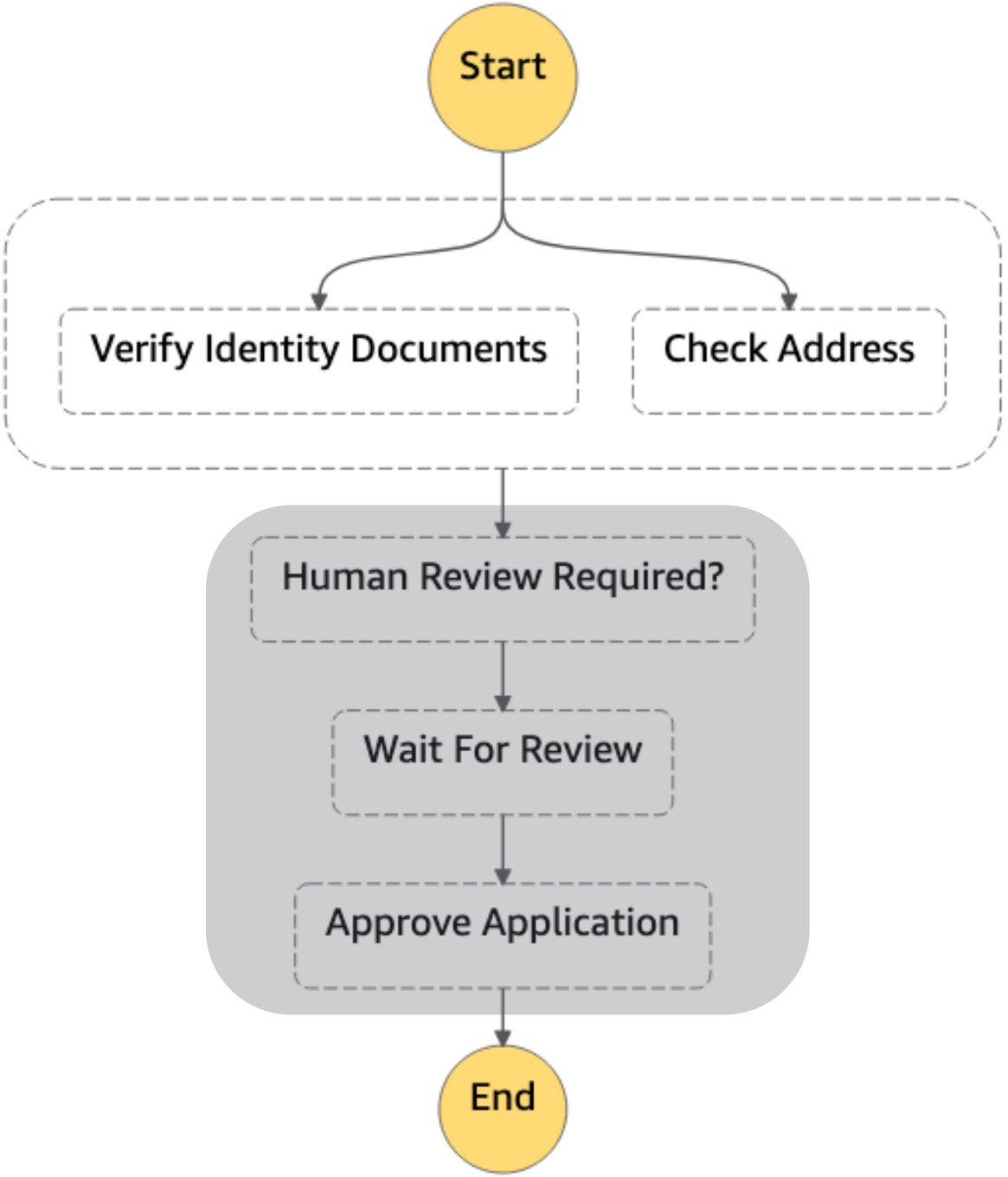

Start
Verify Identity Documents
Check Address
Human Review Required?

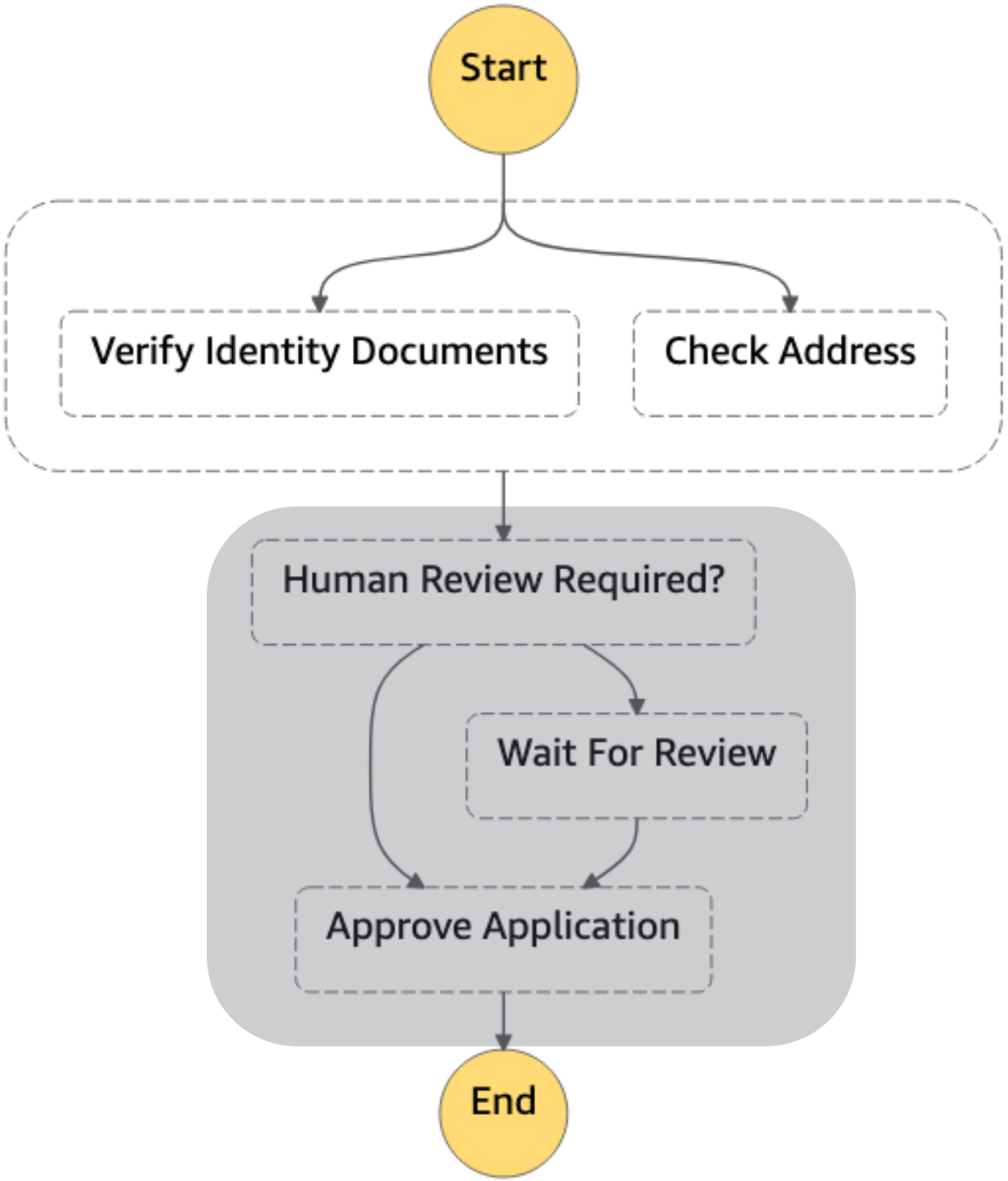

Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review

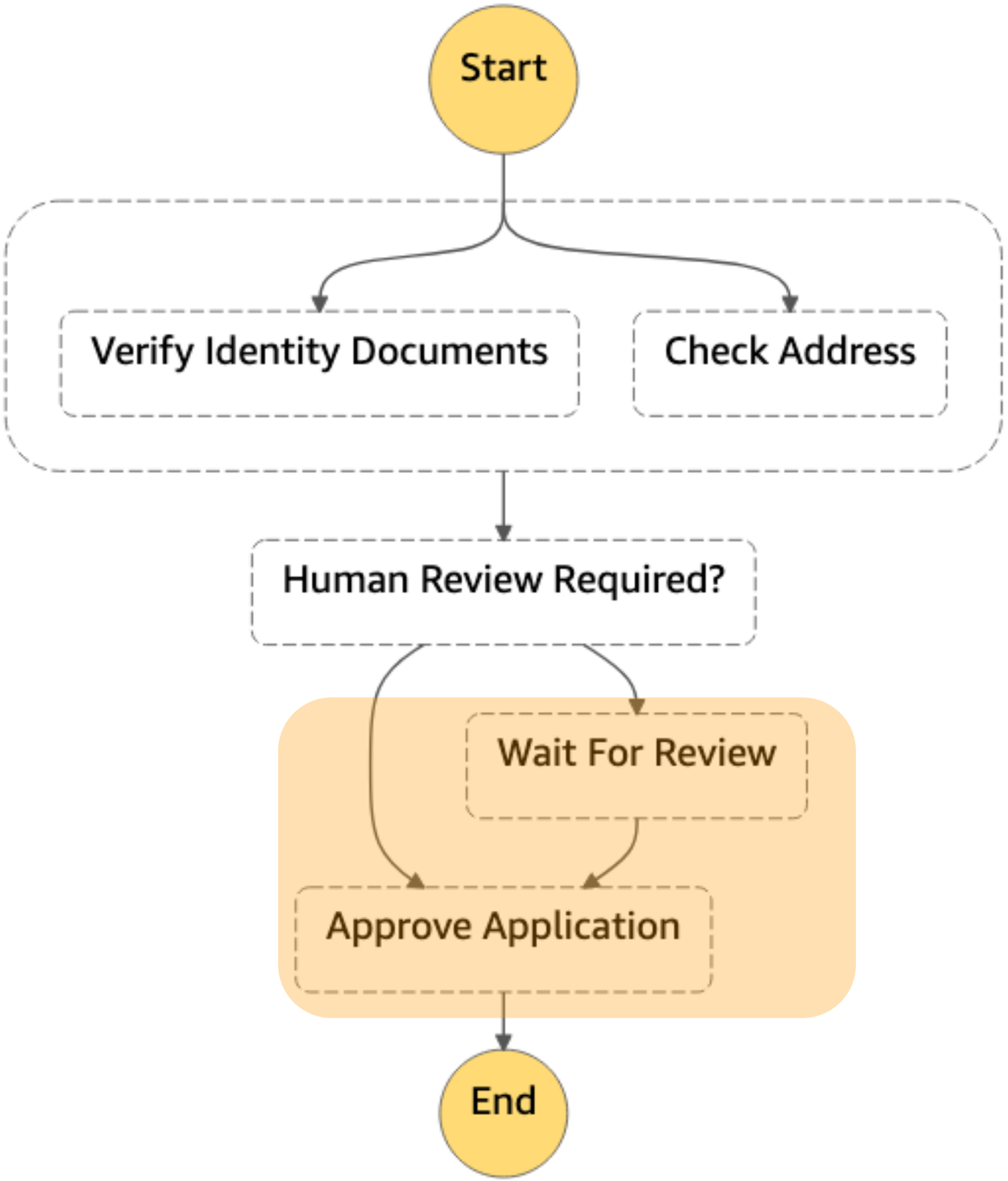

Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End

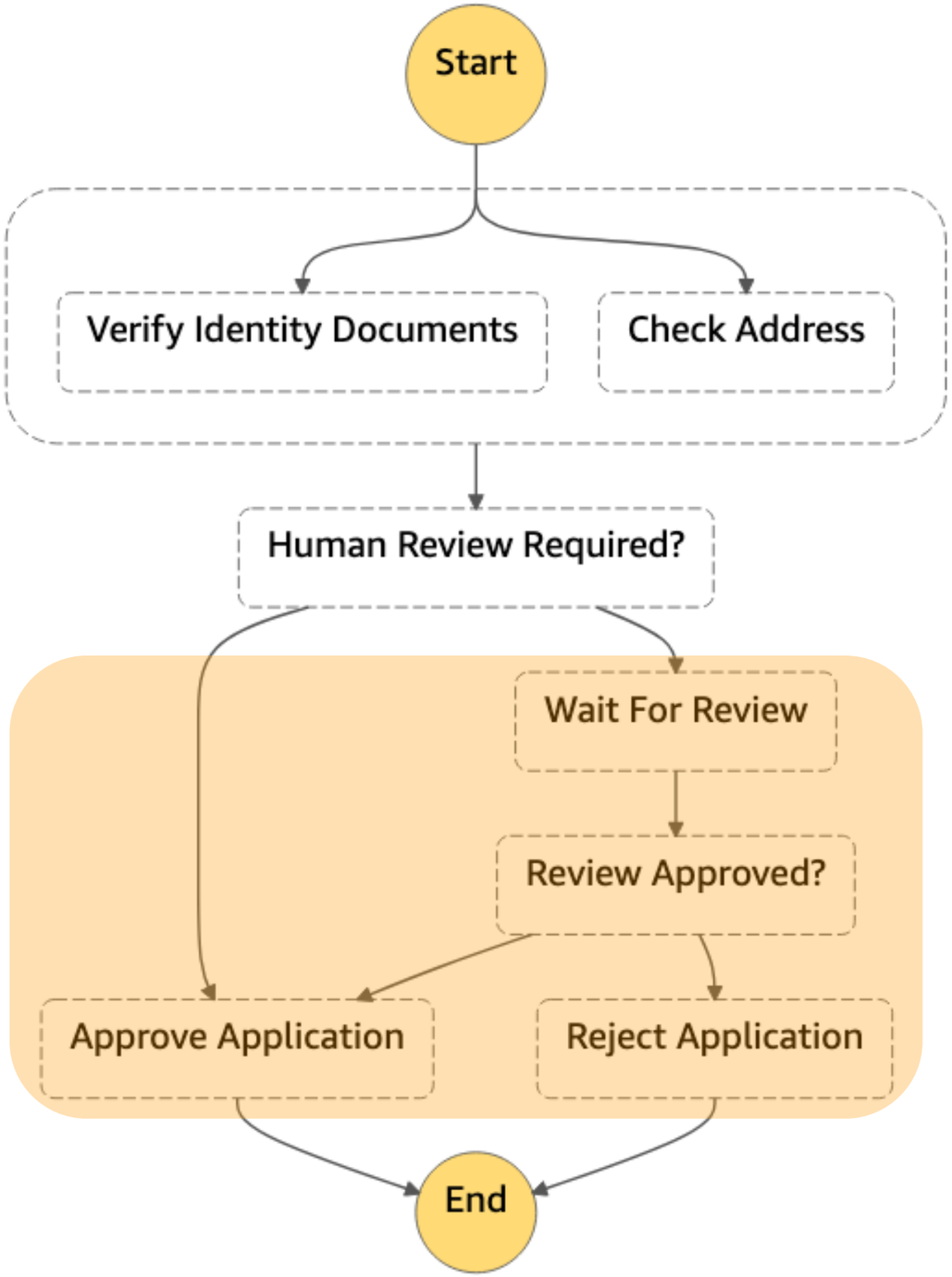

Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End

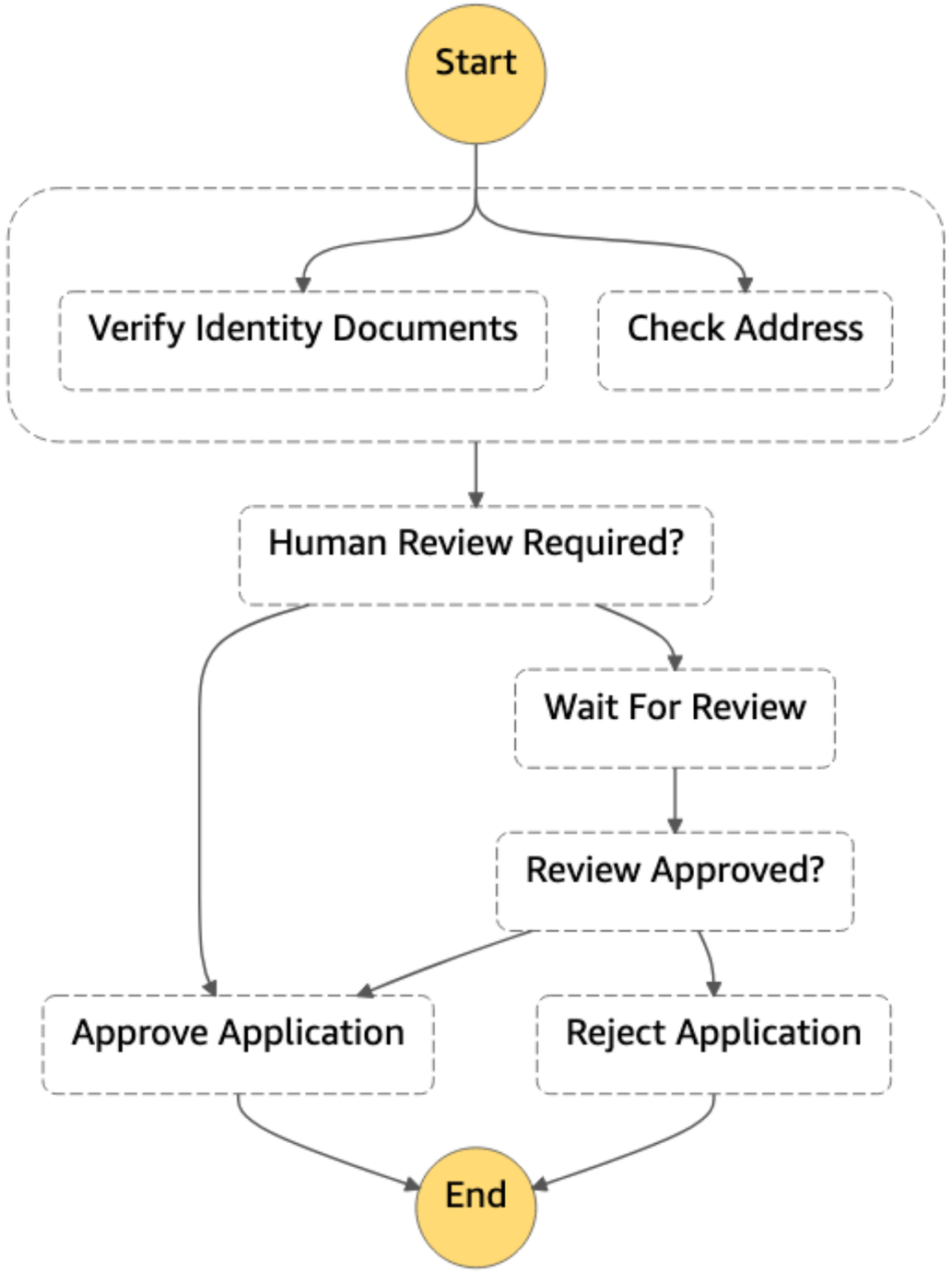

Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Approve Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Review Approved?
Approve Application
Reject Application
End


Start
Verify Identity Documents
Check Address
Human Review Required?
Wait For Review
Review Approved?
Approve Application
Reject Application
End

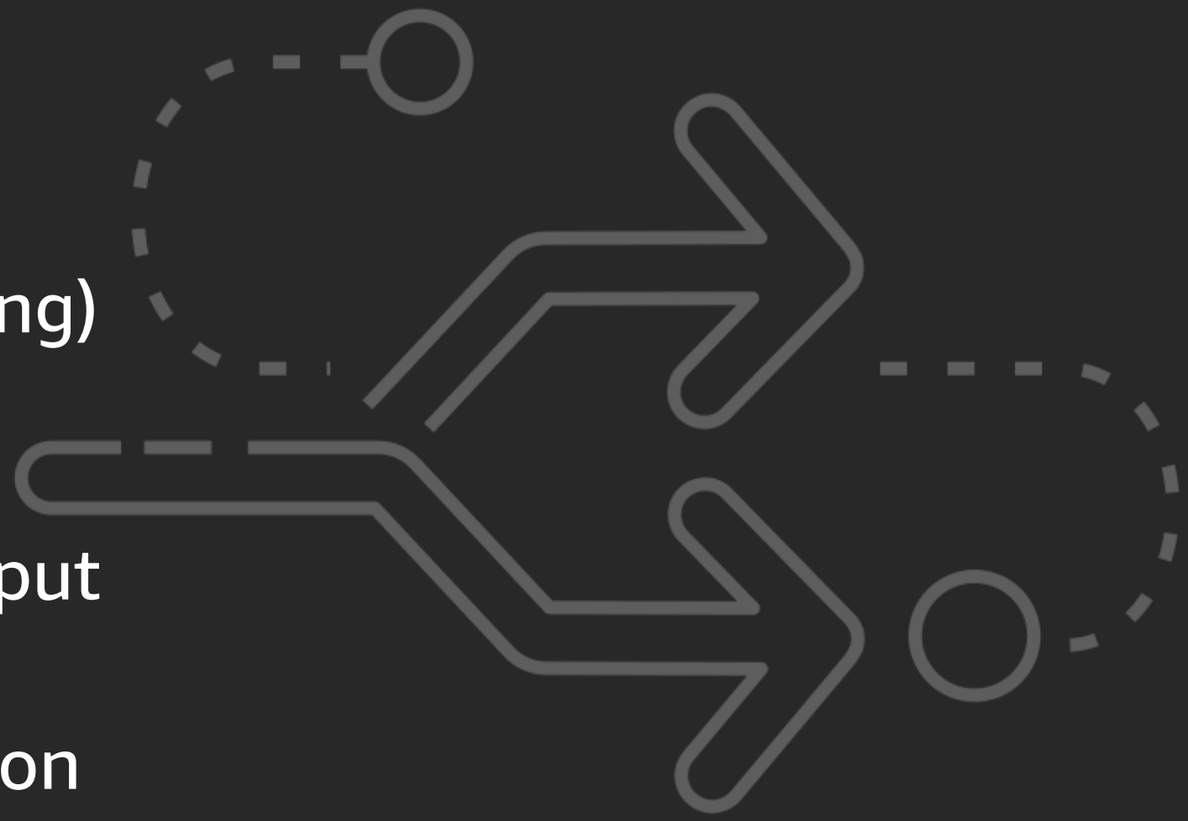
A state machine

Describes a collection of computational steps split into discrete states

Has one starting state and always one active state (while executing)

The active state receives input, takes some action, and generates output

Transitions between states are based on state outputs and rules that we define



AWS Step Functions

Fully-managed state machines on AWS

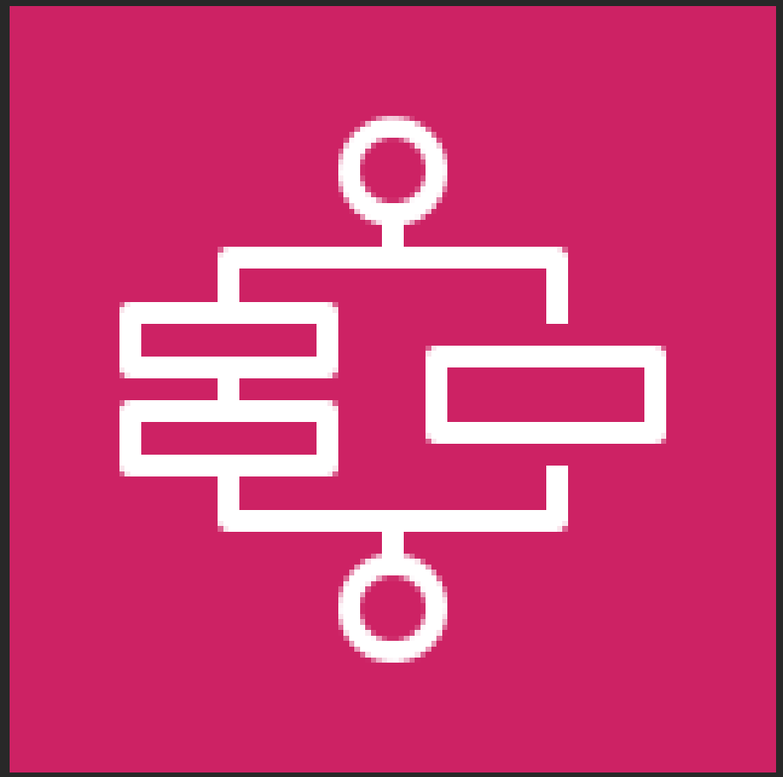
Resilient workflow automation

Built-in error handling

Powerful AWS service integration

First-class support for integrating with your own services

Auditable execution history and visual monitoring



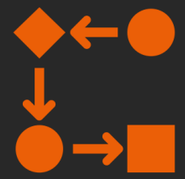
A Successful Application Flow

Manually Reviewing a Bad Address

AWS Step Functions
The basics

aws SUMMIT ONLINE

How AWS Step Functions work



The workflows you build with Step Functions are called state machines, and each step of your workflow is called a state.

When you execute your state machine, each move from one state to the next is called a state transition.

You can reuse components, easily edit the sequence of steps or swap out the code called by task states as your needs change.

Amazon States Language

https://states-language.net/spec.html

```
{
    "Comment": "A simple minimal example",
    "StartAt": "Hello World",
    "States": {
      "Hello World": {
        "Type": "Task",
        "Resource": "arn:aws:lambda...HelloWorld",
        "End": true
      },
      [. . .]
  }
}
```

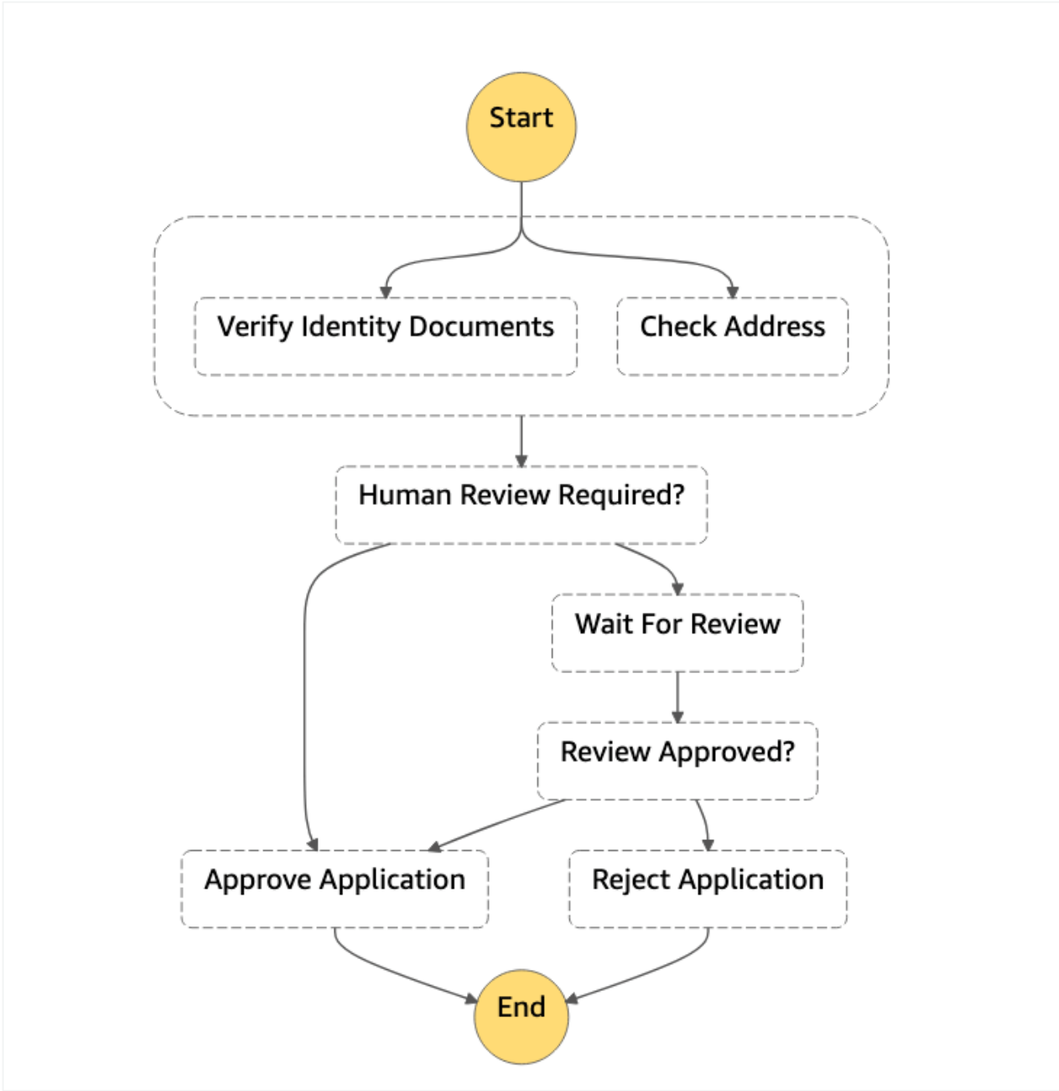
Example workflow: opening an account

Tasks

Parallel Steps

Branching Choice

Wait for a callback

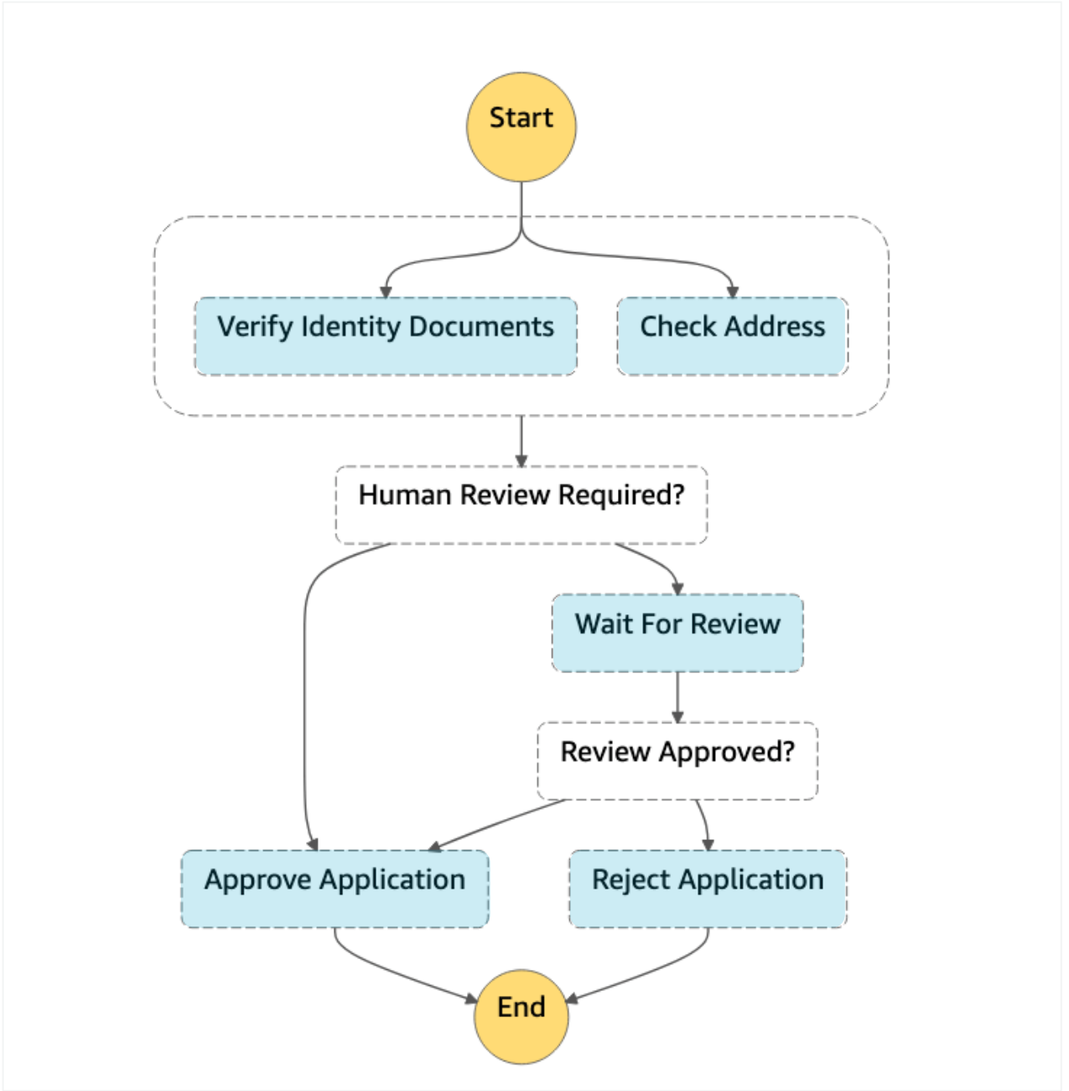


Performing a task

Call an AWS Lambda Function

Wait for a polling worker to perform an activity

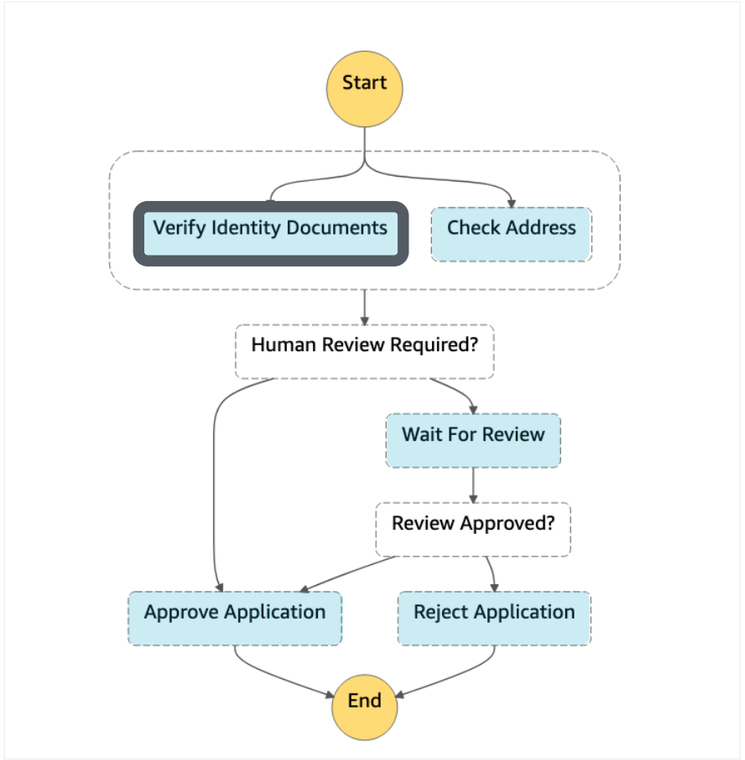
Pass parameters to an API of an integrated AWS Service



Performing a task

Example: Execute a AWS Lambda Function

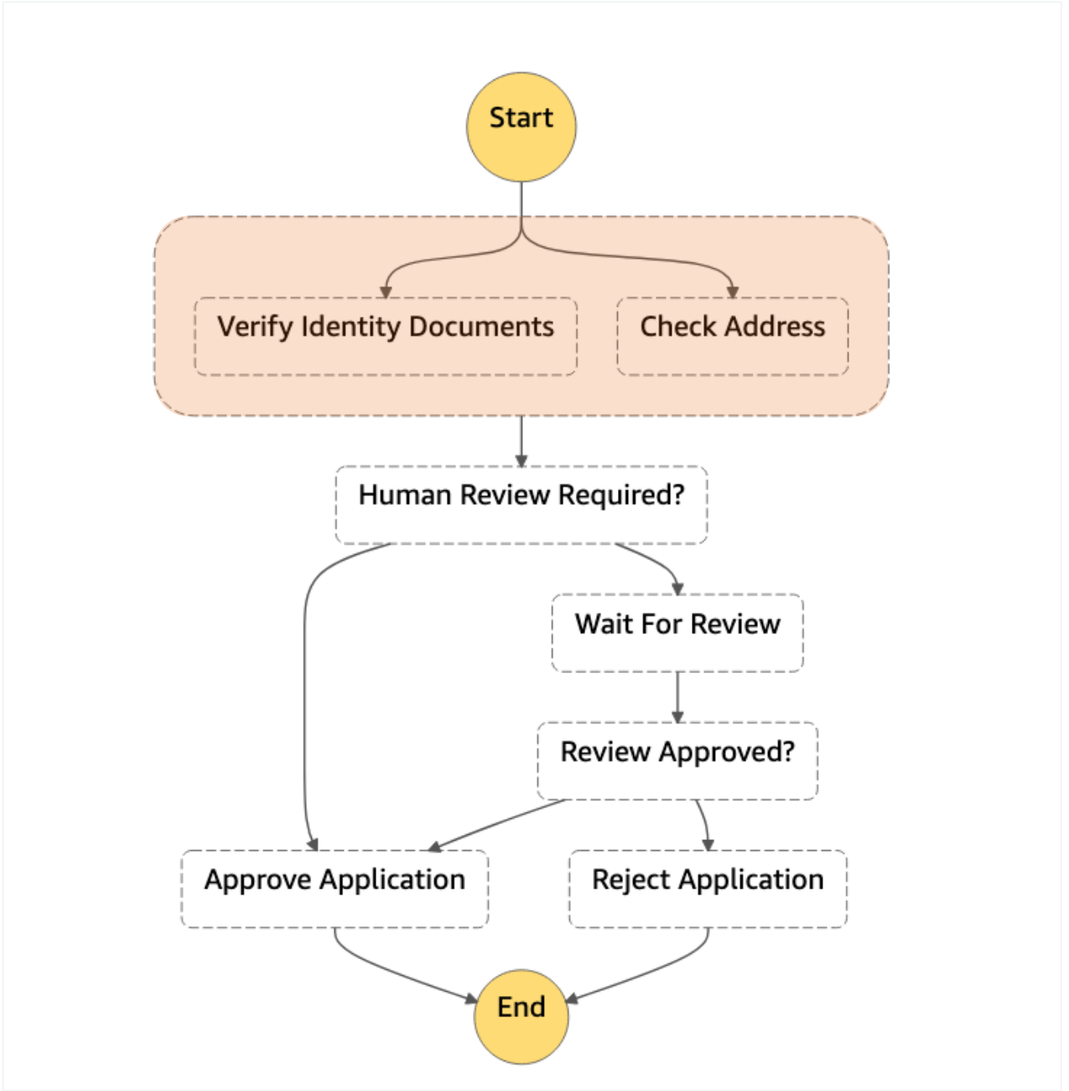
```
"Verify Identity Documents": {
  "Type": "Task",
    "Parameters": {
      "name.$": "$.application.name"
      "identityDoc.$": "$.application.idDocS3path"
    },
    "Resource": "arn:aws:lambda...VerifyIdDocs",
    "End": true
}
```



Executing branches in parallel

Contains an array of state machines branches to execute in parallel

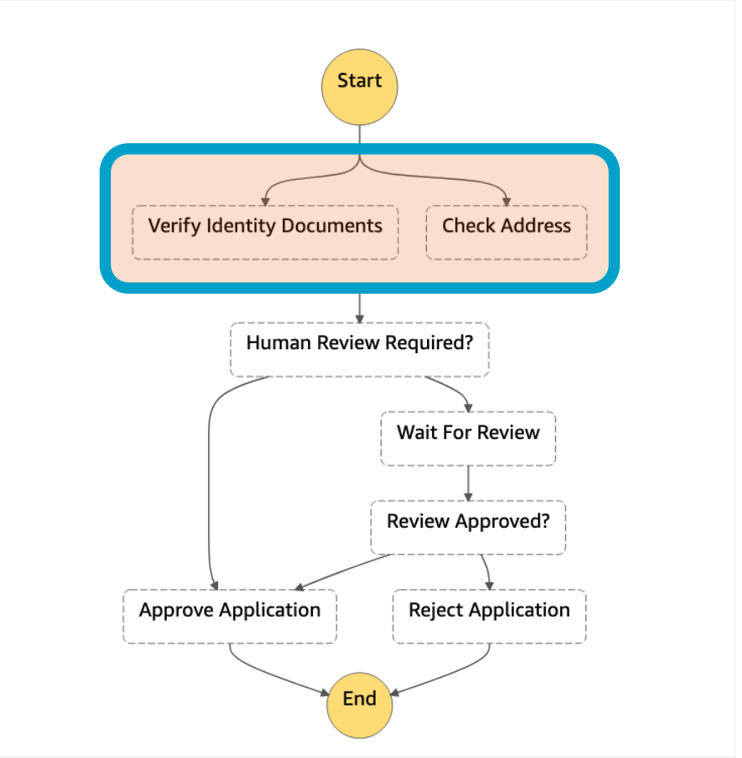
Outputs an array of outputs from each state machine in its branches



Executing branches in parallel

Example: Run two branches in parallel

```
"Perform Automated Checks": {
  "Type": "Parallel",
    "Branches": [
      {
        "StartAt": "Verify Identity Documents",
        "States": { "Verify Identity Documents": { … } }
      },
      {
        "StartAt": "Check Address",
        "States": { "Check Address": { … } }
      }
    ]
  },
  "ResultPath": "$.checks",
  "Next": "Human Review Required?"
}
```

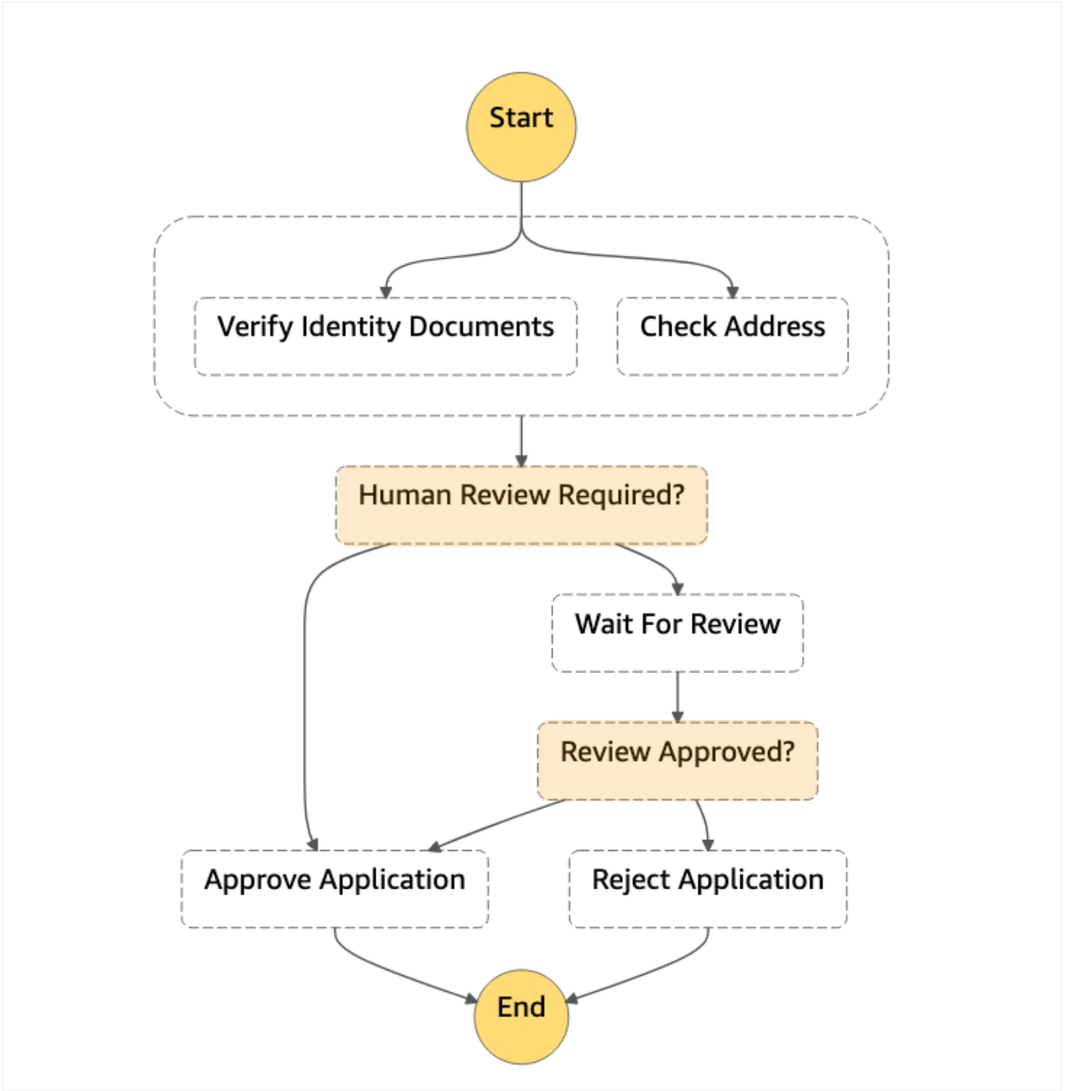


Making a choice

Like a switch statement in programming

Inspects an array of *choice* expressions, comparing variables to values

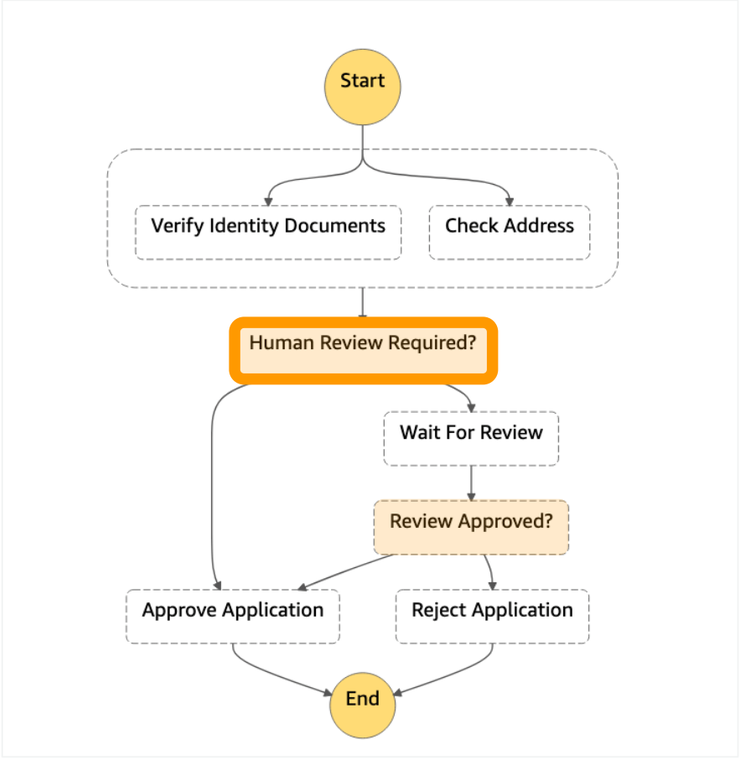
Determines which state to transition to next



Making a choice

Example: Choose next step based on state outputs

```
"Human Review Required?": {
  "Type": "Choice",
  "Choices": [
    {
      "Variable": "$.checks[0].flagged",
      "BooleanEquals": true,
      "Next": "Wait For Review"
    },
    {
      "Variable": "$.checks[1].flagged",
      "BooleanEquals": true,
      "Next": "Wait For Review"
    }
  ],
  "Default": "Approve Application"
}
```

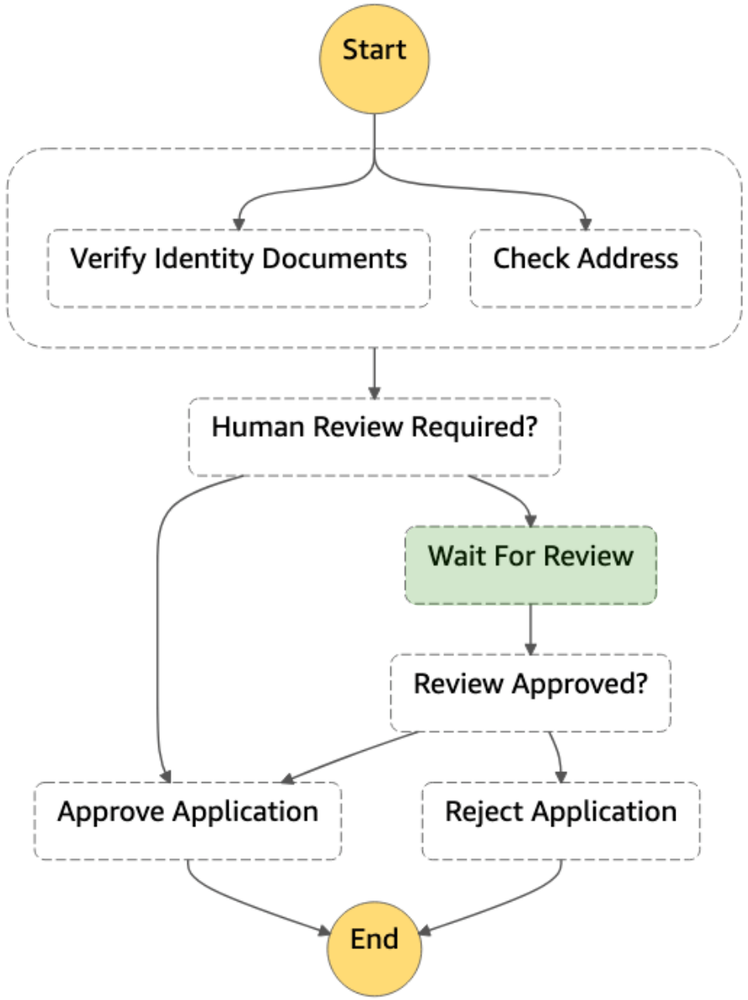


Waiting for a callback

Generates a Task Token and passes it to an integrated service

When the recipient process is complete, it calls SendTaskSuccess or SendTaskFailure with the Task Token

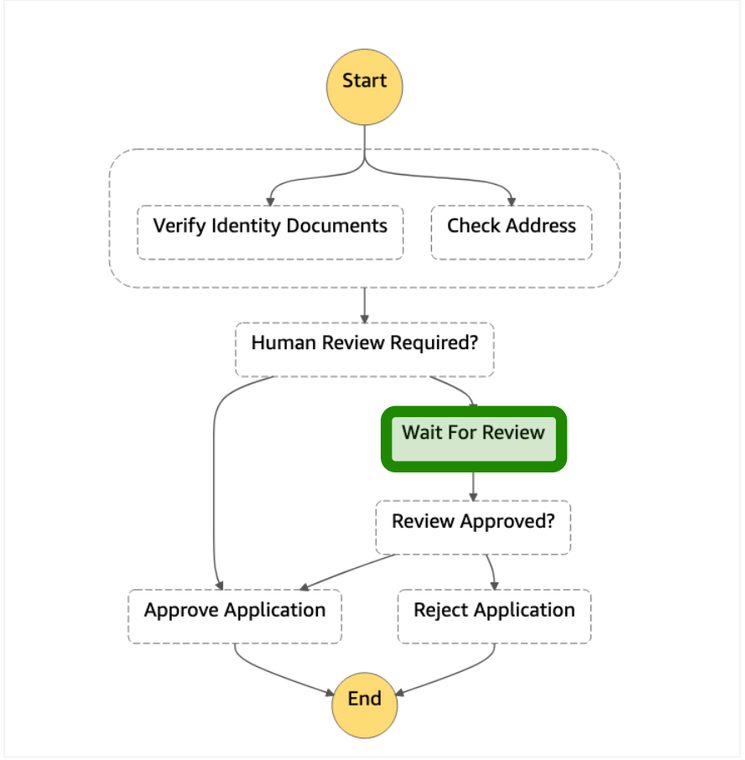
Workflow then resumes its execution



Waiting for a callback

Example: Pause and wait for an external callback

```
"Type": "Task",
"Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
"Parameters": {
    "FunctionName": "FlagApplicationForReview",
    "Payload": {
        "applicationId.$": "$.application.id",
        "taskToken.$": "$$.Task.Token"
    }
},
"ResultPath": "$.reviewDecision",
"Next": "ReviewApproved?"
```



Error handling

Failures can happen due to Timeouts, Failed Tasks, or Insufficient Permissions

Tasks can Retry when errors occur using a BackoffRate up to MaxAttempts

Tasks can Catch specific errors and transition to other states

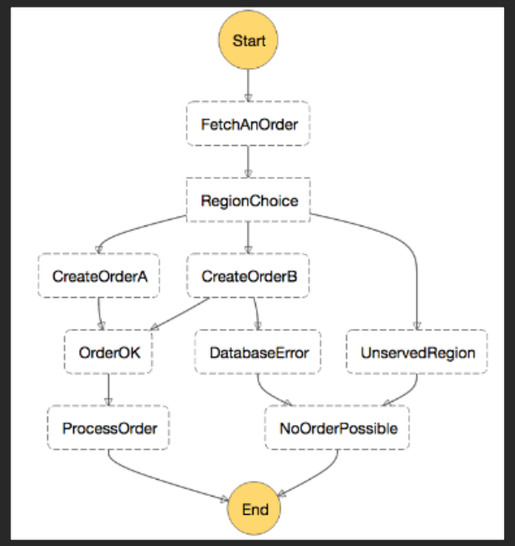
Handling Errors

Working with AWS Step Functions

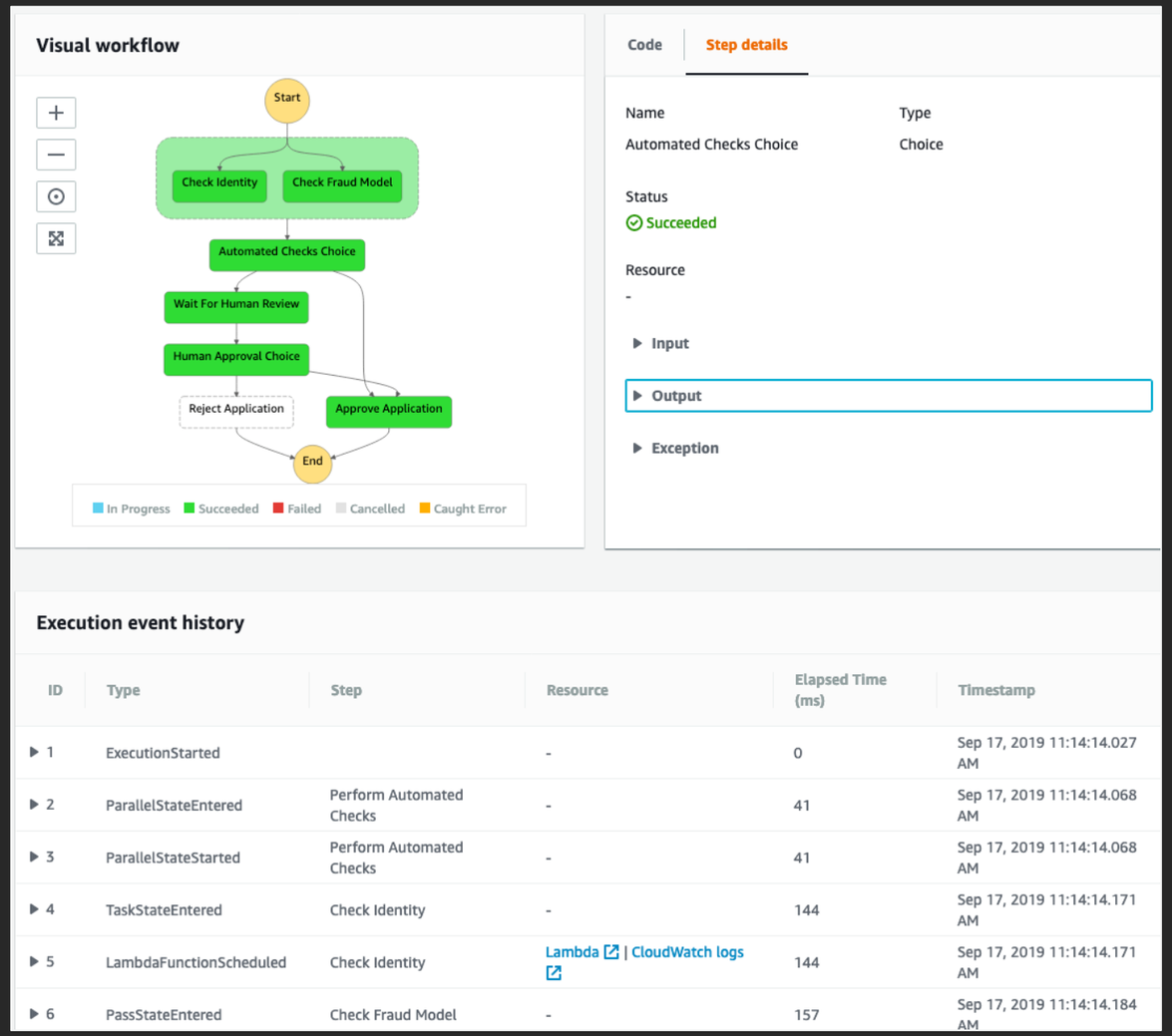
Define in JSON

```
{
    "Comment": "Manage opening an account",
    "StartAt": "Perform Automated Checks",
    "States": {
        "Perform Automated Checks": {
            "Type": "Parallel",
            "Branches": [{
                    "StartAt": "Check Identity",
                    "States": {
                        "Check Identity": {
                            "Type": "Task",
                            "Parameters": {
```

Visualise in the Console



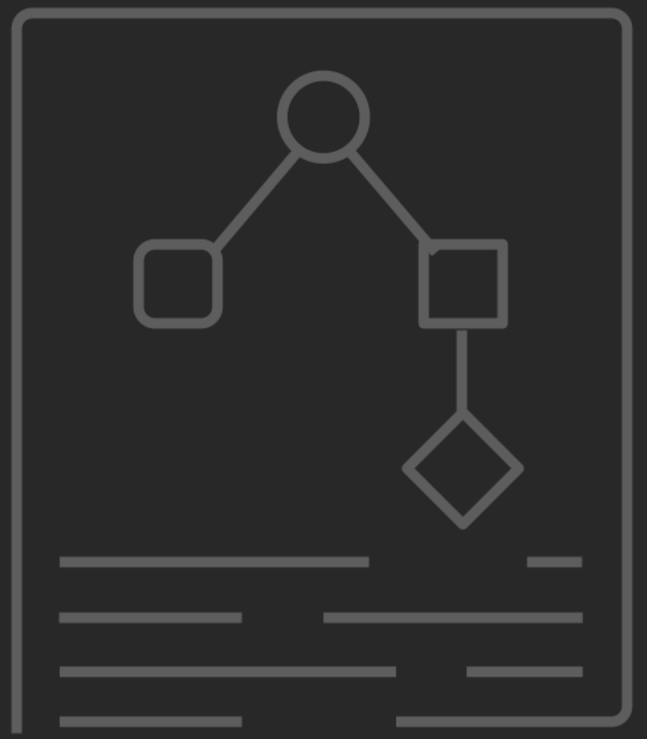
Monitor Executions



AWS Step Functions
Diving deeper

aws SUMMIT ONLINE

State types



Task	Execute work
Choice	Add branching logic
Wait	Add a timed delay
Parallel	Execute branches in parallel
Map	Process each of an input array's items with a state machine
Succeed	Signal a successful execution and stop
Fail	Signal a failed execution and stop
Pass	Pass input to output

AWS Step Functions service integrations

AWS
Lambda

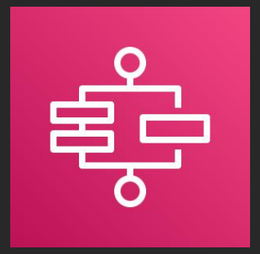
Amazon
Elastic Container Service

AWS
Batch

Amazon
DynamoDB

AWS
Glue

Amazon
SageMaker



AWS
Step Functions

Amazon
Simple Notification Service

Amazon
Simple Queue Service

Customer examples

On-demand, audited host access pipeline

nib

When an operator needs to ‘break glass’ into an environment, they authenticate and request permission.

An AWS Step Function notifies the required actor and waits for an approval response that will kick off provisioning a securely configured bastion host.

It also starts a timer that will ensure the bastion environment is cleaned up in a timely manner.

https://www.youtube.com/watch?v=ICEhR3mshyg

Shortened processing time for updating nutrition labels from 36 hours down to 10 seconds

The Coca-Cola Company

Data validation and transformation steps are designed visually with non-technical personnel

Validation and transformation steps verified in real-time as data flows through the state machine

Process optimisations are identified and implemented on the spot

https://www.youtube.com/watch?v=sMaqd5J69Ns

NEW

AWS Step Functions Express Workflows

aws SUMMIT ONLINE

AWS Step Functions Express Workflows NEW

Orchestrate AWS compute, database, and messaging services at rates up to 100,000 events per second, suitable for high-volume event processing workloads such as IoT data ingestion, microservices orchestration, and streaming data processing and transformation

Faster: greater than 100K state transitions per second

Designed for short-duration workflows: < 5 mins.

Cost effective at scale

Standard vs. express workflows

	Standard	Express
Maximum duration	365 days	5 minutes
Execution start rate	Over 2,000 per second	Over 100,000 per second
State transition rate	Over 4,000 per second per account	Nearly unlimited
Execution semantics	Exactly-once workflow execution	At-least-once workflow execution

Standard vs. express workflows (continued)

	Standard	Express
Executions	Executions are persisted and have ARNs	Executions are not persisted except as log data
Execution history	Stored in Step Functions, with tooling for visual debugging in the console	Sent to Amazon CloudWatch Logs
Service integrations	Supports all service integrations and activities	Supports all service integrations. Does not support activities.
Patterns	Supports all patterns	Does not support Job-run (.sync) or Callback (.wait For Callback)

Get building

Development tips

AWS Step Functions Local
https://docs.aws.amazon.com/step-functions/latest/dg/sfn-local.html

Statelint
https://github.com/awslabs/statelint

Serverless Framework Plug-in
https://github.com/horike37/serverless-step-functions

Visual Studio Code
aws-step-functions-constructor extension
https://marketplace.visualstudio.com/items?itemName=paulshestakov.aws-step-functions-constructor

Get started building with AWS Step Functions

Workshop: Intro to Service Coordination ~2 hours

https://step-functions-workshop.go-aws.com/

Developer Guide ~2 hours

https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

Reference Architectures

https://aws.amazon.com/step-functions/resources/

AWS Step Functions key benefits

Fully-managed service

High availability & automatic scaling

Visual monitoring & state management

Auditable execution history

Built-in error handling

Pay per use

Step up. Go build!

aws SUMMIT ONLINE

Thank you!

Gabe Hollombe

 @gabehollombe

aws SUMMIT ONLINE