# Using Step Functions to Control Flow

**Dror Helper**

@dhelper www.helpercode.com

## Module Overview

**What are step functions?**

- Why and when to use Step functions

**Working with state machines**

- Creating and running workflows
  - Invocation options
- Building state machines
  - Different state types
  - Calling Lambda functions
  - Handling errors

## Step Functions

**Introduced in 2016**

**Coordinate distributed tasks**

- State machine framework
- Visualized flows
- Trigger and track each step

**Written using ASL**

## Why Use Step Functions?

Orchestration

Long Running Tasks

Built-in Error Handling

No Resource Provisioning Required

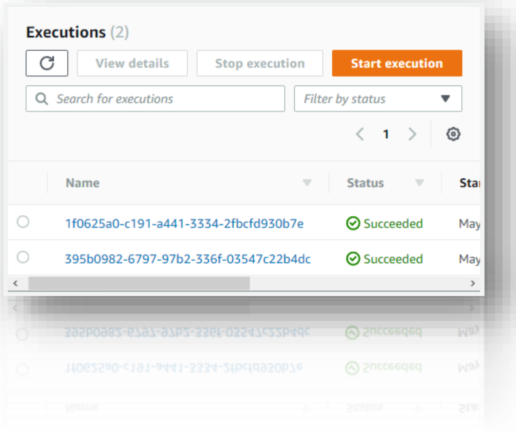
# Amazon States Language (ASL)

```
{
  "Comment": "Very informative and enlightening comment",
  "StartAt": "State1",
  “TimeoutSeconds": "2520",
  “Version": "2.0",
  "States": {
    ...
  }
}
```

# Defining States

```
"states": {
     "state1": {
          "Comment": “This is a simple state",
          "Type": "Pass",
          "Result": "I am first",
          "Next": "State2"
     },
     "state2": {
          ...
          "End": true
     }
}
```

# Triggering Step Functions



| Name | Status | Sta |
|---|---|---|
| 1f0625a0-c191-a441-3334-2fbcfd930b7e | Succeeded | May |
| 395b0982-6797-97b2-336f-03547c22b4dc | Succeeded | May |

**Manually**

Using the AWS Console

**API Gateway**

Call from APIs

**CloudWatch**

Step Functions as the event's target

**Lambda**

Start execution from code

## State Types

**Pass**

Passes input to output

**Task**

Single unit of work

**Wait**

Delays execution

**Choice**

Adds branching logic

**Parallel**

Run parallel states

**Fail/Succeed**

Stops execution

## Task State

```
"TaskState": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloWorld",
  "TimeoutSeconds": 300,
  "HeartbeatSeconds": 60,
  "Next": "NextState"
}
```

## Input and Output Processing

**InputPath**

Select a portion of the state input to pass for processing

**ResultPath**

Takes the results of executing the state's task and places them in the input

**OutputPath**

Selects a portion of the input to send as the state's output

## Input and Output processing

```
 "Add Two Numbers": {

      "Type": "Task",

      "Resource": "arn:aws:lambda:us-east1:1234567:function:Add",

      "Next": "Do Something",

      "InputPath": "$.numbers",

      "ResultPath": "$.sum",

      "OutputPath": "$.sum"

}
```

```
{
    "numbers": [2, 3],
    ...
}   "sum": 55
}
```

# Wait State

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

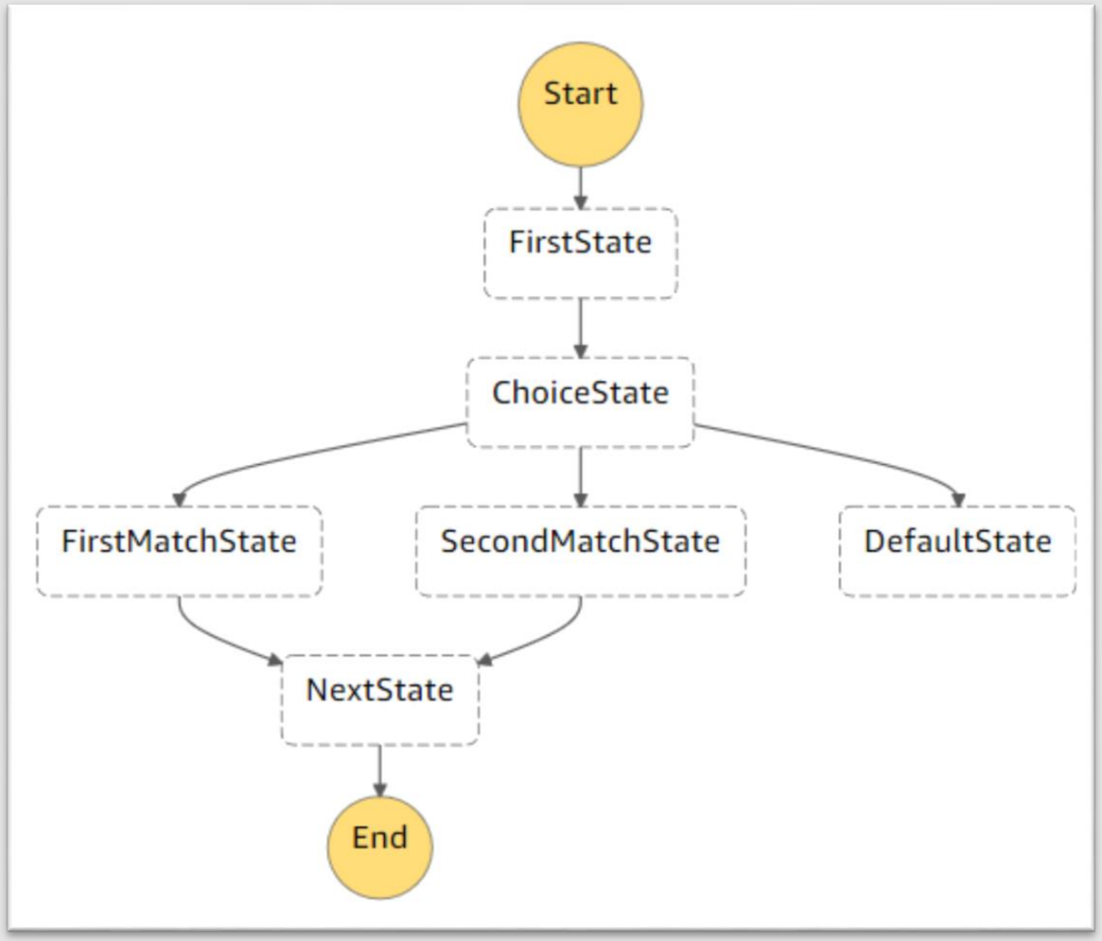
```
"wait_seconds": {
  "Type": "Wait",
  "SecondsPath": "$.waitseconds",
  "Next": "NextState"
}
```

```
"wait_Y2K": {
  "Type": "Wait",
  "Timestamp": "2000-01-01T00:00:00Z",
  "Next": "NextState"
}
```

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

# Choice State

```
"ChoiceState": {
      "Type" : "Choice",
      "Choices": [
       {
          "Variable": "$.foo",
          "NumericEquals": 1,
          "Next": "FirstMatchState"
        },
        ...
      ],
      "Default": "DefaultState"
    },
```



## Supported Comparison Operators

**Supported Types**

- Boolean
- Numeric
- String
- Timestamp

**Comparison Operators**

- <type>Equals
- <type>GreaterThan
- <type>GreaterThanEquals
- <type>LessThan
- <type>LessThanEquals

**Not, And, Or**

- Nested *Choice* without Next fields

## Choice State Examples

```
{
  "Variable": "$.name",
  "StringEquals": "Inigo Montoya",
  "Next": "PrepareToDie"
}
```

```
{
    "Not": {
        "Variable": "$.error",
        "StringEquals": "ok"
    },
    "Next": "DontPanic"
}
```

```
"And": [
        {
          "Variable": "$.age",
          "NumericGreaterThanEquals": 40
        },
        {
          "Variable": "$.age",
          "NumericLessThan": 50
        }
      ],
      "Next": "TheNewThirties"
```

## Retry on Failure

```
"MySimpleTask": {
     "Type": "Task",
     "Resource": "arn:aws:lambda:us-east1:1234567:function:risky_business",
     "Retry": [
       {
         "ErrorEquals": ["CustomError", "OtherCustomerError"],
         "IntervalSeconds": 1,
         "MaxAttempts": 5,
         "BackoffRate": 2.0
       },
```

# Catch Failure

```
"MySimpleTask": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east1:1234567:function:risky_business",
      "Catch": [
        {
          "ErrorEquals": ["CustomError", "OtherCustomerError"],
          "Next": "CustomErrorFallback"
        },
"MyState": {
      "Type": "Parallel",
      "Next": "Final State",
      "Branches": [
         {
          "StartAt": "Wait 20s",
          "States": {
            "Wait 20s": {
              ...
            }
          }
        },
        ...
```

## Parallel Task

## Fail and Succeed States

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

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

## Summary

**Creating state machines**

**State Types**

- Pass
- Task
- Wait
- Choice
- Parallel
- Fail
- Succeed

**Error Handling**

- Retry
- Catch