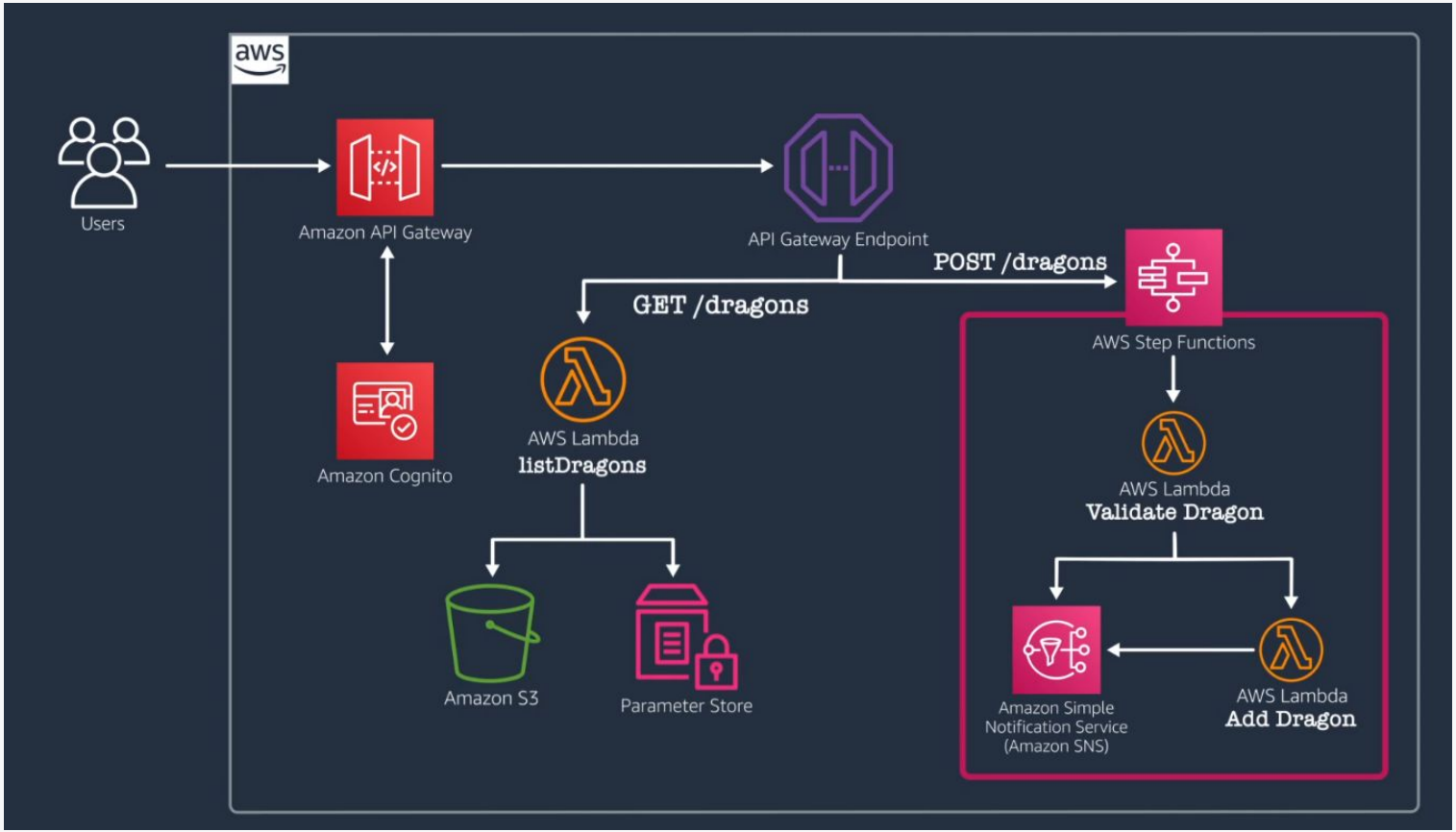
04 - AWS Step Function

recap

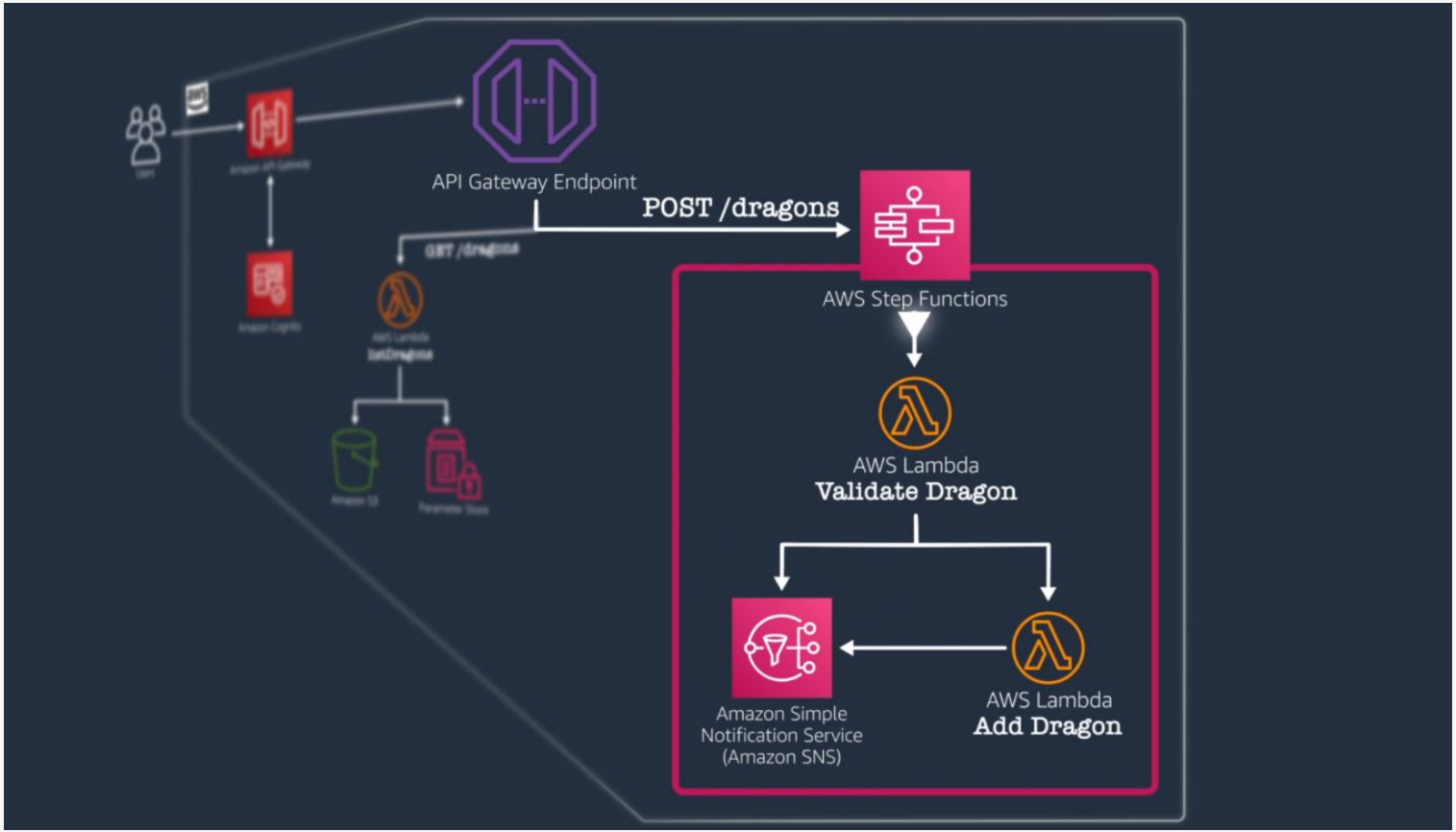
We have set up API Gateway with the POST method that will accept the data about a newly cited dragon.

Currently, it is backed by a mock endpoint.


aws
Users
Amazon API Gateway
API Gateway Endpoint
POST /dragons
GET /dragons
AWS Step Functions
Amazon Cognito
AWS Lambda
listDragons
AWS Lambda
Validate Dragon
Amazon S3
Parameter Store
Amazon Simple
Notification Service
(Amazon SNS)
AWS Lambda
Add Dragon

We have already created the Lambda functions
for validate dragon and add dragon.

but we haven't implemented anything that
makes these Lambda functions execute in the correct order.



We need to determine what order to run each task, whether to run tasks in parallel, or to run one task over another based on some sort of criteria.

We could have one Lambda call the next one depending on decisions made in the code.

But the problem with that is that those Lambda functions would be tightly coupled.

In general, it's not a great idea to inject your workflow logic into your code when working with something like Lambda.

In order to change the way your process works, you'll maybe need multiple code changes across multiple Lambda functions.

what tool can we use to create this workflow?

AWS Step Functions

AWS Step Functions is a serverless workflow tool.

The workflow in Step Functions is called a state machine.

Step Functions is based on the concept of state machines, states, and tasks.

A State Machine is a computational model used to represent the behavior of a system.

It defines a set of states and transitions between those states.

Real-World Example: Imagine a Traffic Light as a State Machine

Used in various domains like:

- Software Engineering (Workflow Management)
- Automation and Control Systems
- Game Development (Game States)
- Finite State Machines in Theory of Computation

State machines are built using
Amazon States Language to represent the required elements.

The Amazon States Language
is a JSON-based, structured language used
to define the state machine, the workers, the workflow and so on.

Step Functions allows you to create a state machine
that consists of one or many states.

It keeps track of the state of the state machine,
as well as keeping a history of executions of that state machine.

Step Functions logs the state of each step
so that you have the ability to diagnose and debug any problems encountered.

A state is essentially one step in your workflow.

A step might be executing a Lambda function,
running code inside of a container,
reaching out to another AWS service,
or many other options.

In our example, we will be creating different states for validating a dragon, sending an error text message, adding a dragon, and sending a success text message.

AWS Step Functions is serverless.

We don’t need to manage a server to host the state machine.

The code running on Lambda also is not aware of the fact that it is part of a state machine.

Step Function State Types

In AWS Step Functions,
a "state" refers to a single step or task in your state machine workflow.

Each state defines what happens when it's executed
and what state should follow it.

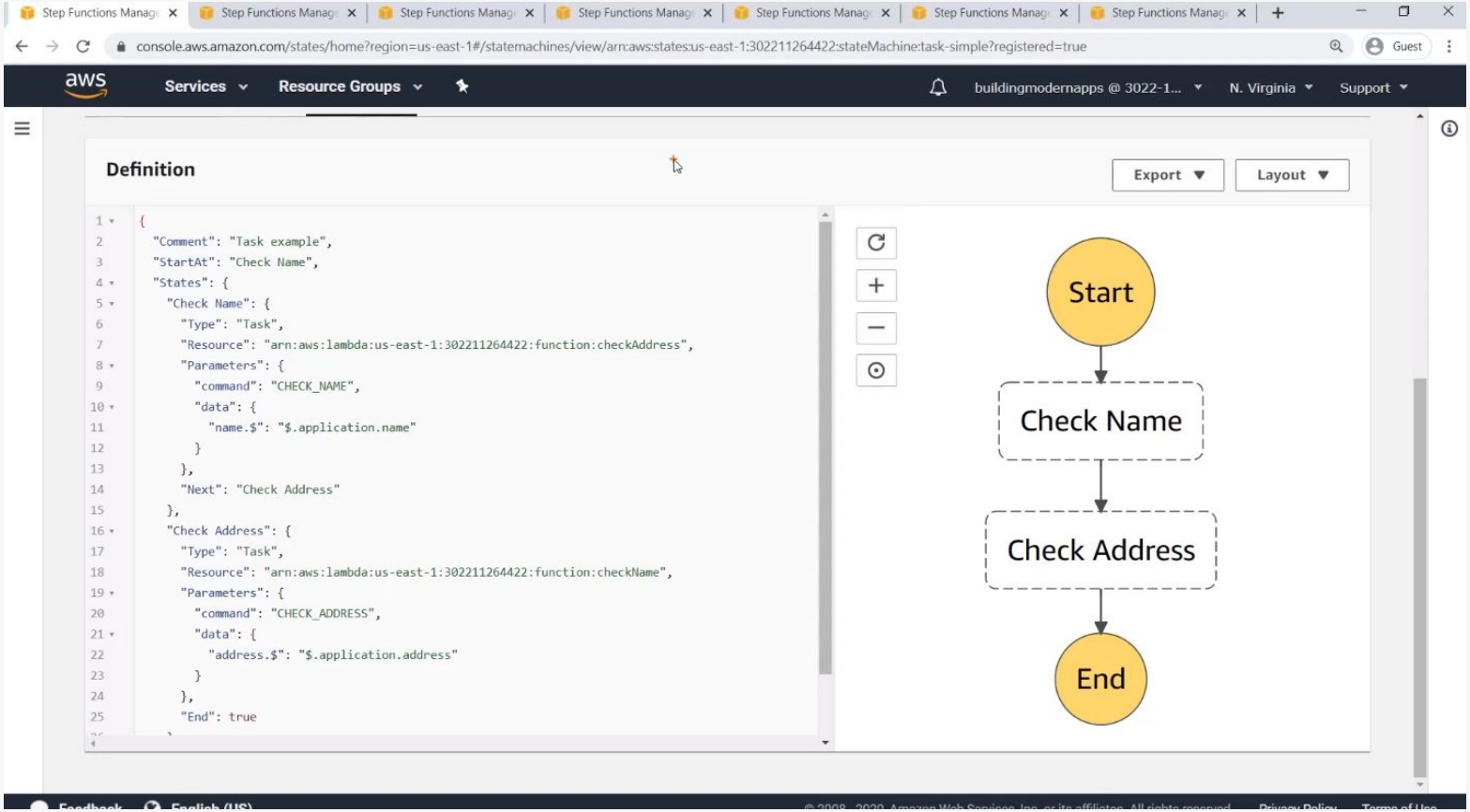
There are several state types, each designed for different purposes.

Task State: This is where you specify exactly what to do when the state is executed.

It can represent a single unit of work,
like invoking a Lambda function, making an API call, or running a container.

One of these integration is to invoke a lambda function,
wait for it's return and then use that to continue into the workflow.

Let's take an example here.



validate name and address

In AWS Step Functions, are written in Amazon States Language using JSON.

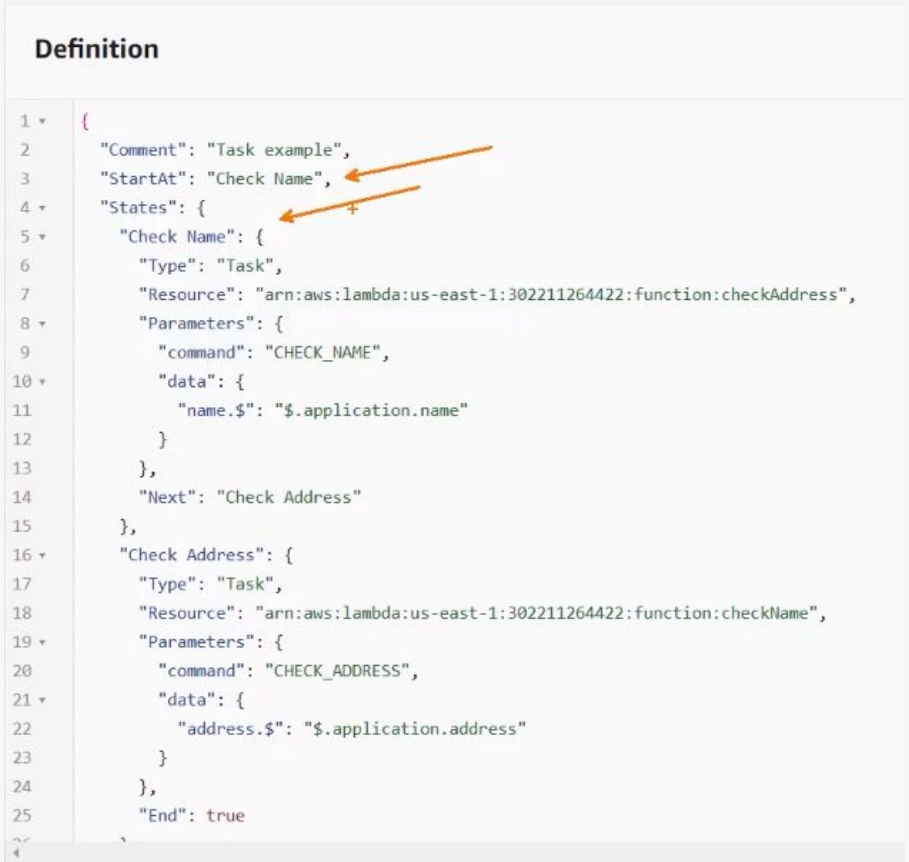
```
{
  "Comment": "A simple state machine example",
  "StartAt": "HelloWorld",
  "States": {
    "HelloWorld": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "End": true
    }
  }
}
```

Definition

```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

Let’s look at the code to validate name and address

Definition



```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

a "StartAt" field to specify the initial state

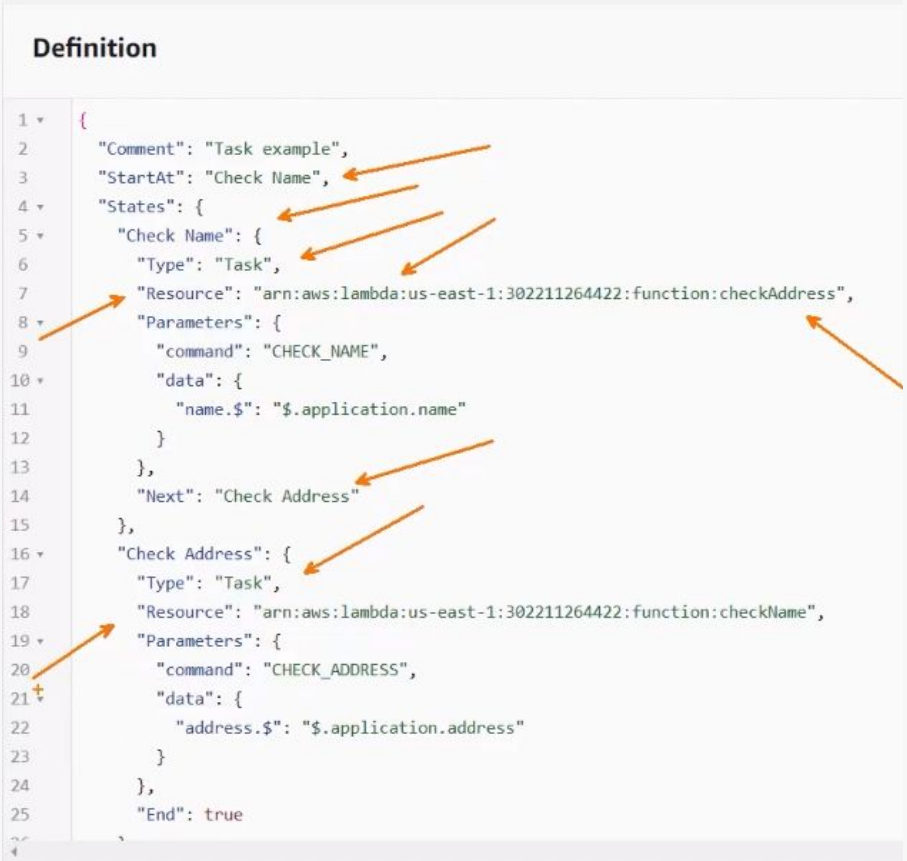
"States" field where you define individual states.

Definition

```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

Each state is defined as a JSON object with various properties like "Type", "Resource", and others depending on the type of state.

Definition



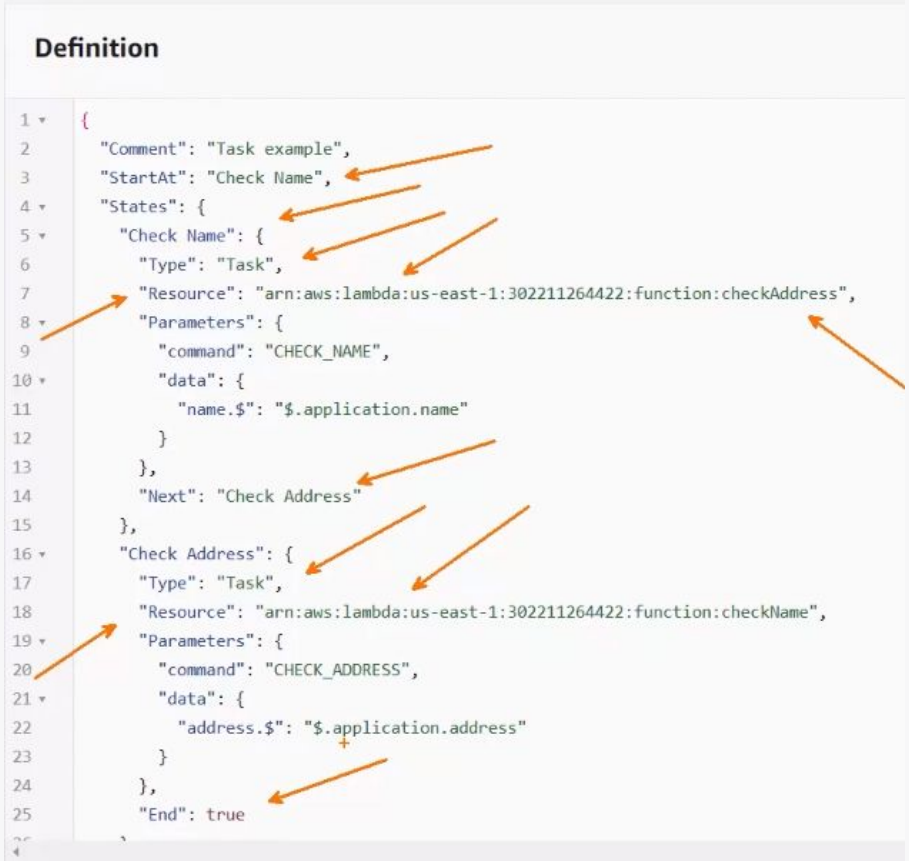
```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

Next specifies the state that should be executed next after the current state completes successfully.

Definition

```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

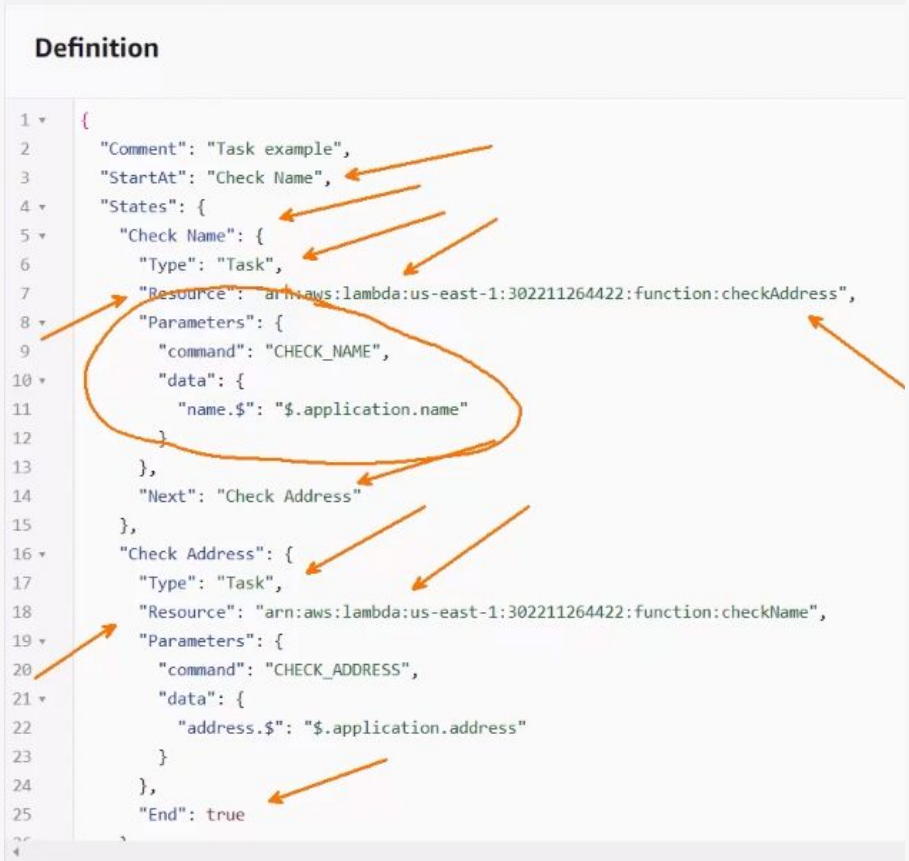
Definition



```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
```

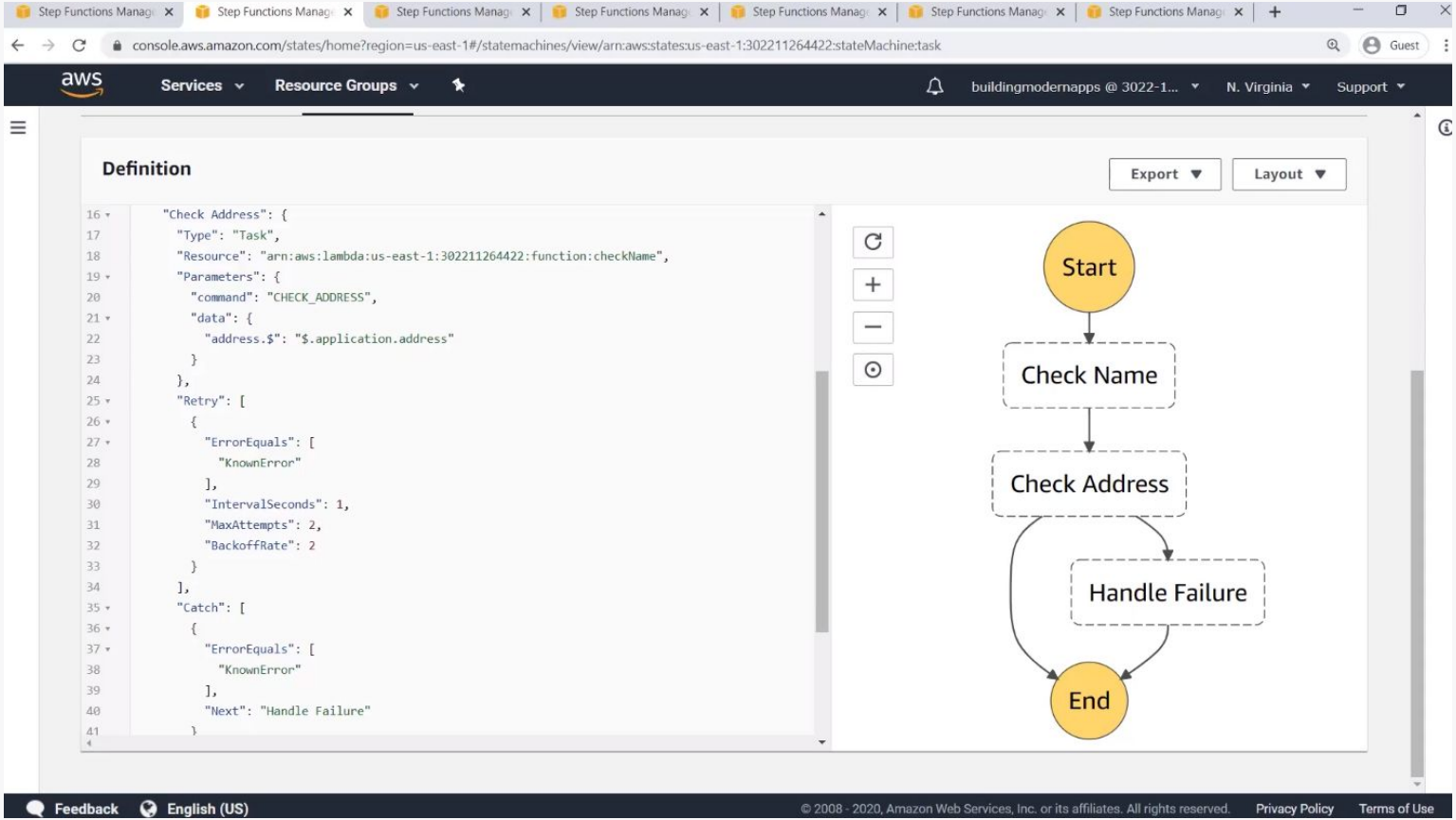
End is used to indicate that the current state is a terminal state, meaning it's the last state in the state machine. When a state is marked as an "End" state, it signifies that the state machine has completed its execution.

Definition

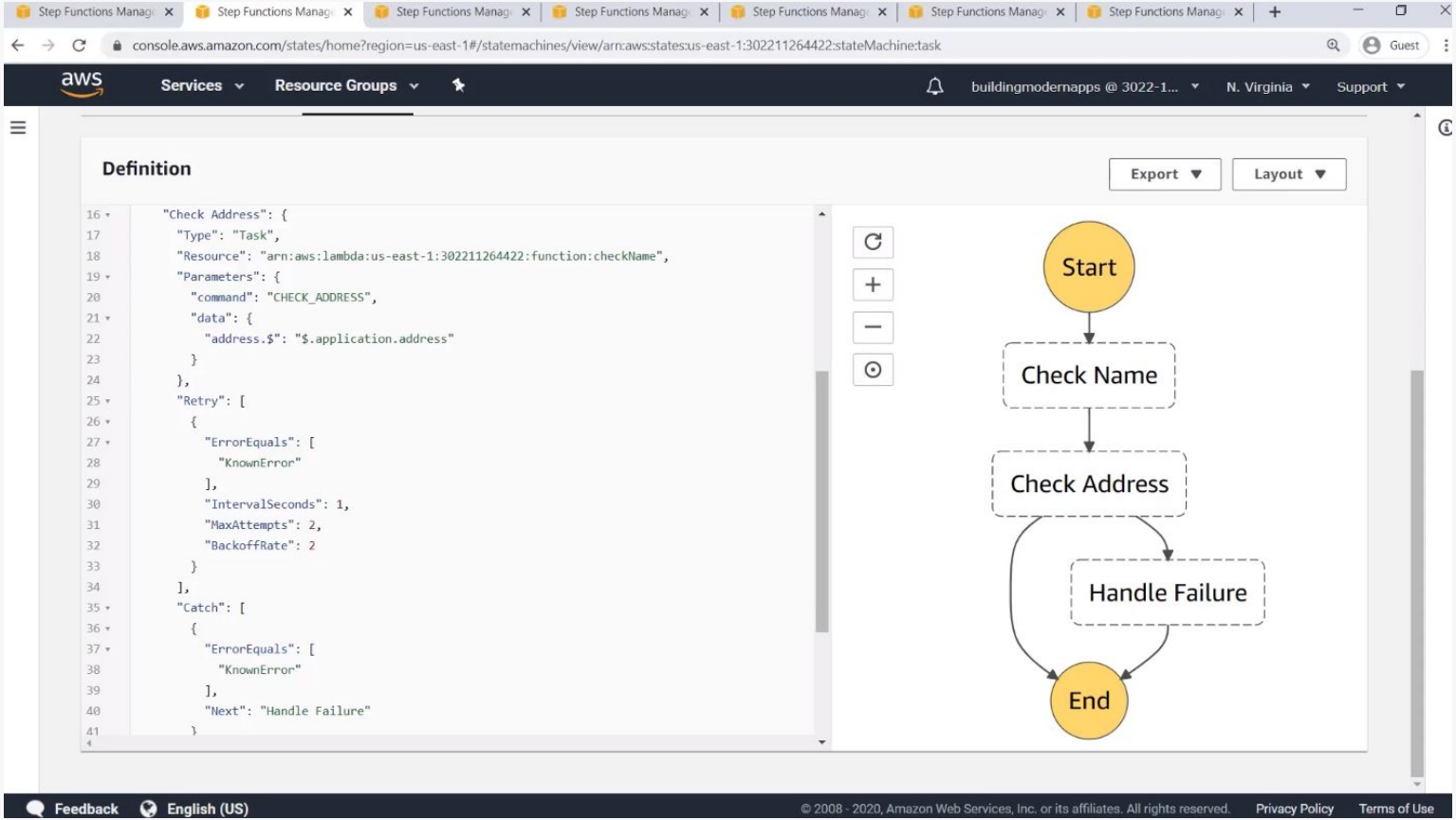


```
{
  "Comment": "Task example",
  "StartAt": "Check Name",
  "States": {
    "Check Name": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
      "Parameters": {
        "command": "CHECK_NAME",
        "data": {
          "name.$": "$.application.name"
        }
      },
      "Next": "Check Address"
    },
    "Check Address": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkName",
      "Parameters": {
        "command": "CHECK_ADDRESS",
        "data": {
          "address.$": "$.application.address"
        }
      },
      "End": true
    }
```

In the Parameters field, you map input parameters using the "inputParam.$": "$.inputParam" syntax.



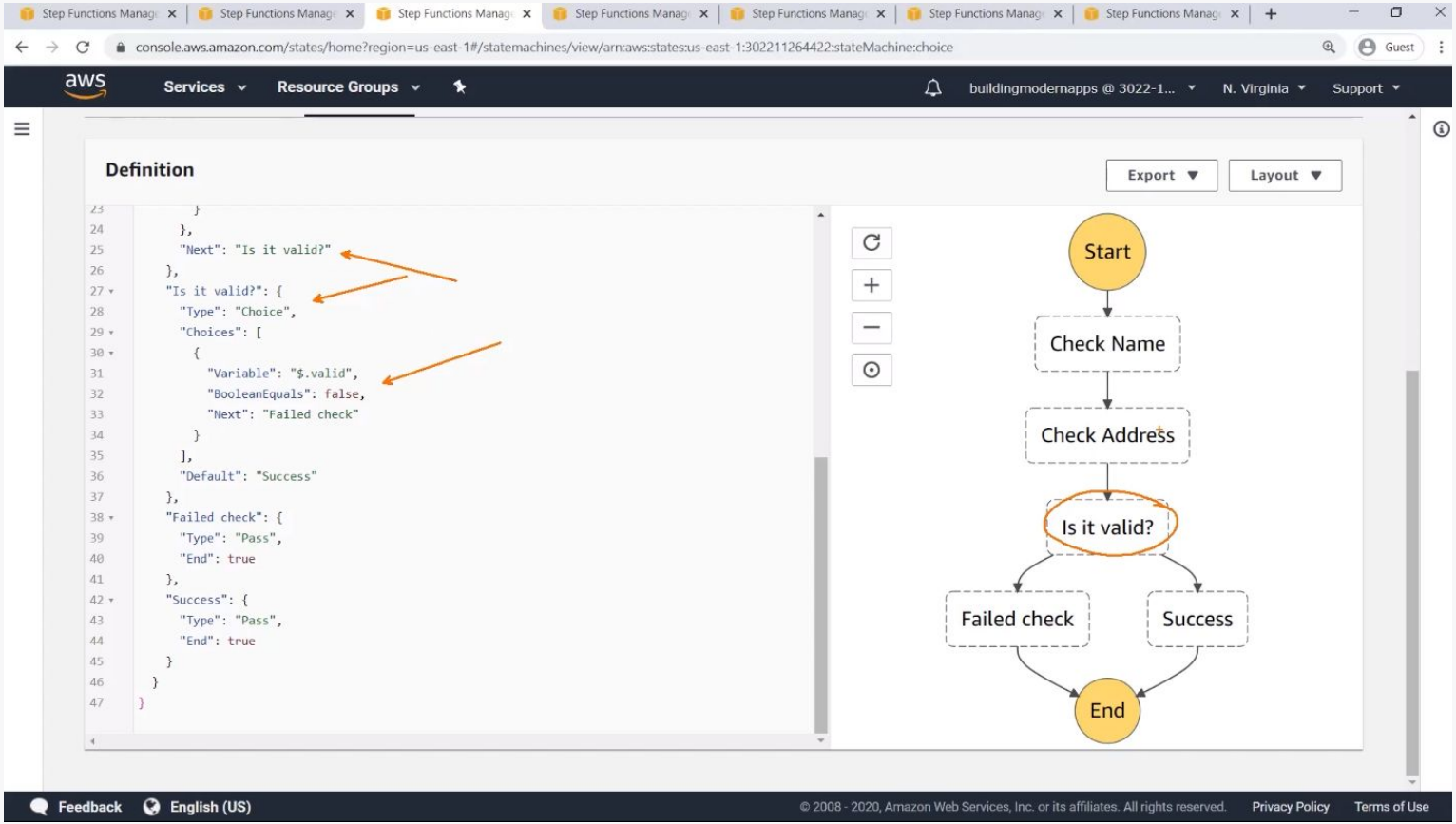
Let’s add another state here to Handle Failure



The **retry** parameter is used to define how many retry attempts, back off rate, and intervals between items when an error is caught.

Choice State: This state adds branching logic to your workflow.

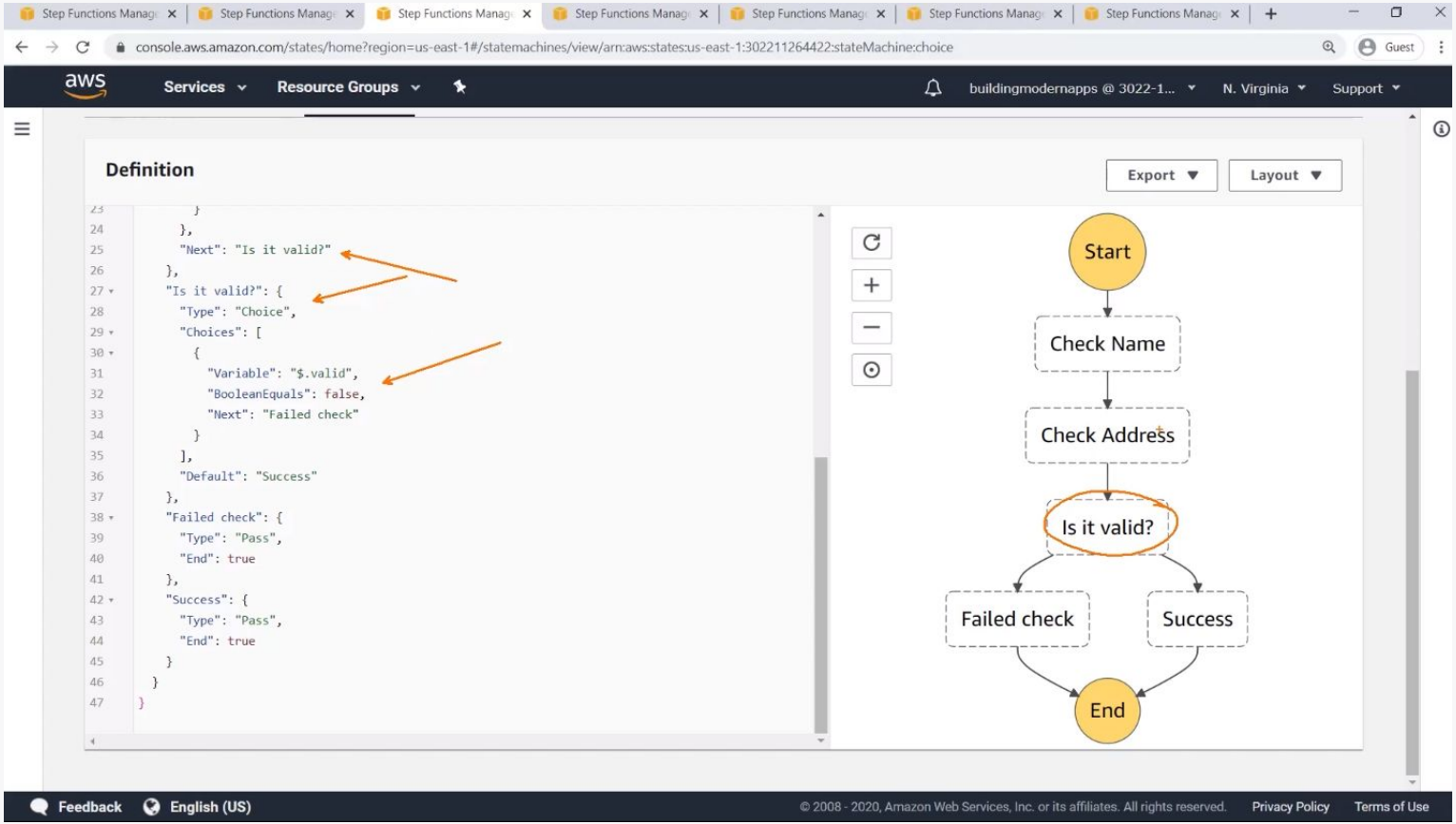
It allows you to make decisions based on the input and transition to different states accordingly.



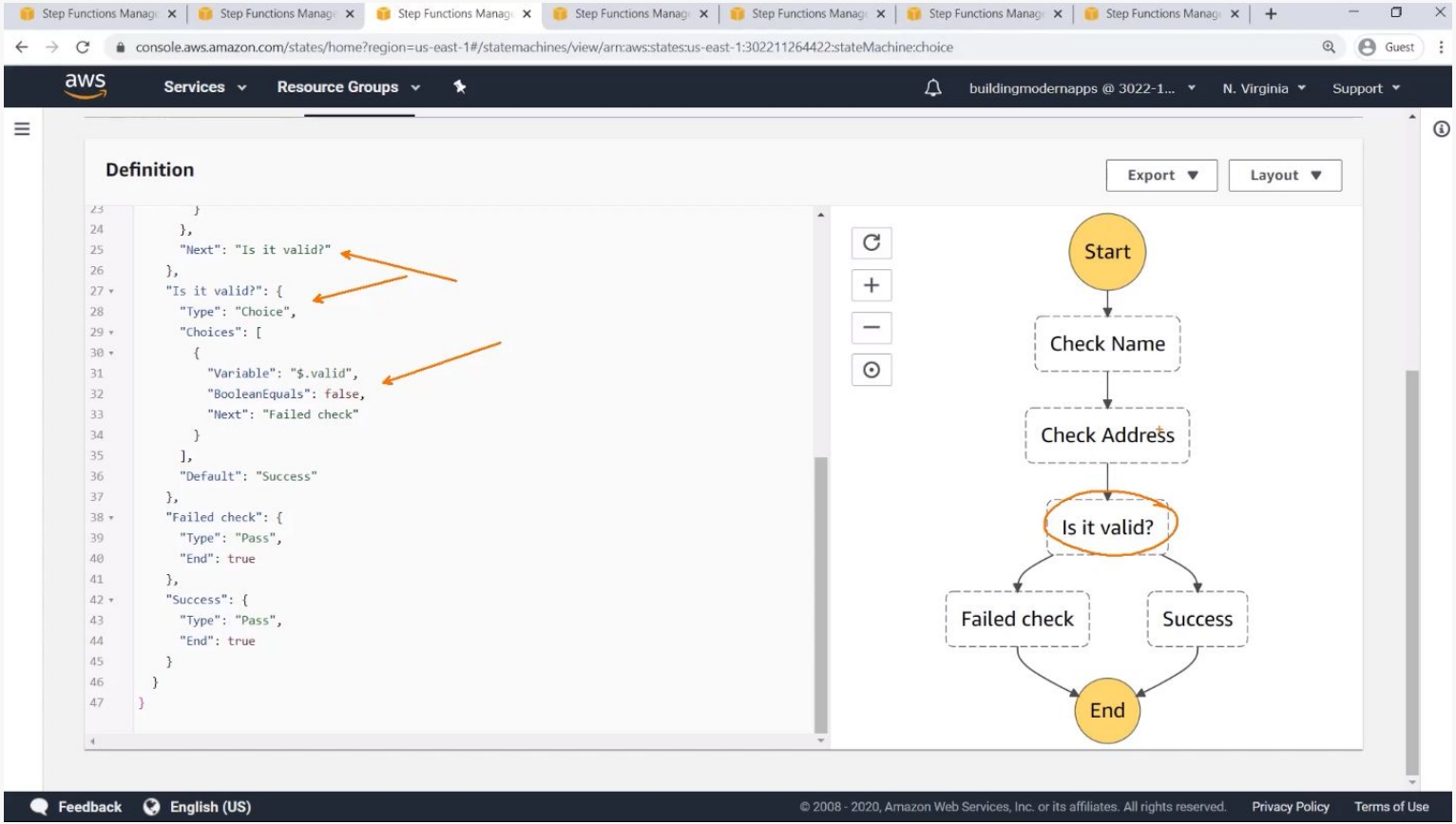
A sample of Choice State type

Pass State: This state simply passes its input to its output.

It's often used for debugging or to transform the input in some way.



In this simple examples, the Success and Failed Check states are set to Pass for simplicity.



why check name and check address are done in a serial way? Check address doesn't need the output of check name to work. They could be done in parallel.

Parallel State: This state enables parallel execution of multiple branches of your workflow. Each branch is a separate state.

Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag

console.aws.amazon.com/states/home?region=us-east-1#/statemachines/view/arn:aws:states:us-east-1:302211264422:stateMachine:parallel

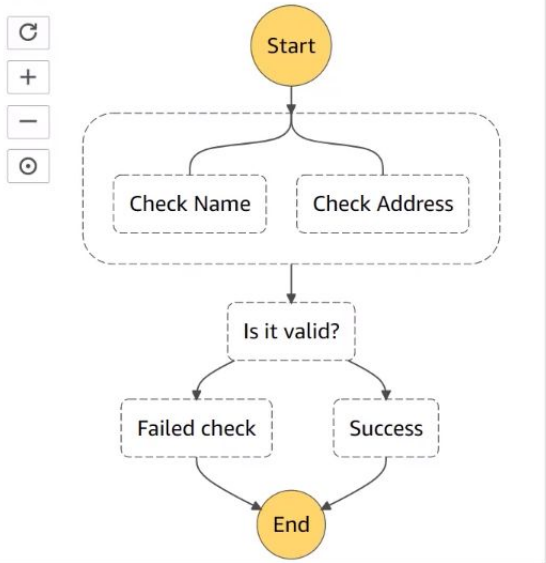
Guest

aws Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Support

Definition

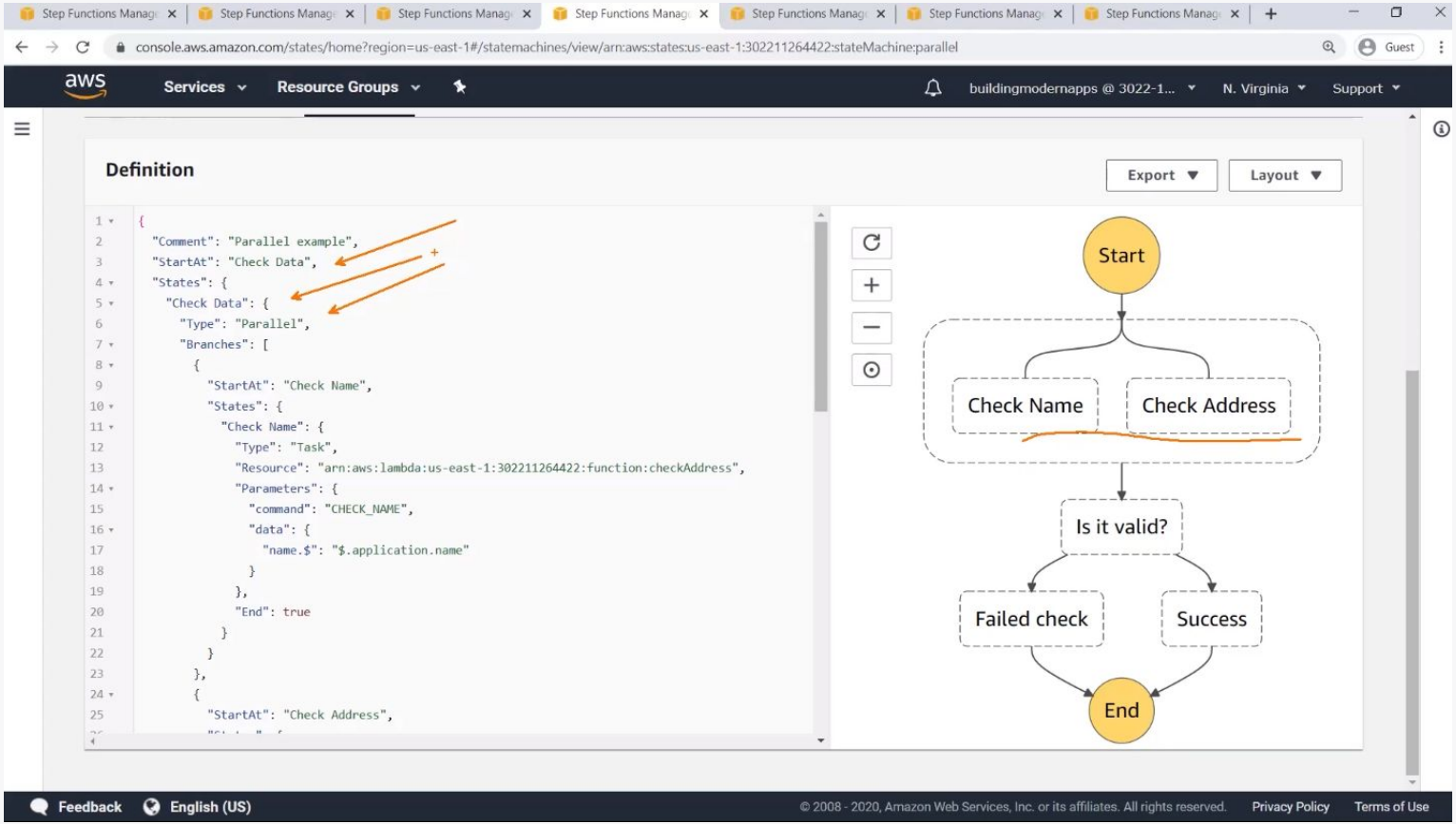
Export Layout

```
{
  "Comment": "Parallel example",
  "StartAt": "Check Data",
  "States": {
    "Check Data": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Check Name",
          "States": {
            "Check Name": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
              "Parameters": {
                "command": "CHECK_NAME",
                "data": {
                  "name.$": "$.application.name"
                }
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Check Address",
```

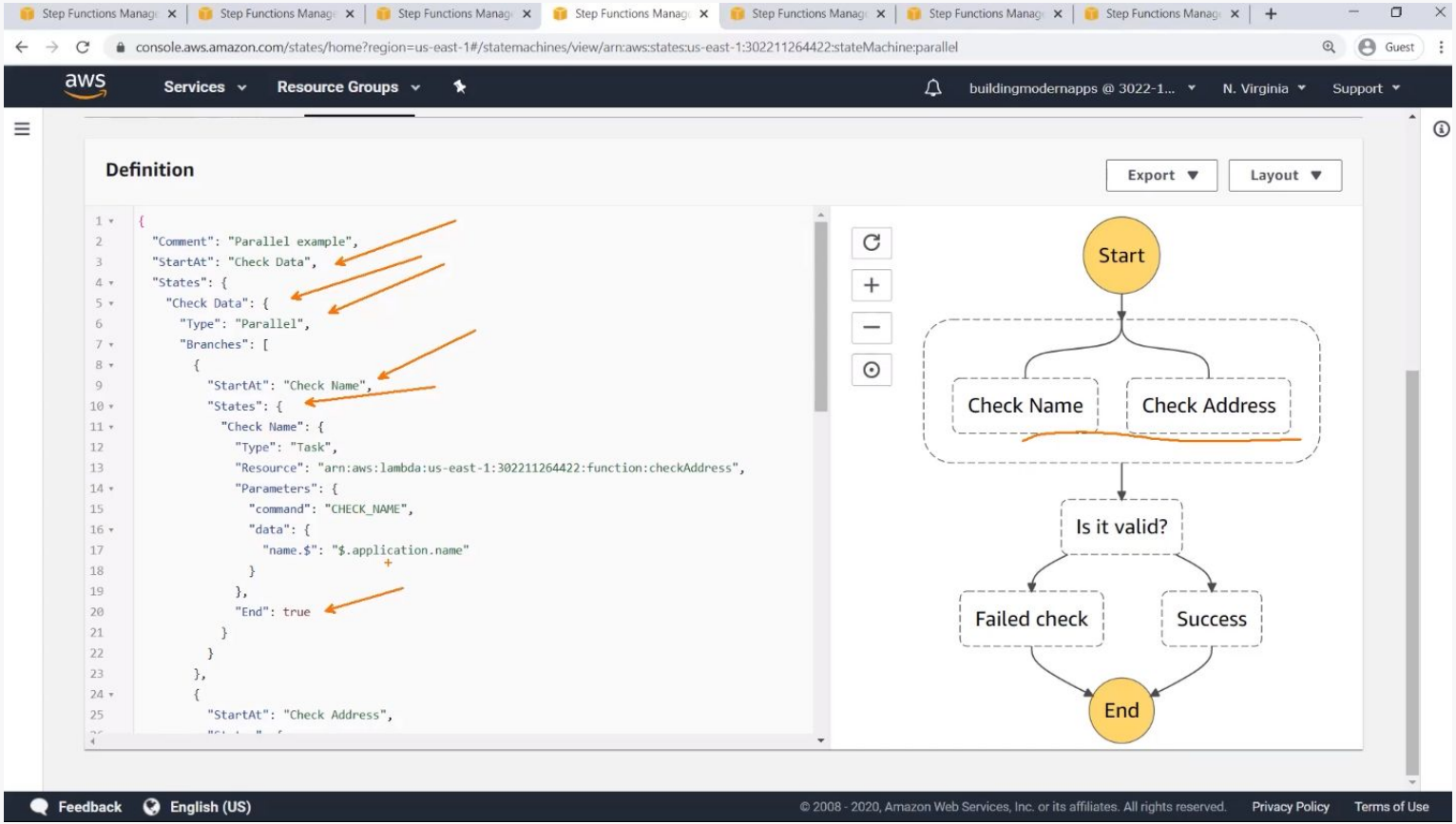


Feedback English (US)

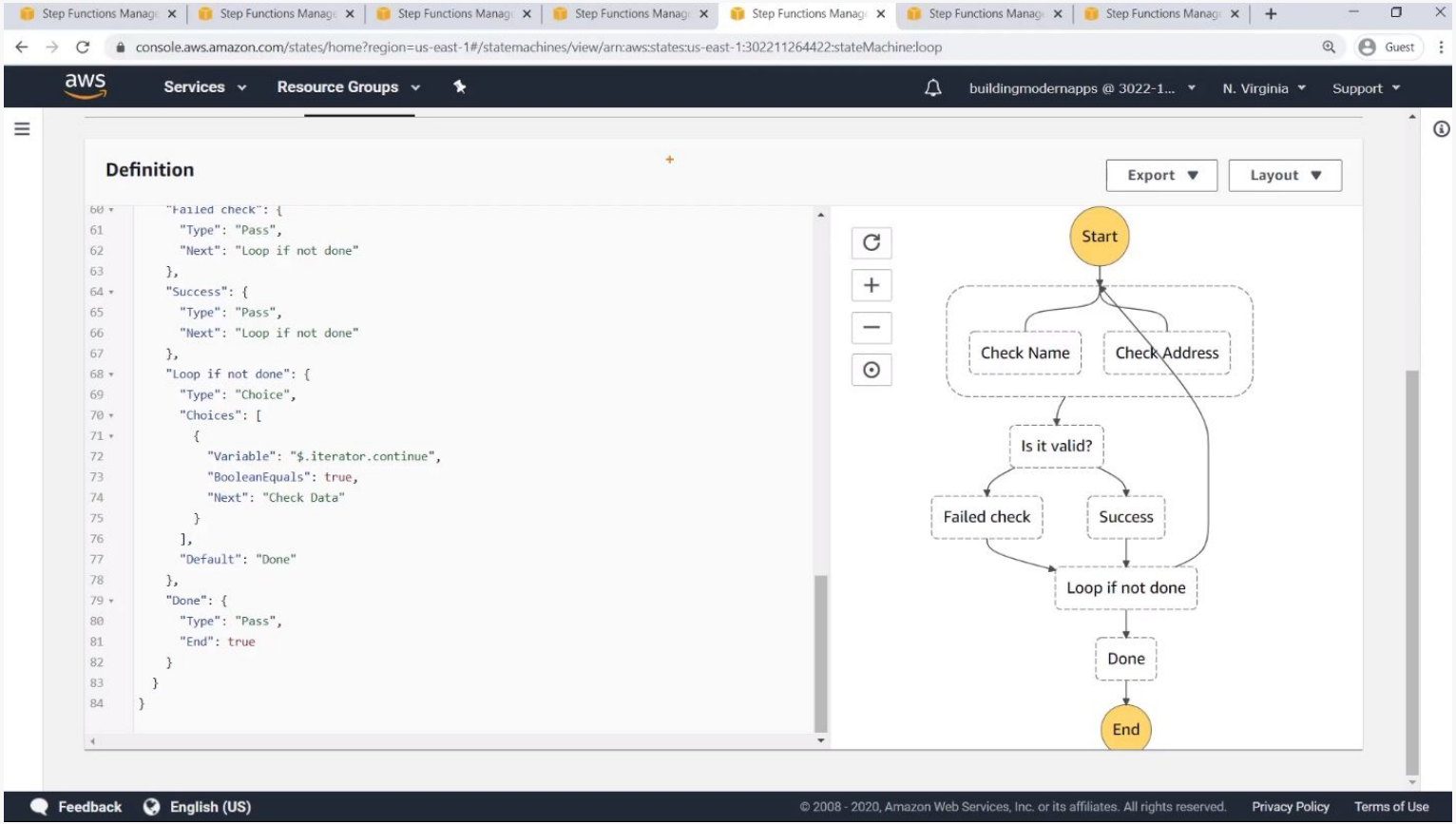
Privacy Policy Terms of Use



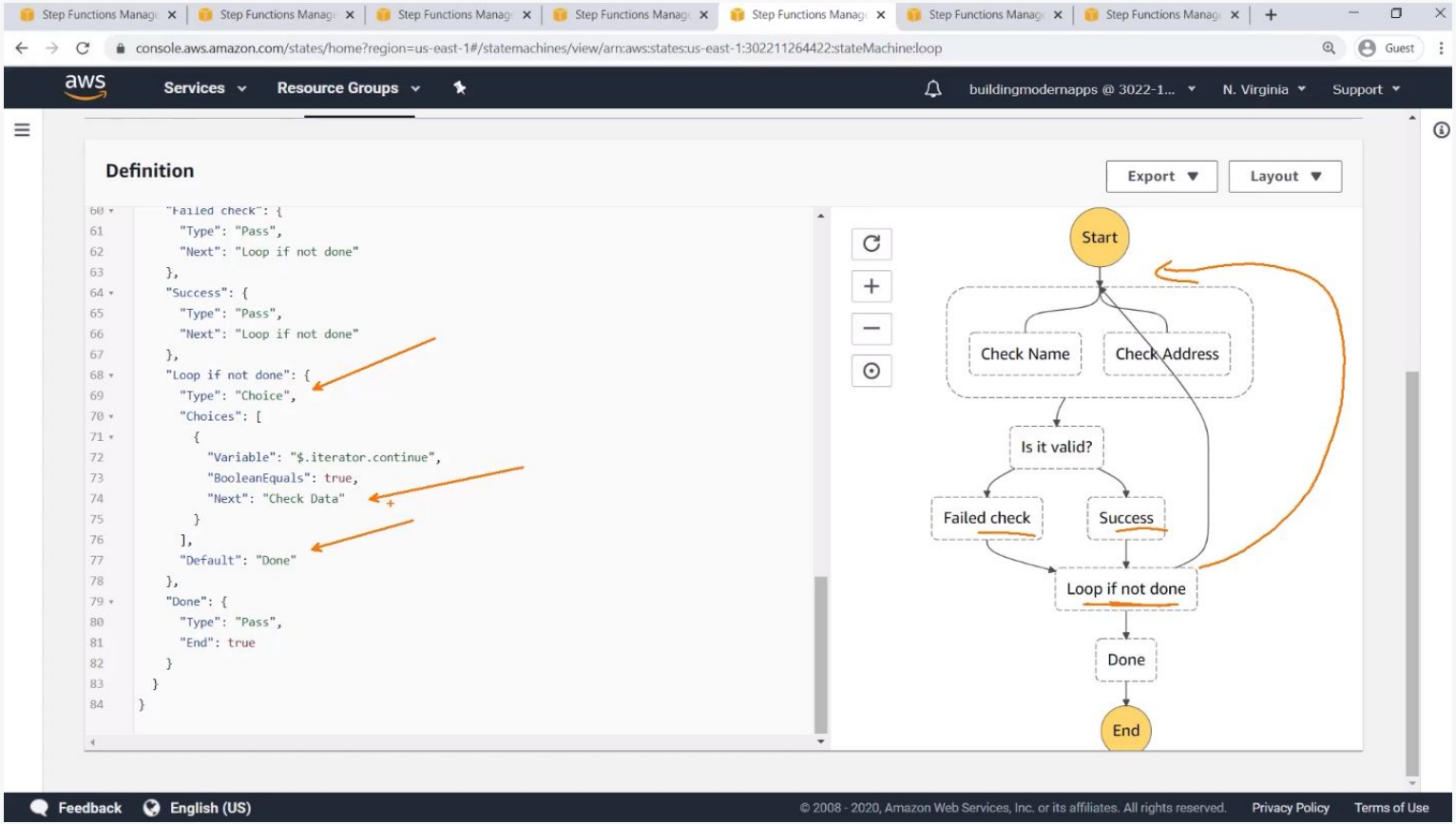
StartAt state is set to “Check Data” which is of type parallel.



This has branches which takes an array of state machines.

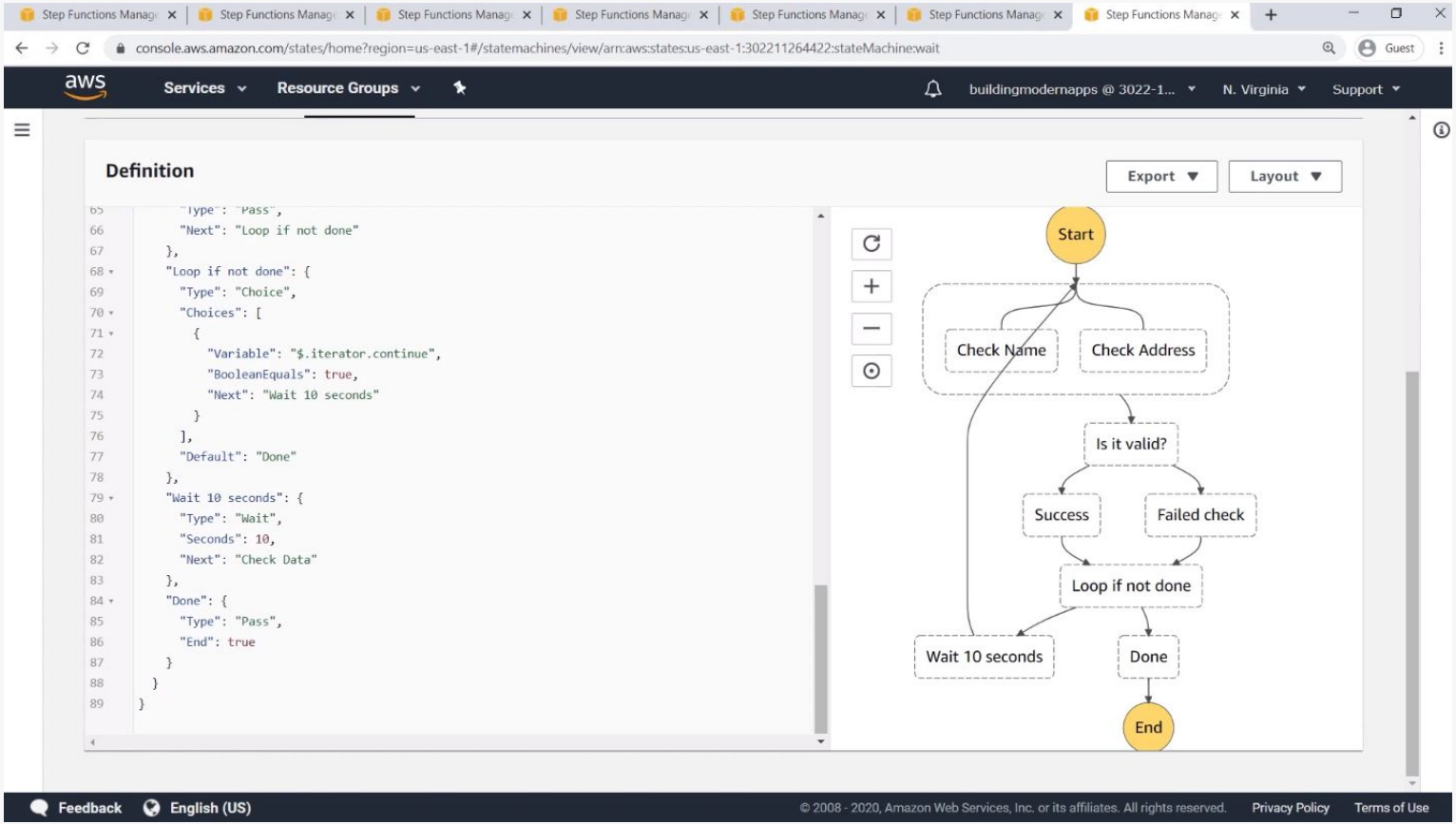


we can add a loop to the states

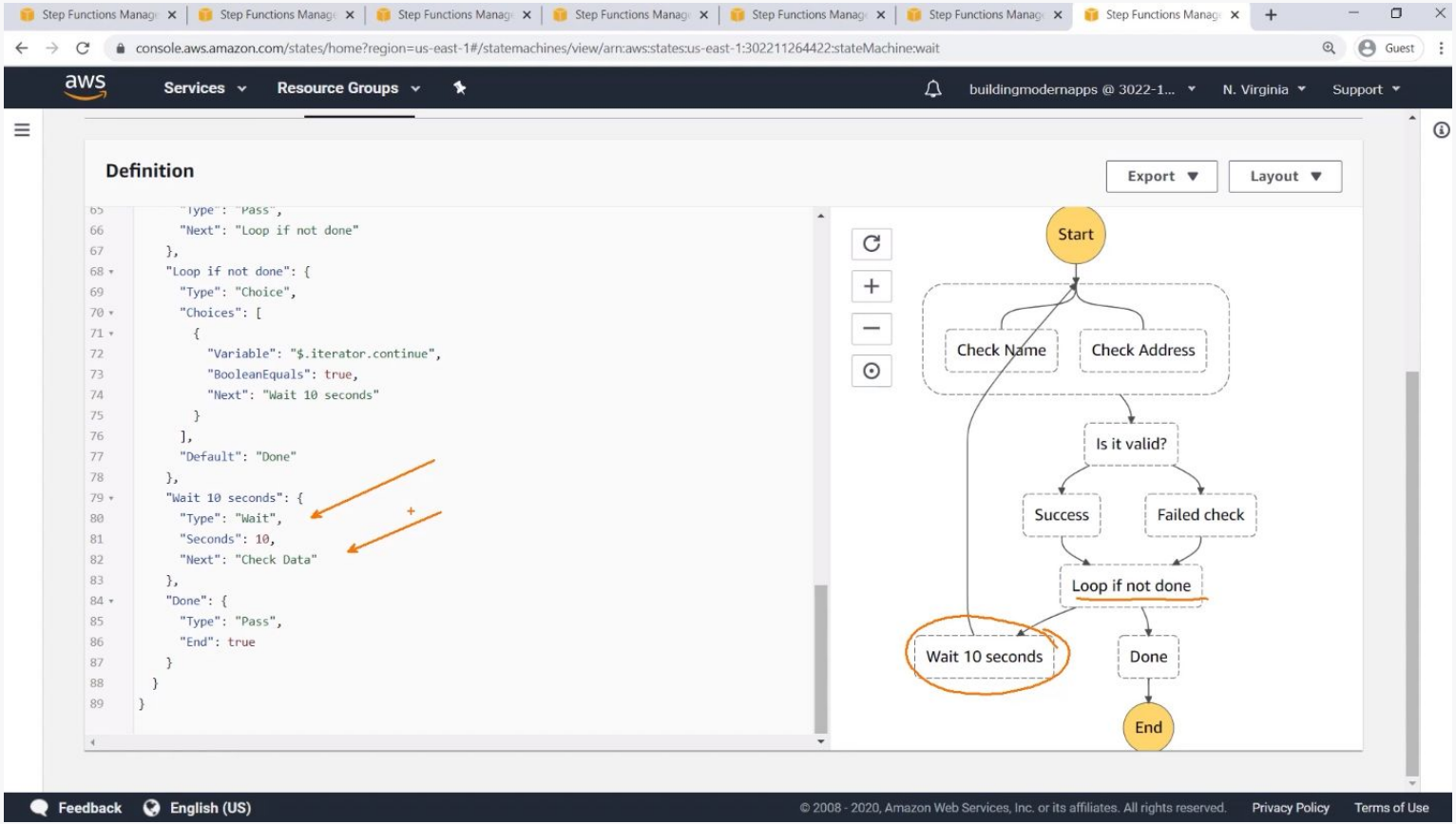


note that we created loop using a choice type. It's not a new step type.

Wait State: This state inserts a delay or pause into your workflow. It can wait for a specified duration or until a specified timestamp.


Step Functions Manag
console.aws.amazon.com/states/home?region=us-east-1#/statemachines/view/arn:aws:states:us-east-1:302211264422:stateMachine:wait
Guest
aws
Services
Resource Groups
buildingmodernapps @ 3022-1...
N. Virginia
Support
Definition
Export
Layout
"Type": "Pass",
"Next": "Loop if not done"
},
"Loop if not done": {
"Type": "Choice",
"Choices": [
{
"Variable": "$.iterator.continue",
"BooleanEquals": true,
"Next": "Wait 10 seconds"
}
],
"Default": "Done"
},
"Wait 10 seconds": {
"Type": "Wait",
"Seconds": 10,
"Next": "Check Data"
},
"Done": {
"Type": "Pass",
"End": true
}
}
}
Start
Check Name
Check Address
Is it valid?
Success
Failed check
Loop if not done
Wait 10 seconds
Done
End
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

wait state



wait for 10 sec and the go to the “Check Data” state

Map State: This state enables you to iterate over a list of items and perform the same set of operations on each item.

Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag | Step Functions Manag

console.aws.amazon.com/states/home?region=us-east-1#/statemachines/view/arn:aws:states:us-east-1:302211264422:stateMachine:map

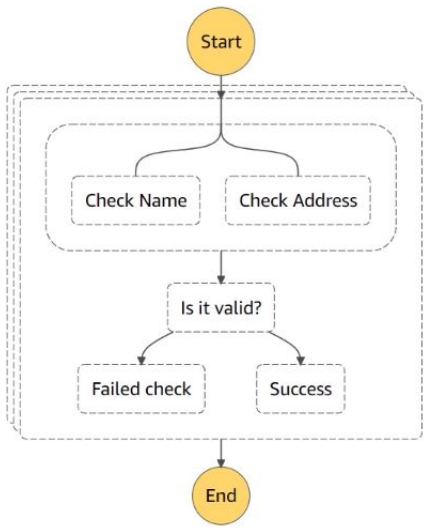
Guest

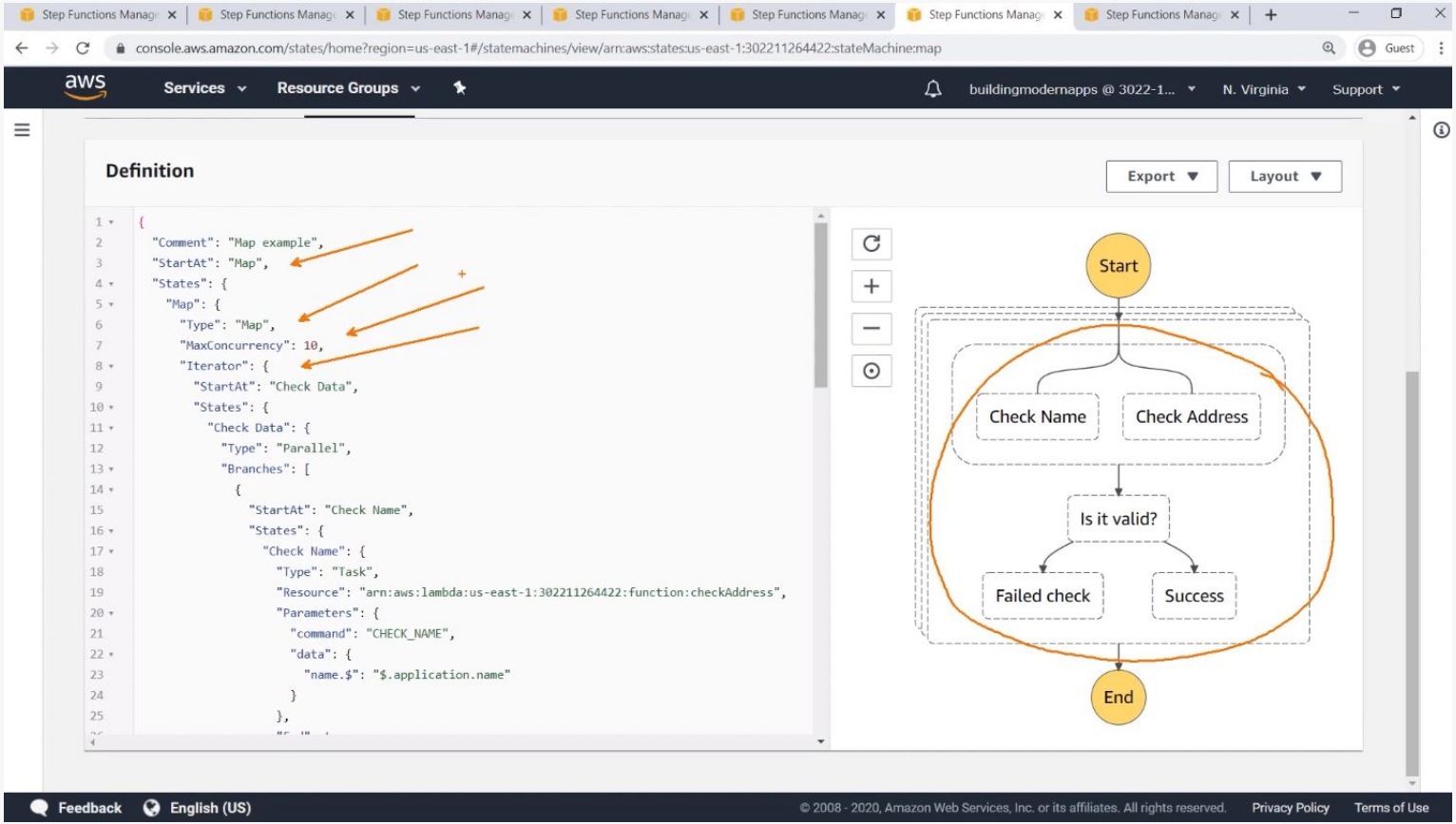
aws Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Support

Definition

Export Layout

```
{
  "Comment": "Map example",
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "MaxConcurrency": 10,
      "Iterator": {
        "StartAt": "Check Data",
        "States": {
          "Check Data": {
            "Type": "Parallel",
            "Branches": [
              {
                "StartAt": "Check Name",
                "States": {
                  "Check Name": {
                    "Type": "Task",
                    "Resource": "arn:aws:lambda:us-east-1:302211264422:function:checkAddress",
                    "Parameters": {
                      "command": "CHECK_NAME",
                      "data": {
                        "name.$": "$.application.name"
                      }
                    },
```





the same state as before into a iterator

The last two state types are for stopping the execution of a state machine.

Succeed and Fail

Succeed State: This state immediately stops the execution of the state machine and marks it as successful.

Fail State: This state immediately stops the execution of the state machine and marks it as failed.

Eight state types of state function:

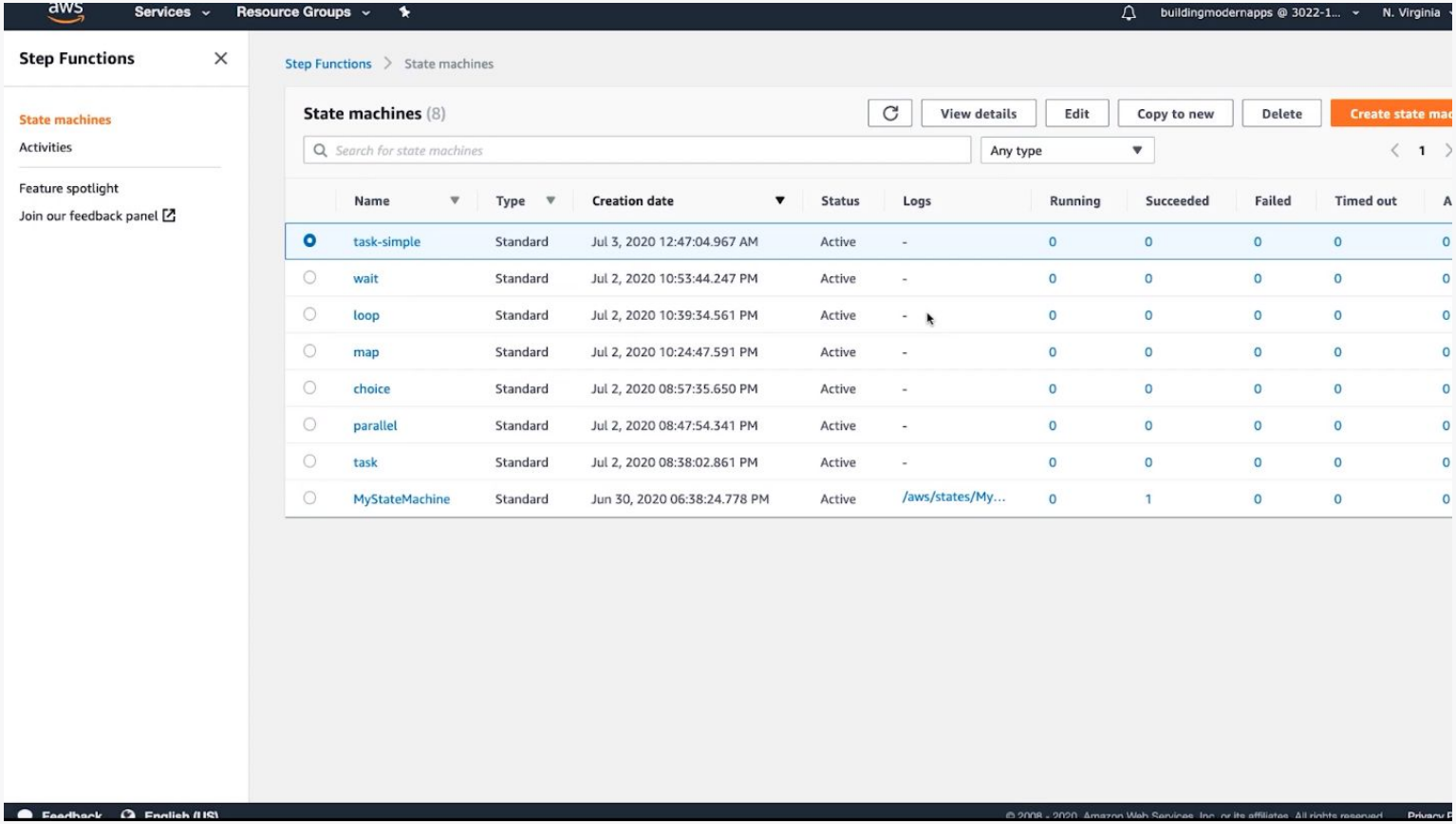
- Task: Do some work in your state machine
- Choice: Make a choice between branches of execution
- Parallel: Begin parallel branches of execution
- Map: Dynamically iterate steps
- Pass: Simply pass its input to its output or inject some fixed data
- Wait: Provide a delay for a certain amount of time or until a specified time/date
- success and failure: Stop an execution with a failure or success

Step Functions for Dragon App

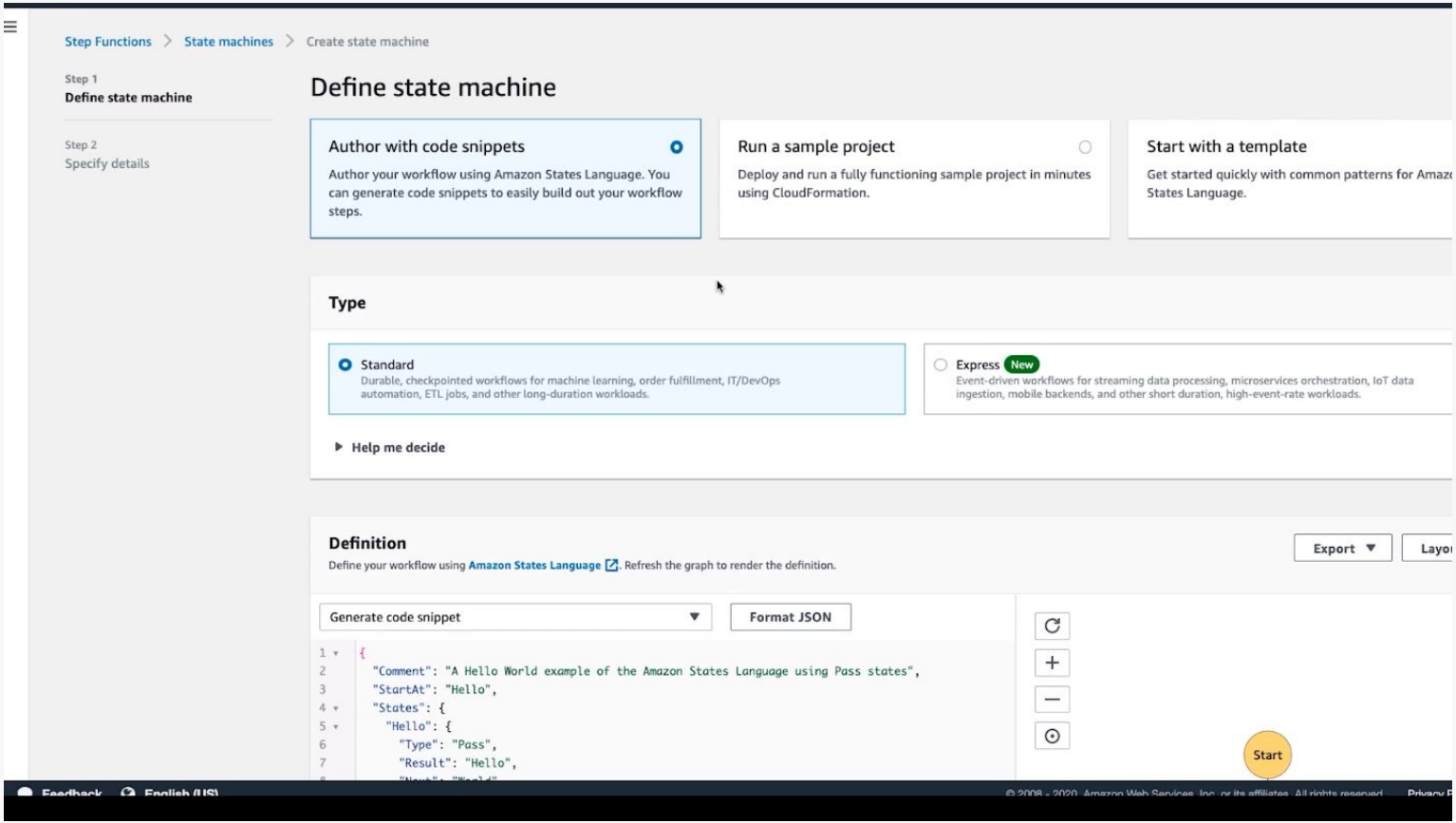
States are the individual elements of a state machine.

Each state can make decisions based on its input,
perform actions and pass output to other states.

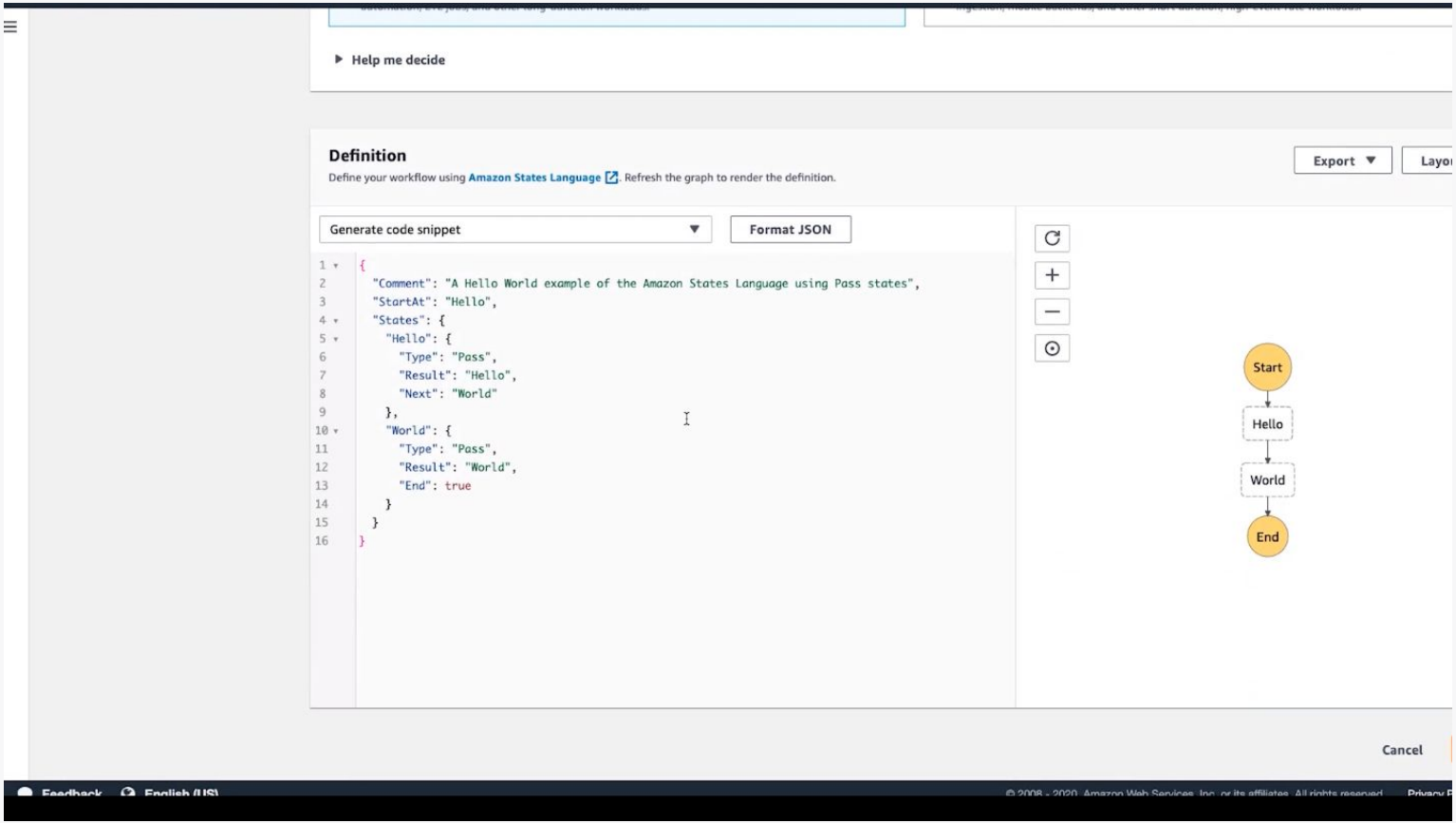
Tasks (workers) represent one unit of work done through activities or AWS services.



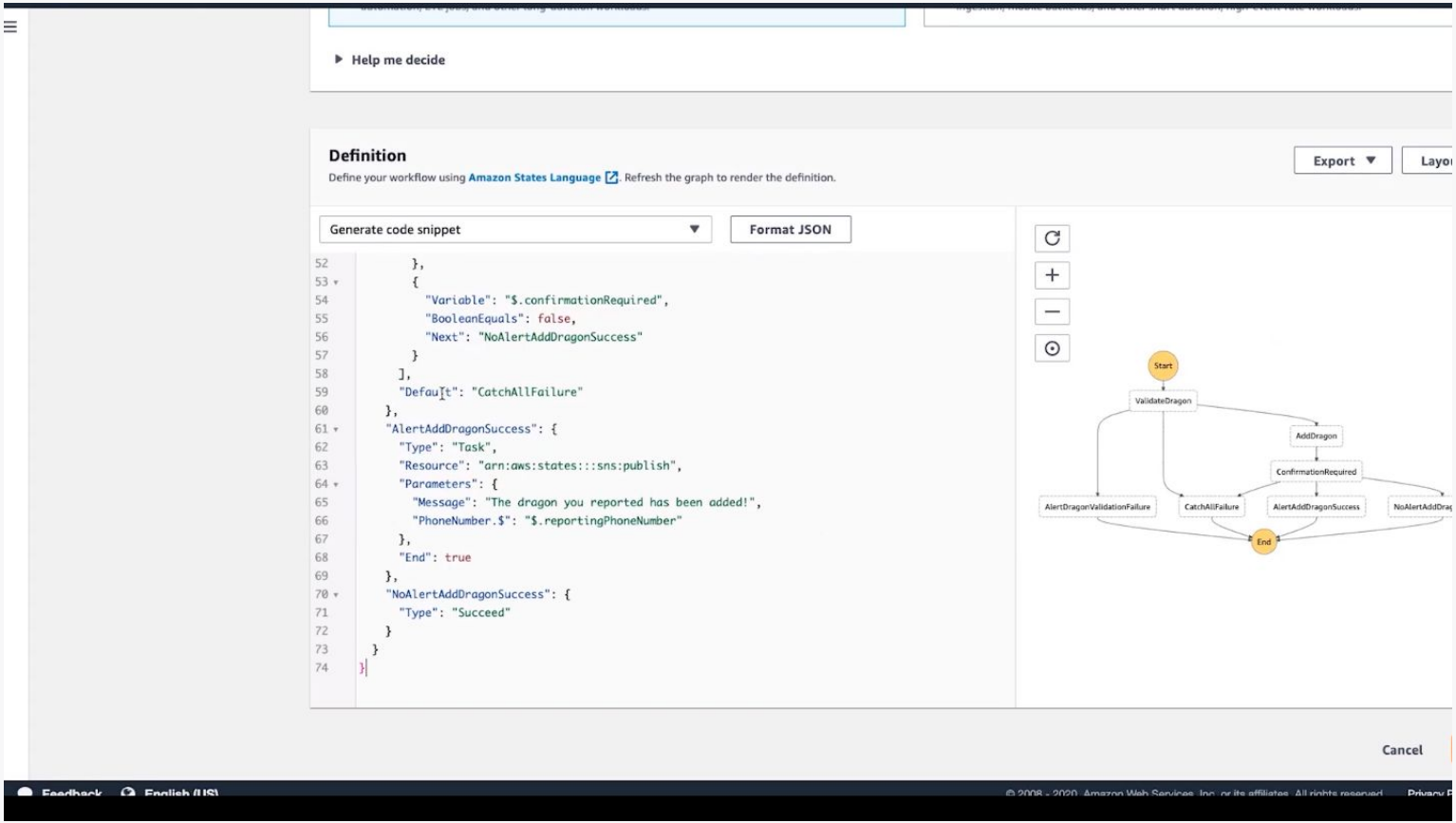
Step Function console with some state machines



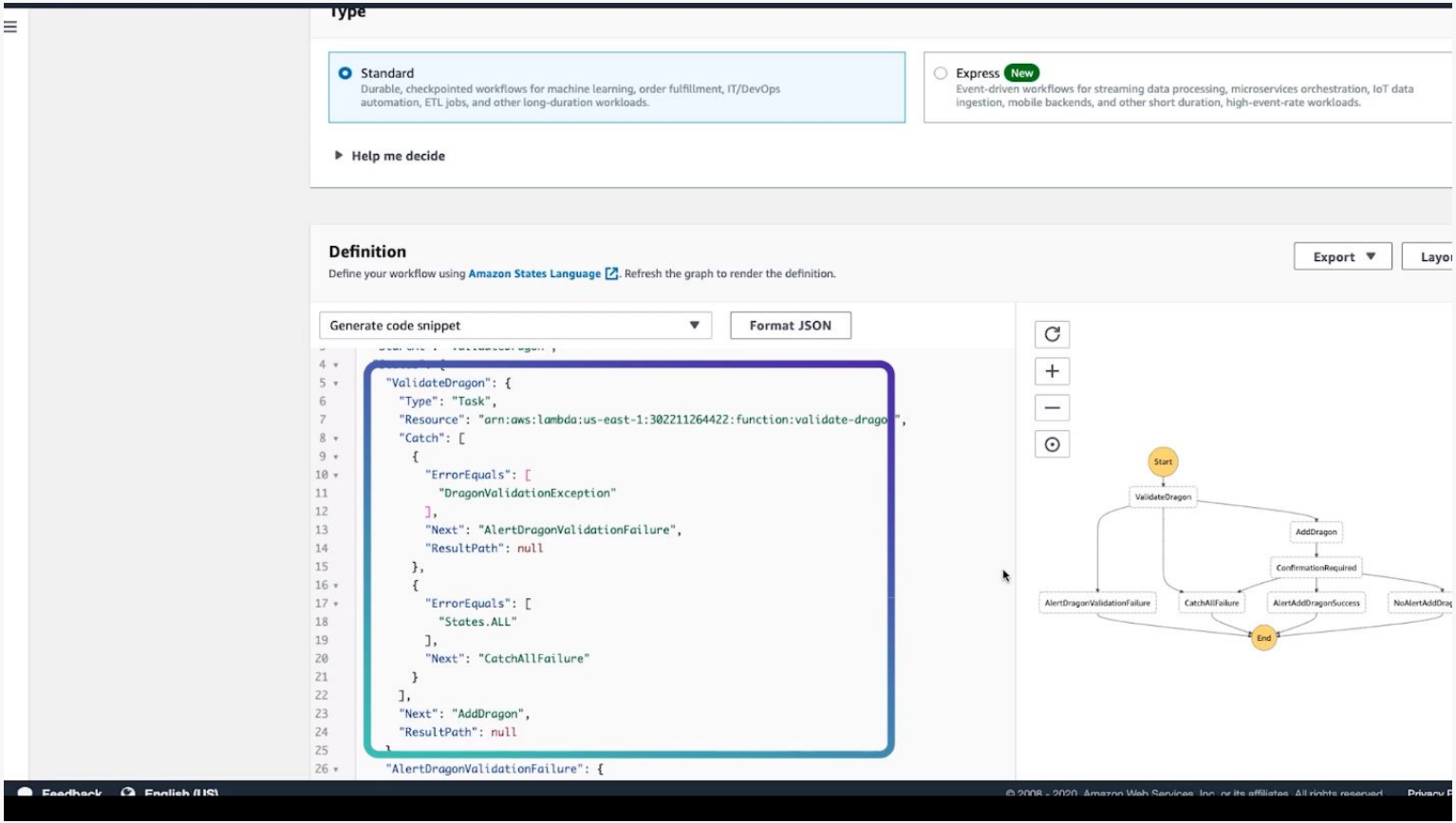
Create a new state machine



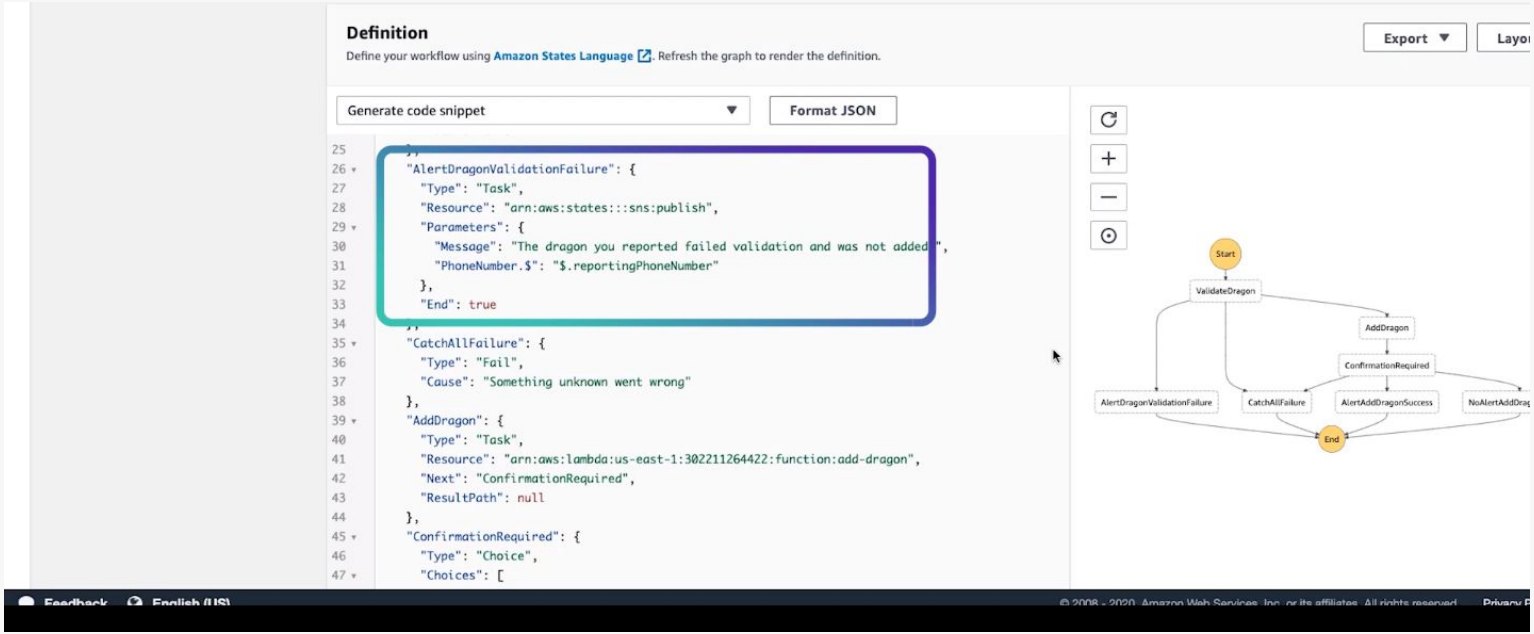
This is the default state machine using Amazon State Language in JSON



Dragon App state machine



the first step is the ValidateDragon state



Alert a user that their dragon reporting failed.
This is an example of integrating with **Amazon Simple Notification Service** or SNS.

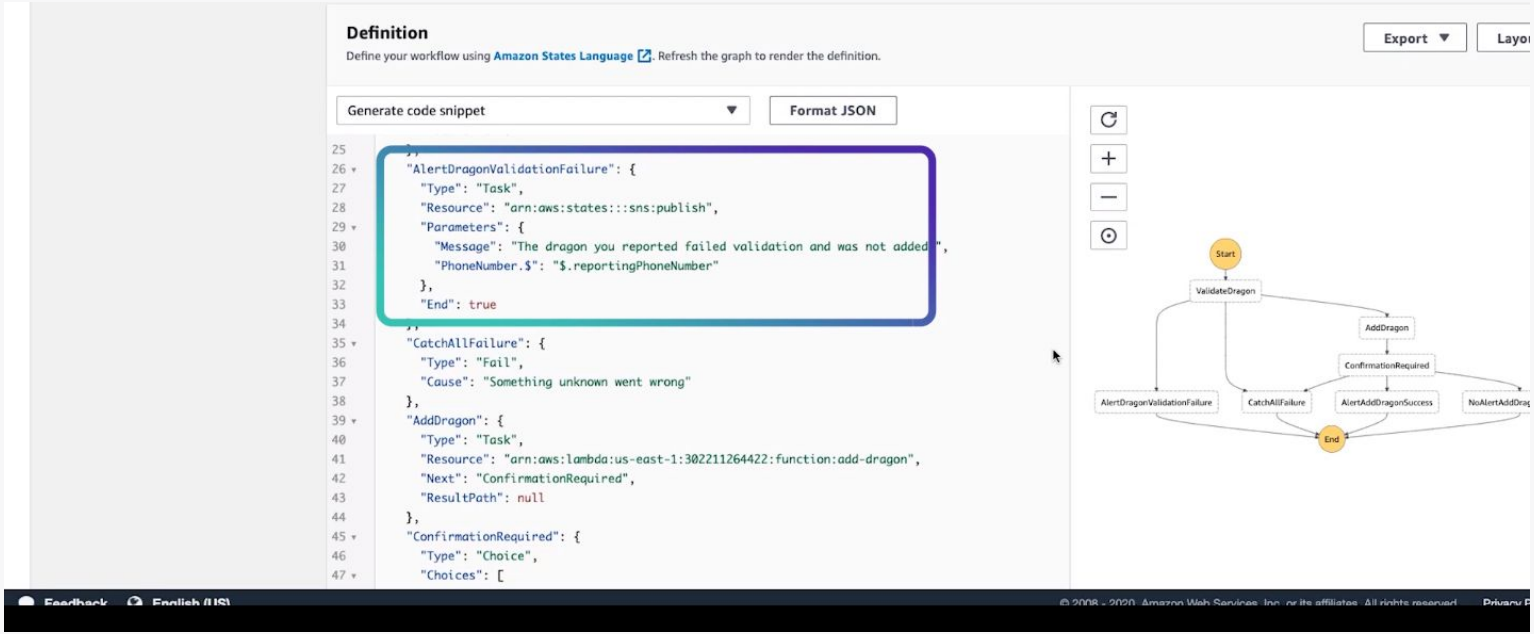
By including the $ after the parameter name and then using the $. syntax, it is able to pull the phone number off of the payload being passed into the state machine execution.

This is an example of how to pull info from the path.

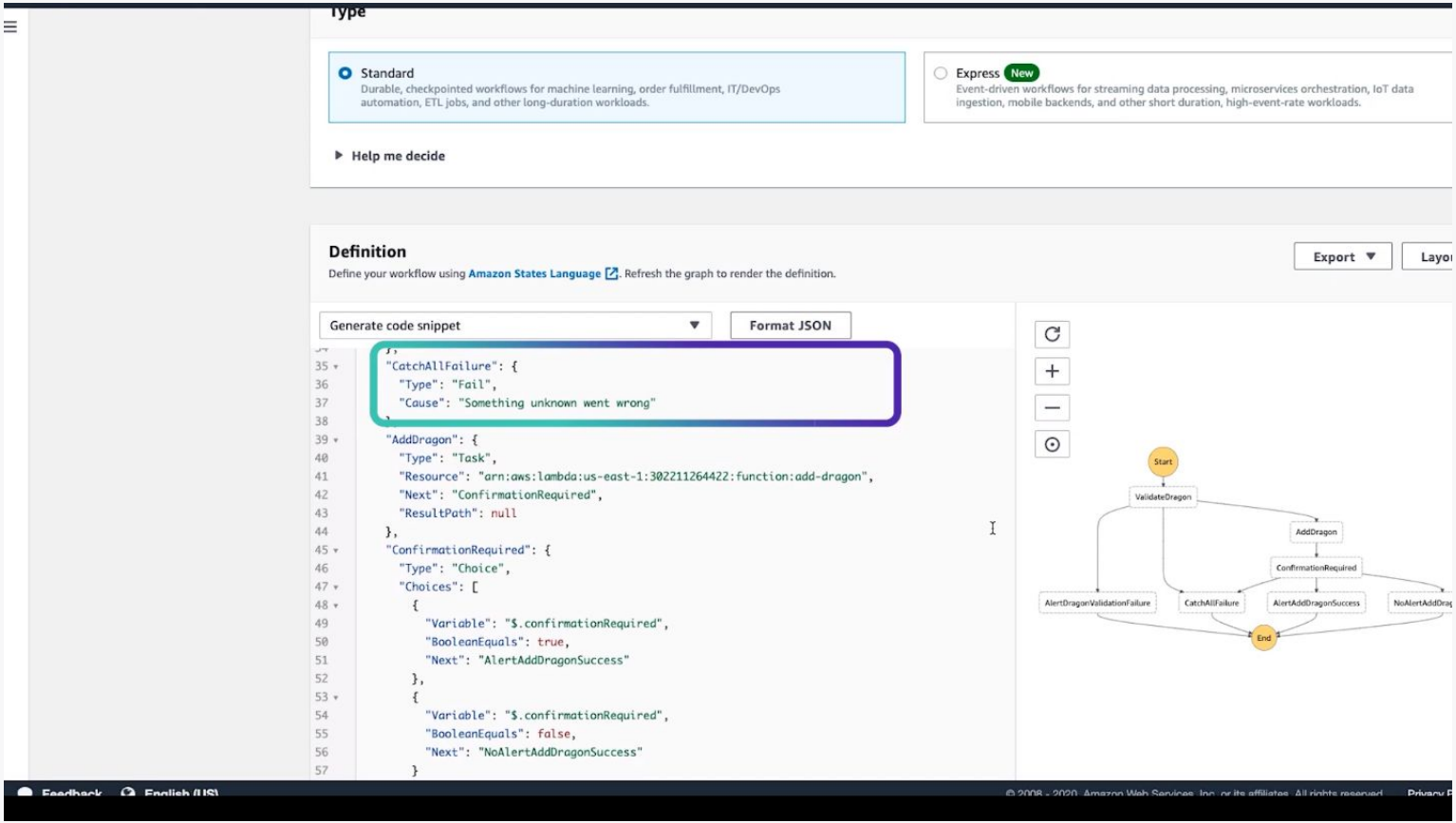
In Amazon State Language, a "Path" is used to specify a JSONPath expression that selects a specific piece of data from the input.

This selected data can then be used within the state.

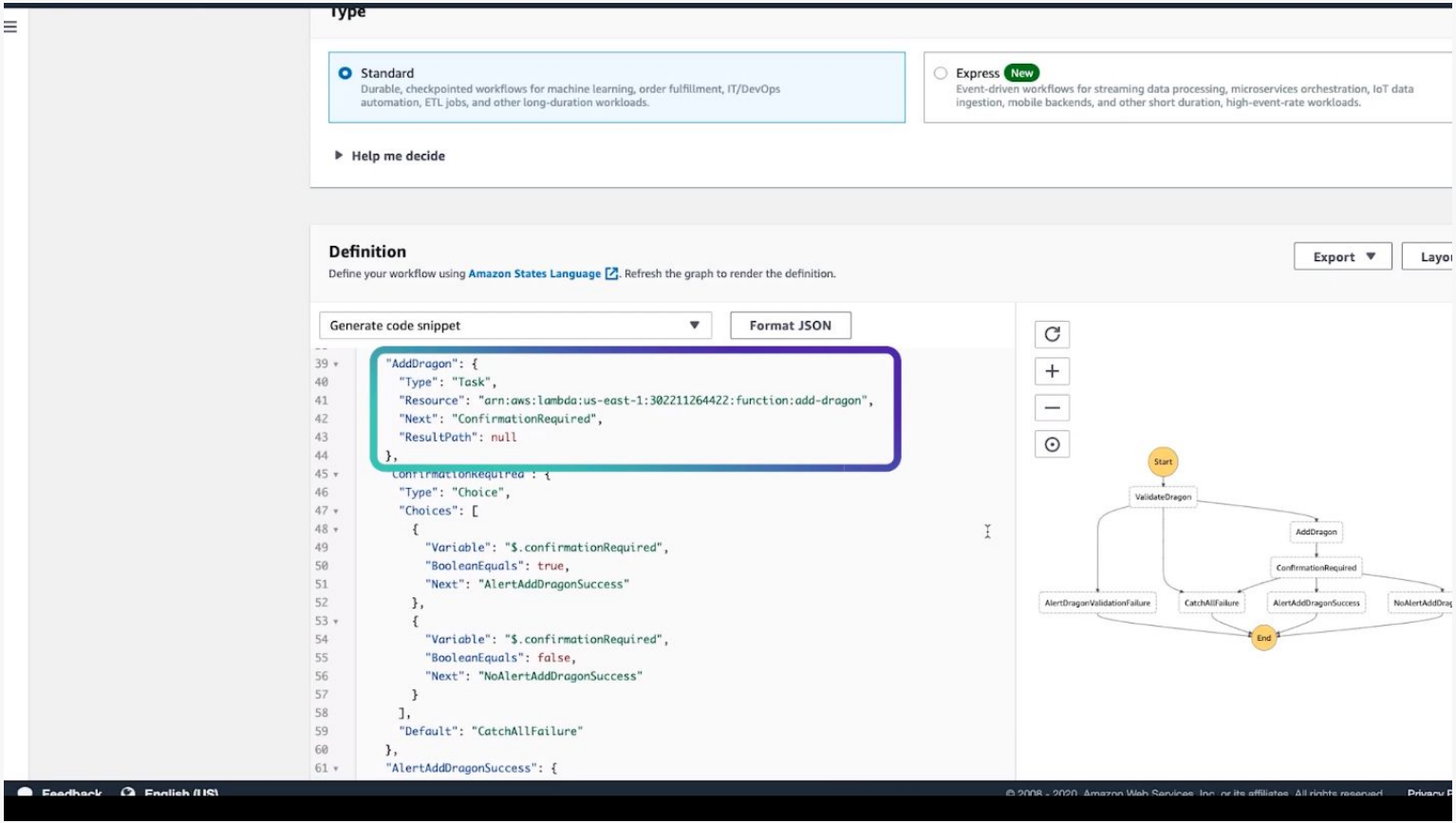
```
"Parameters": {
      "InputData.$": "$.input", // This is the Path expression
      "OutputData.$": "$.input.property" // Another example of using Path}
```



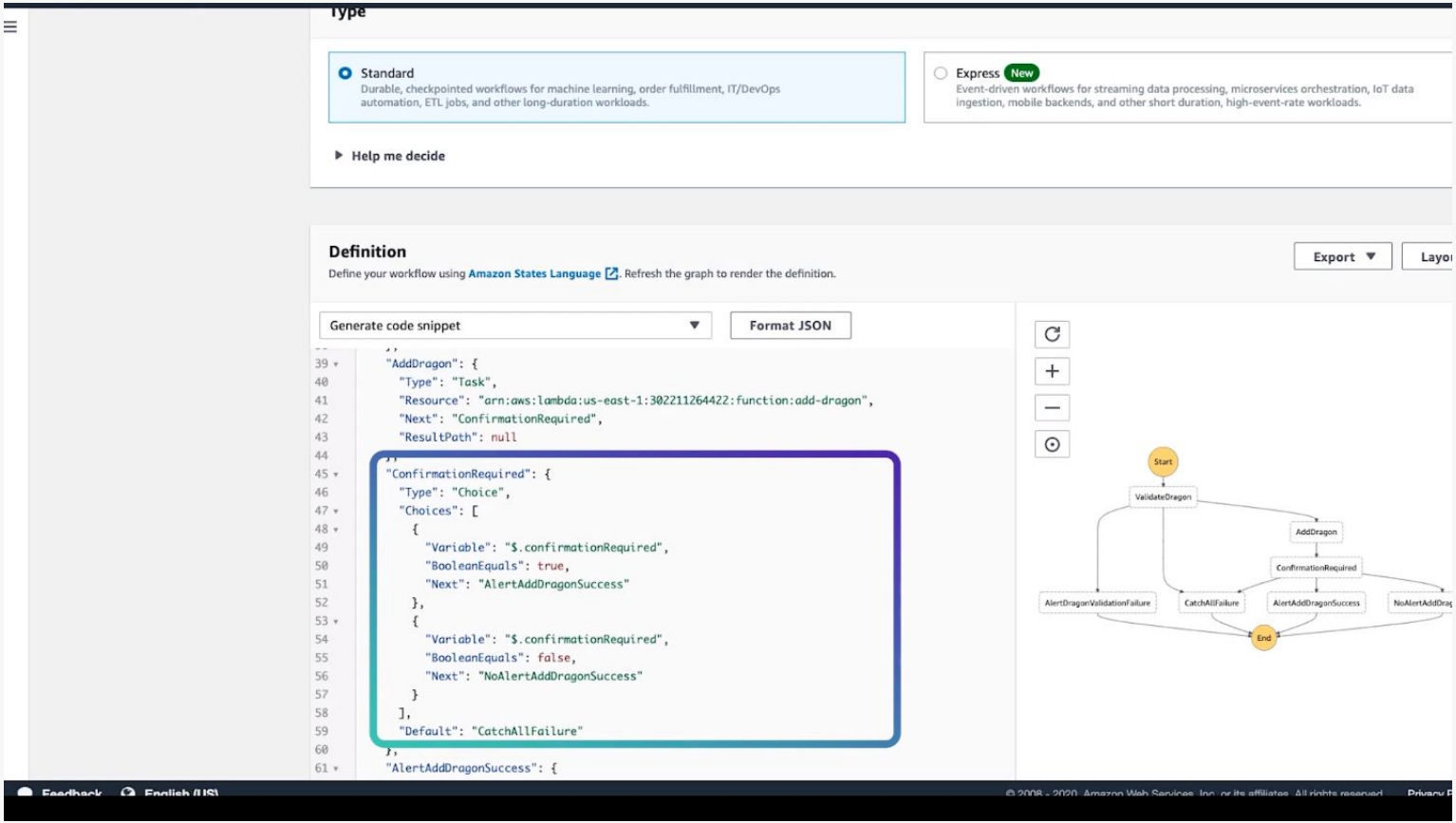
if the alert dragon validation failure is executed, the end is set to be true so that nothing else will execute and the state machine execution ends.



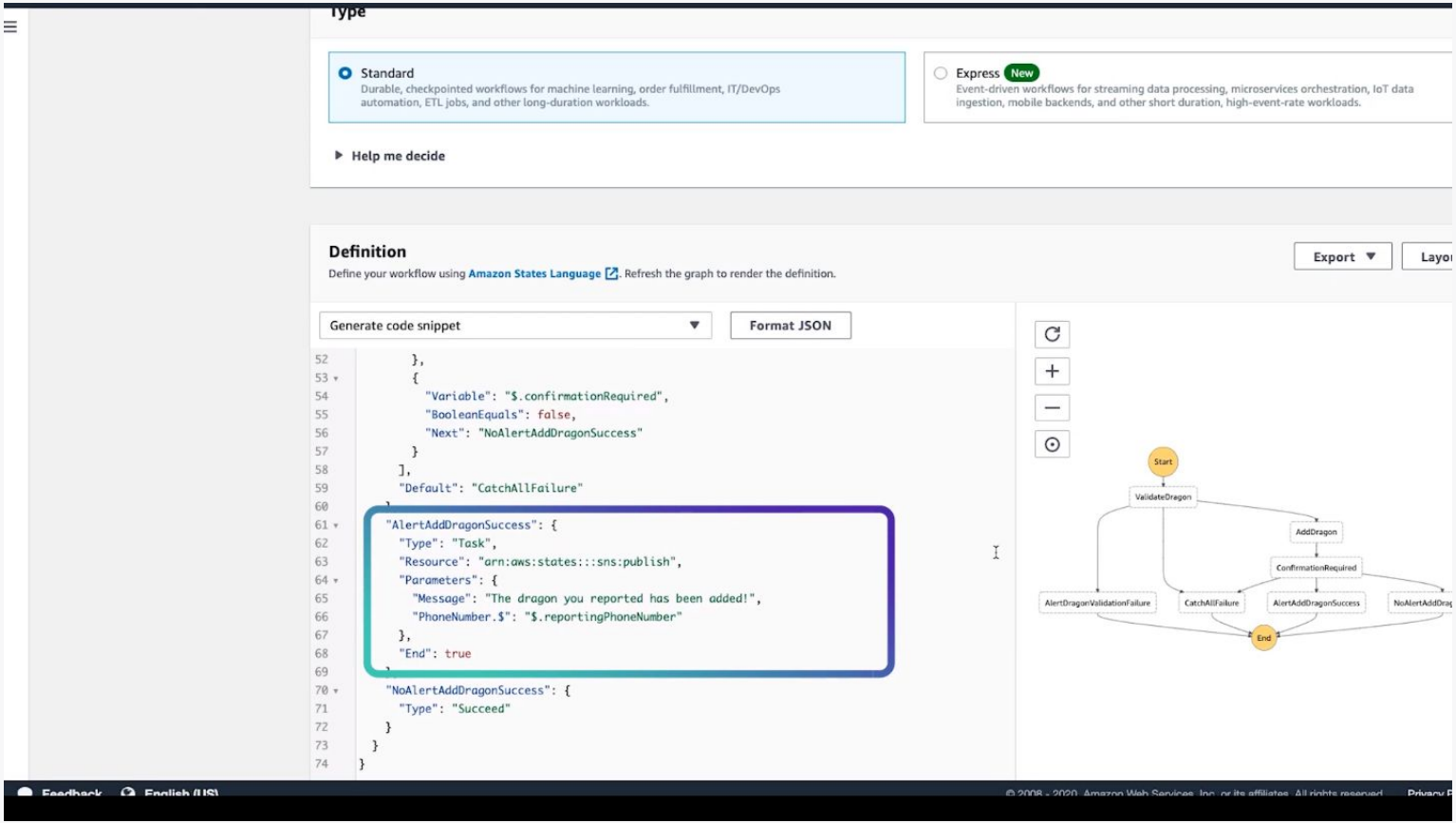
CatchAllFailure is executed if the catch caught an unknown error.



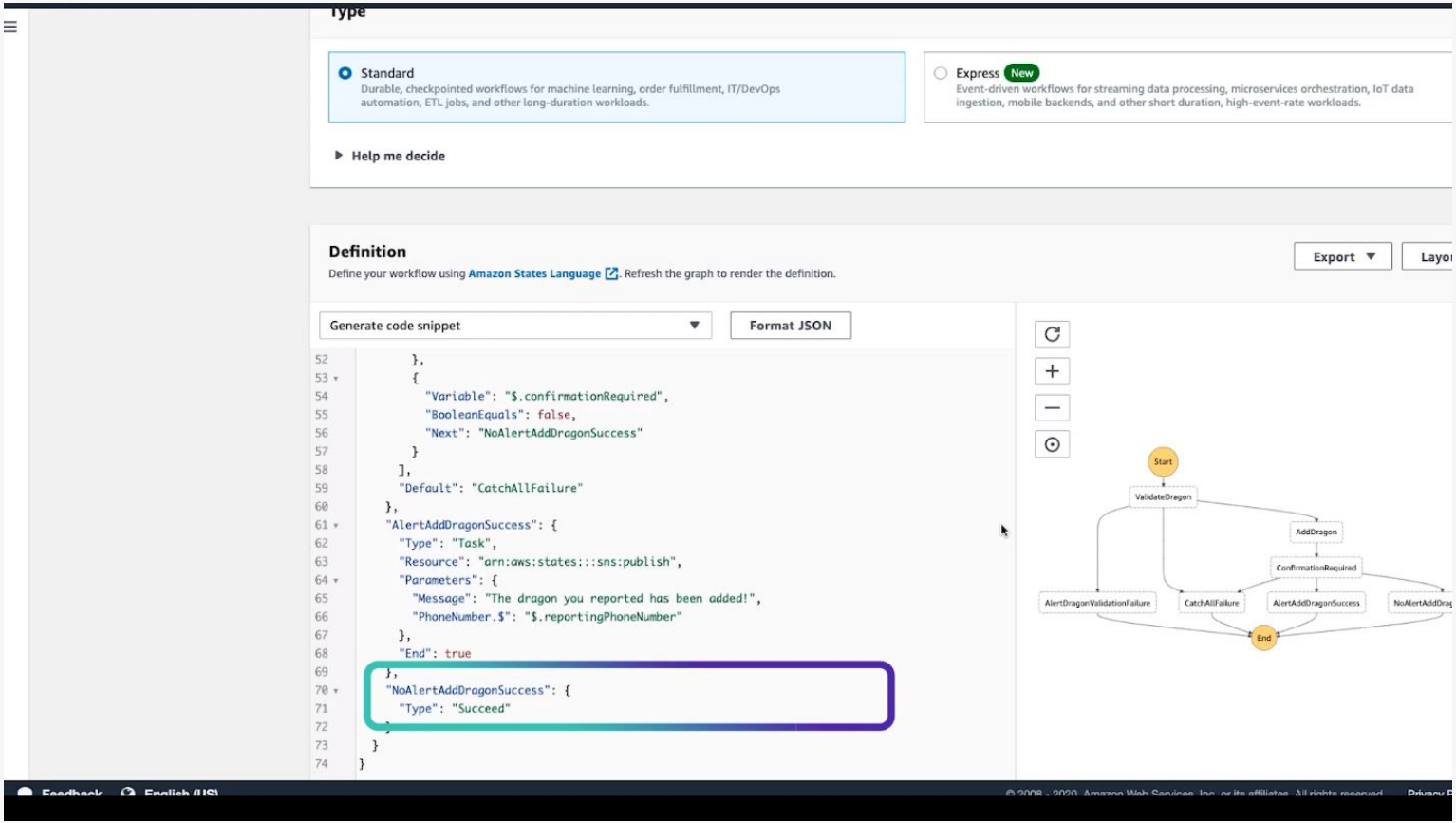
AddDragon state will invoke the associated lambda function



ConfirmationRequired state is a choice state.



next is the alert add dragon success which is another integration with SNS.

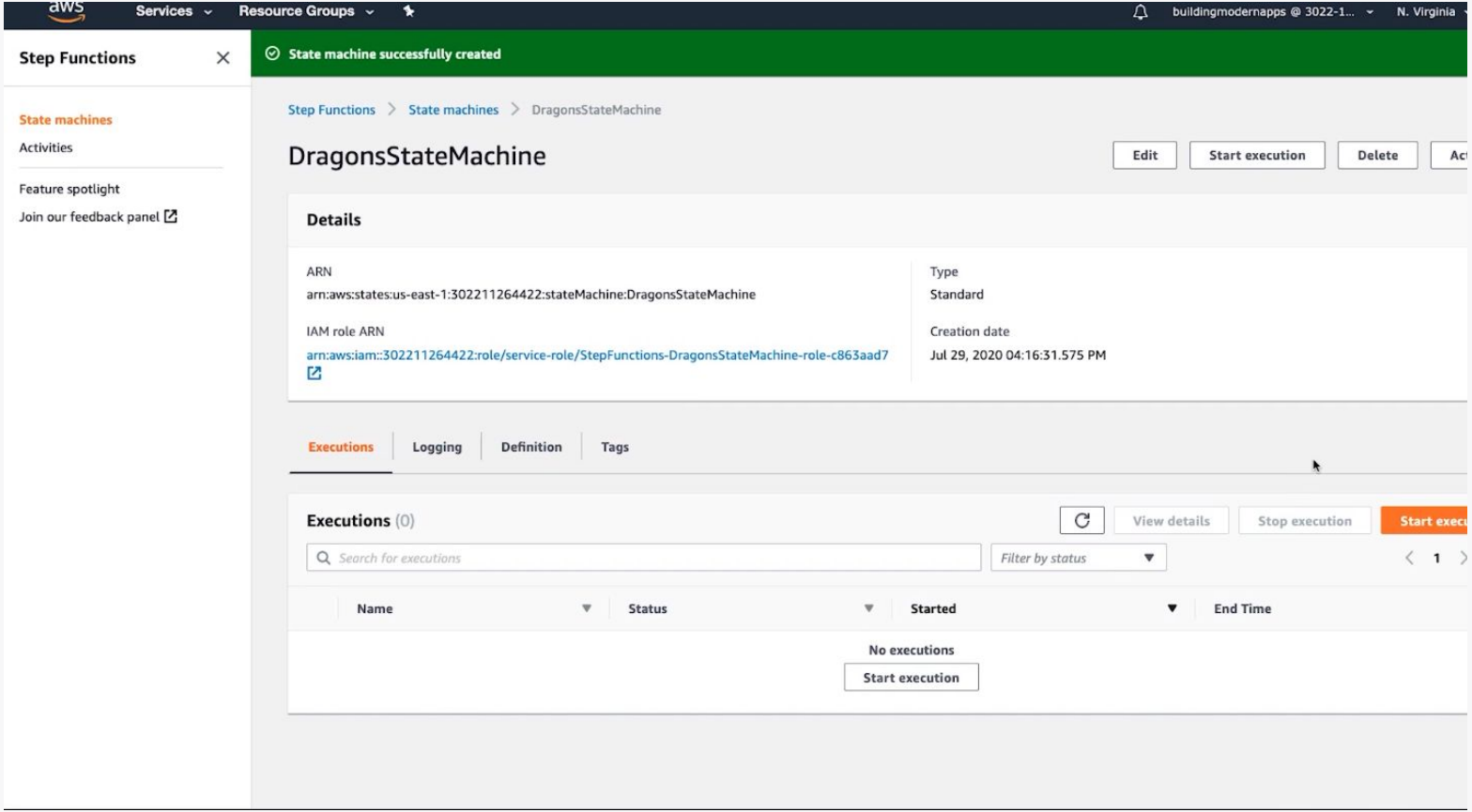


Finally, there's a NoAlertAddDragonSuccess which is a succeed state type.
It marks the execution as a success.

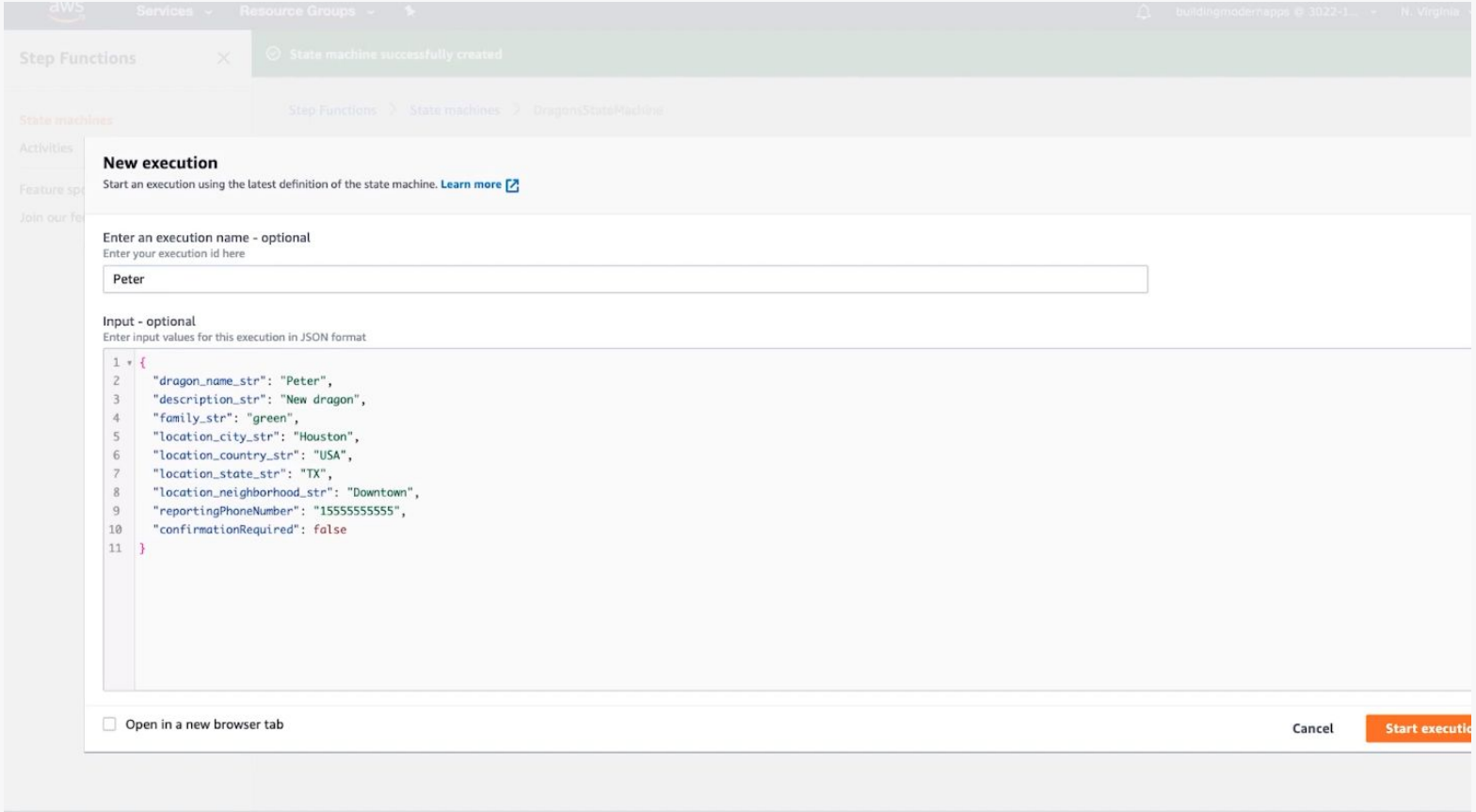
Now lets create an execution

```
{
  "dragon_name_str": "Peter",
  "description_str": "New dragon",
  "family_str": "green",
  "location_city_str": "Houston",
  "location_country_str": "USA",
  "location_state_str": "TX",
  "location_neighborhood_str": "Downtown",
  "reportingPhoneNumber": "15555555555",
  "confirmationRequired": false
}
```

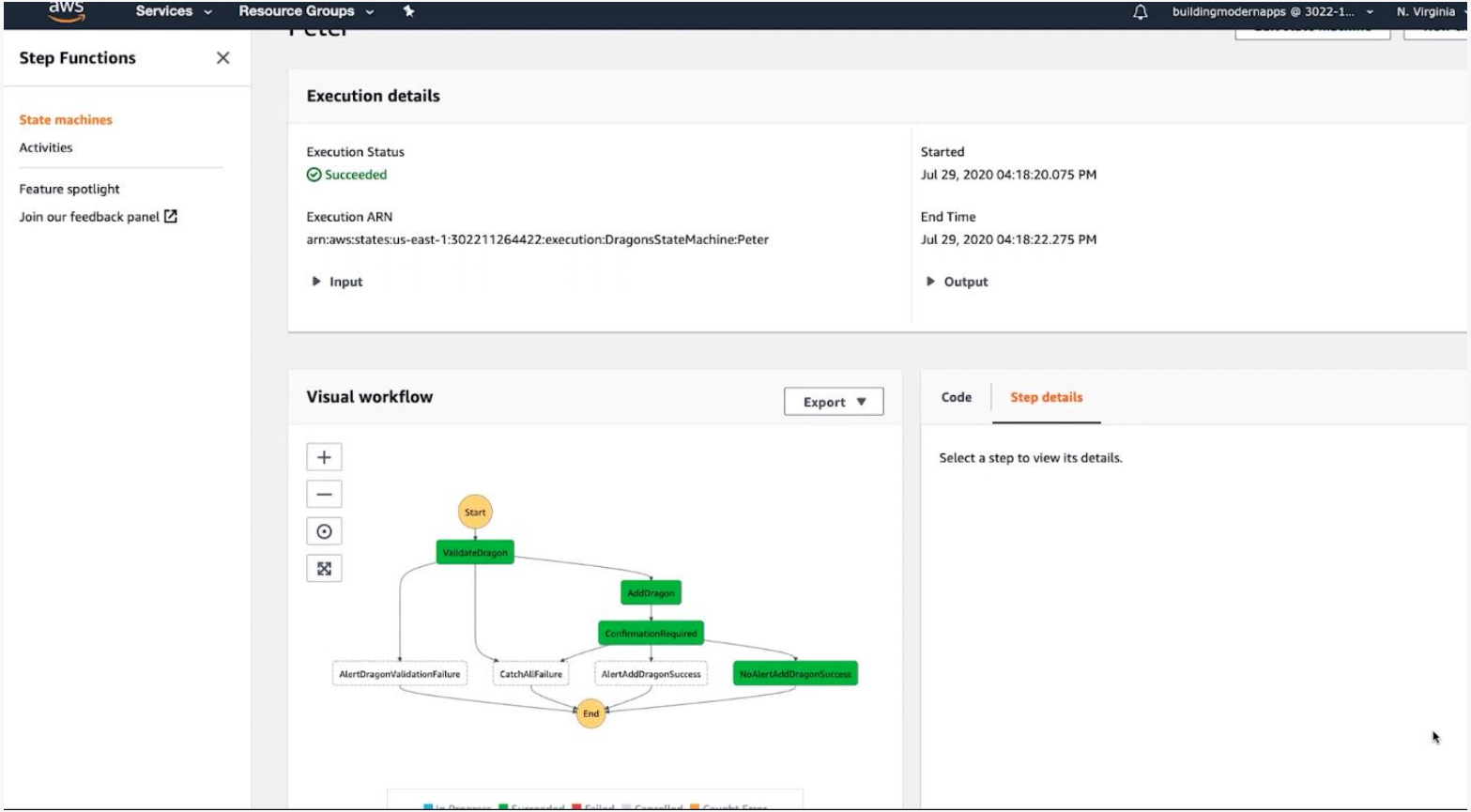

This is the payload that we will use to create executions.



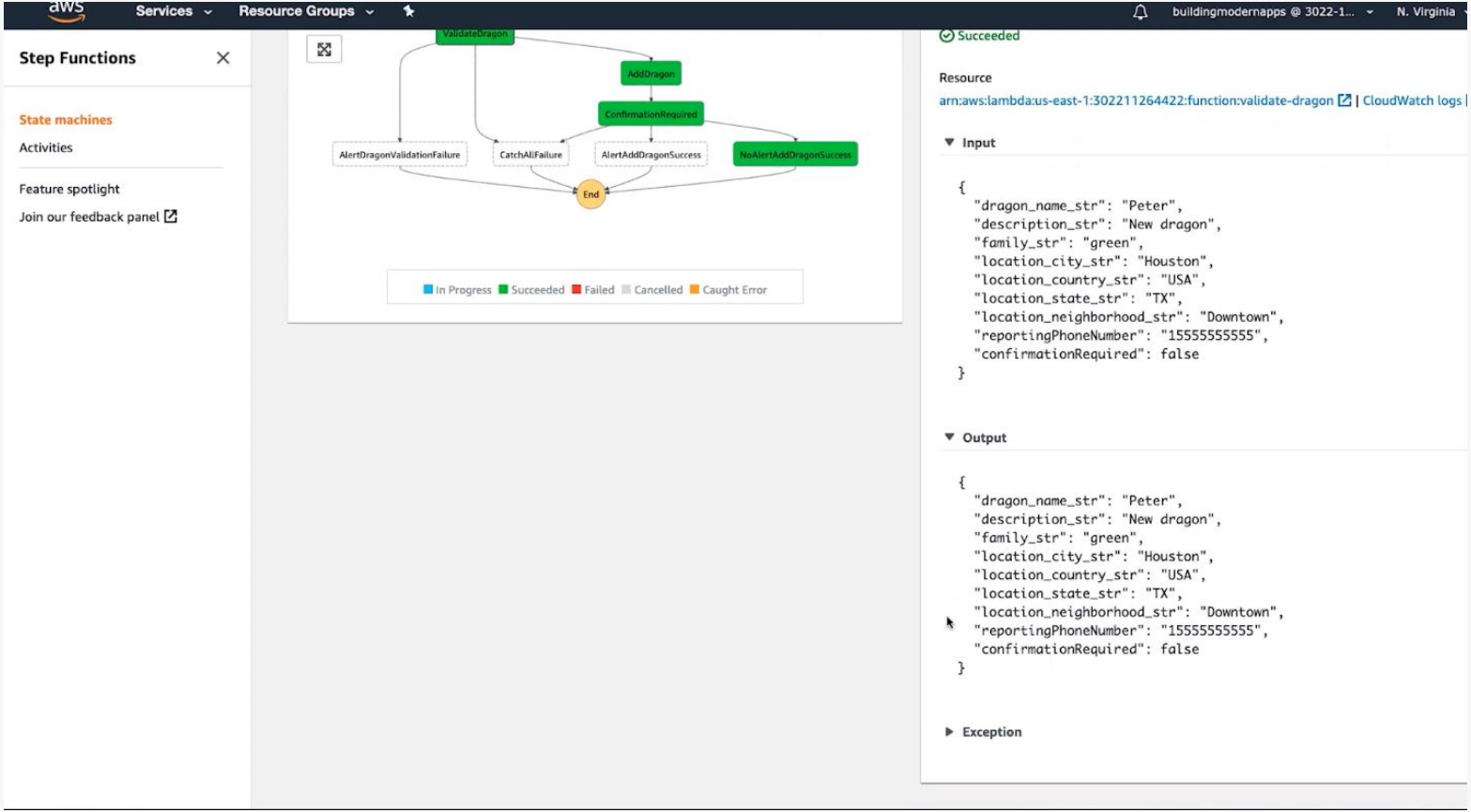
click on Start Execution



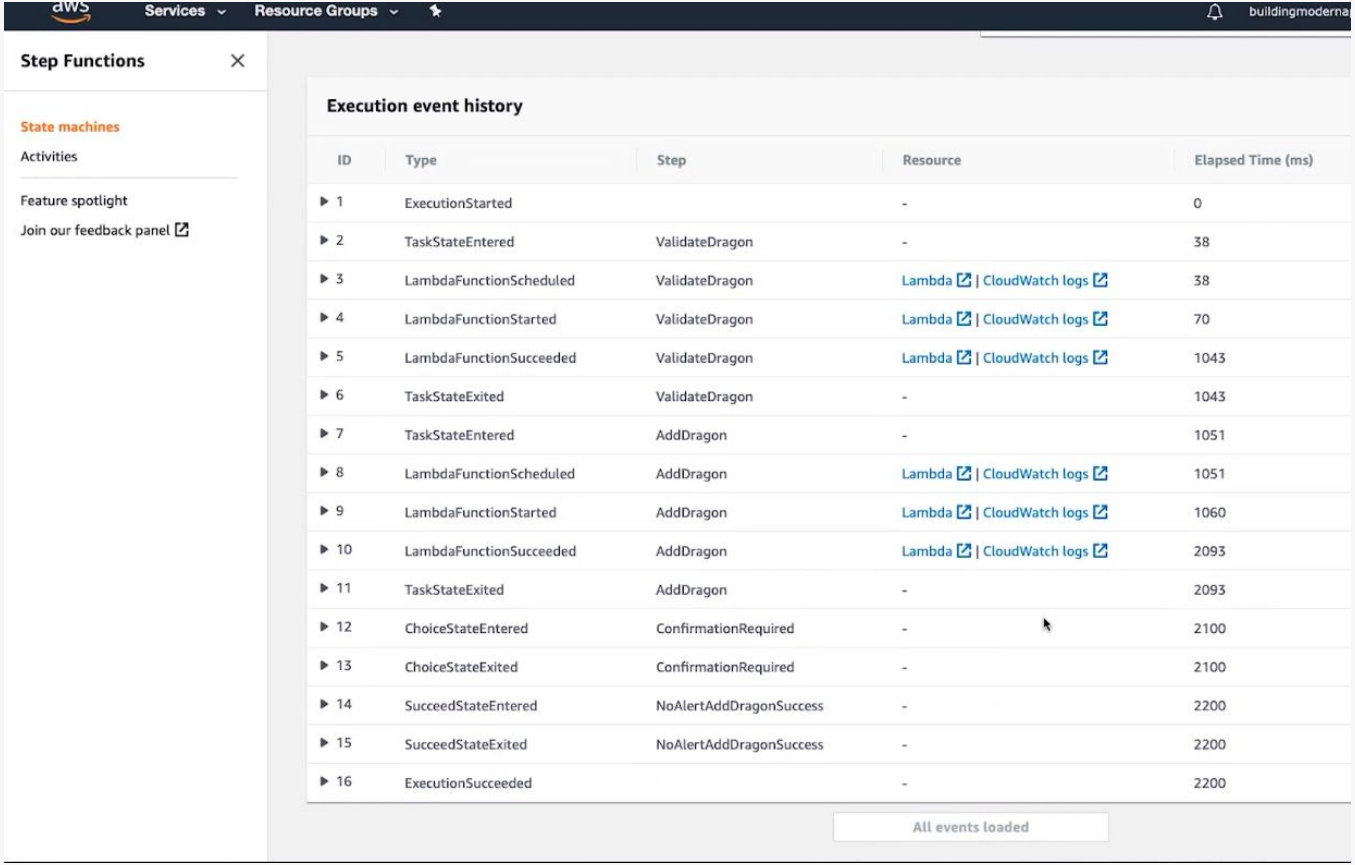
name it and paste the payload



This one was success and looking through, we can see the process path that was taken through the state machine.

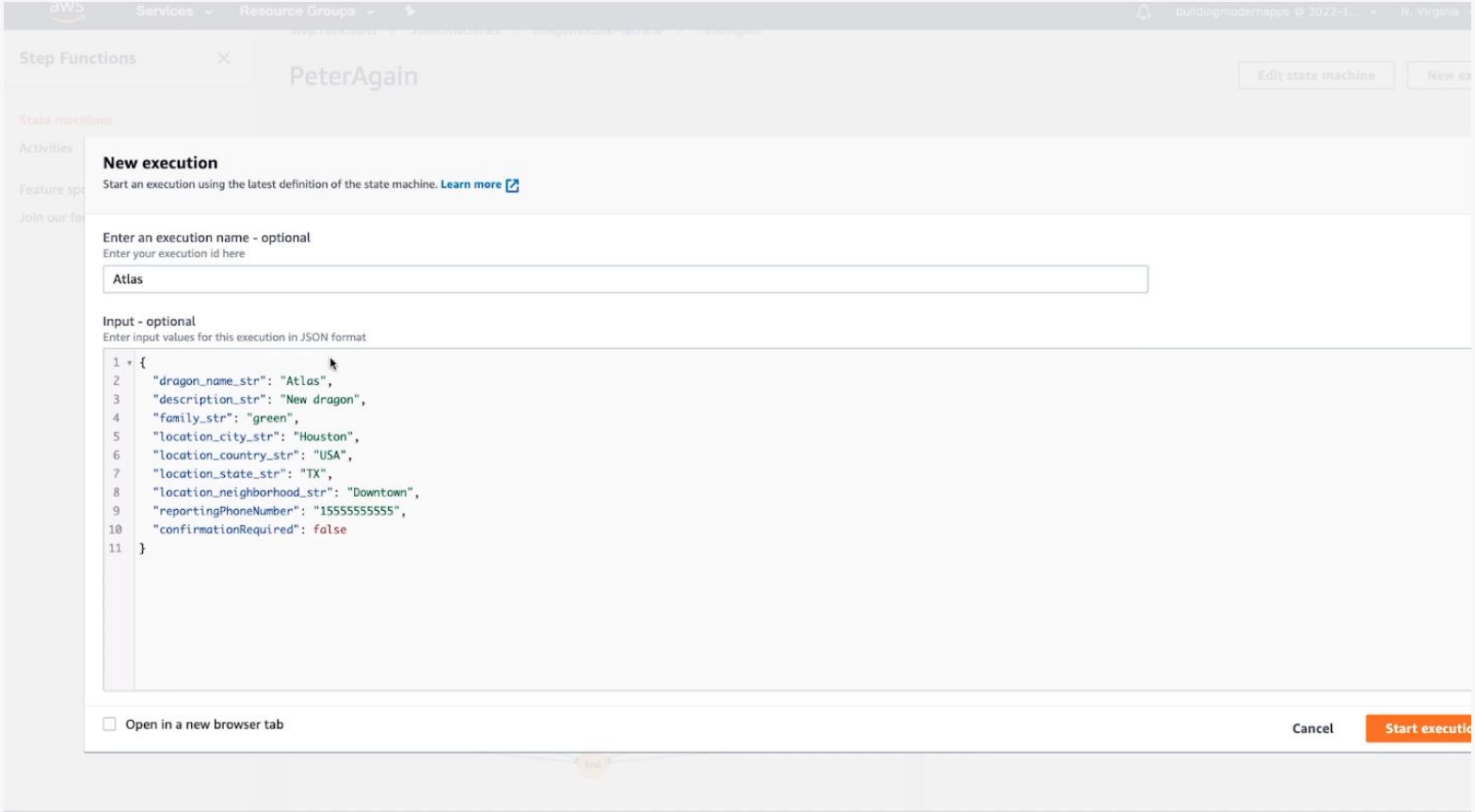


we can see the process path as well as the input and output of each of the states.

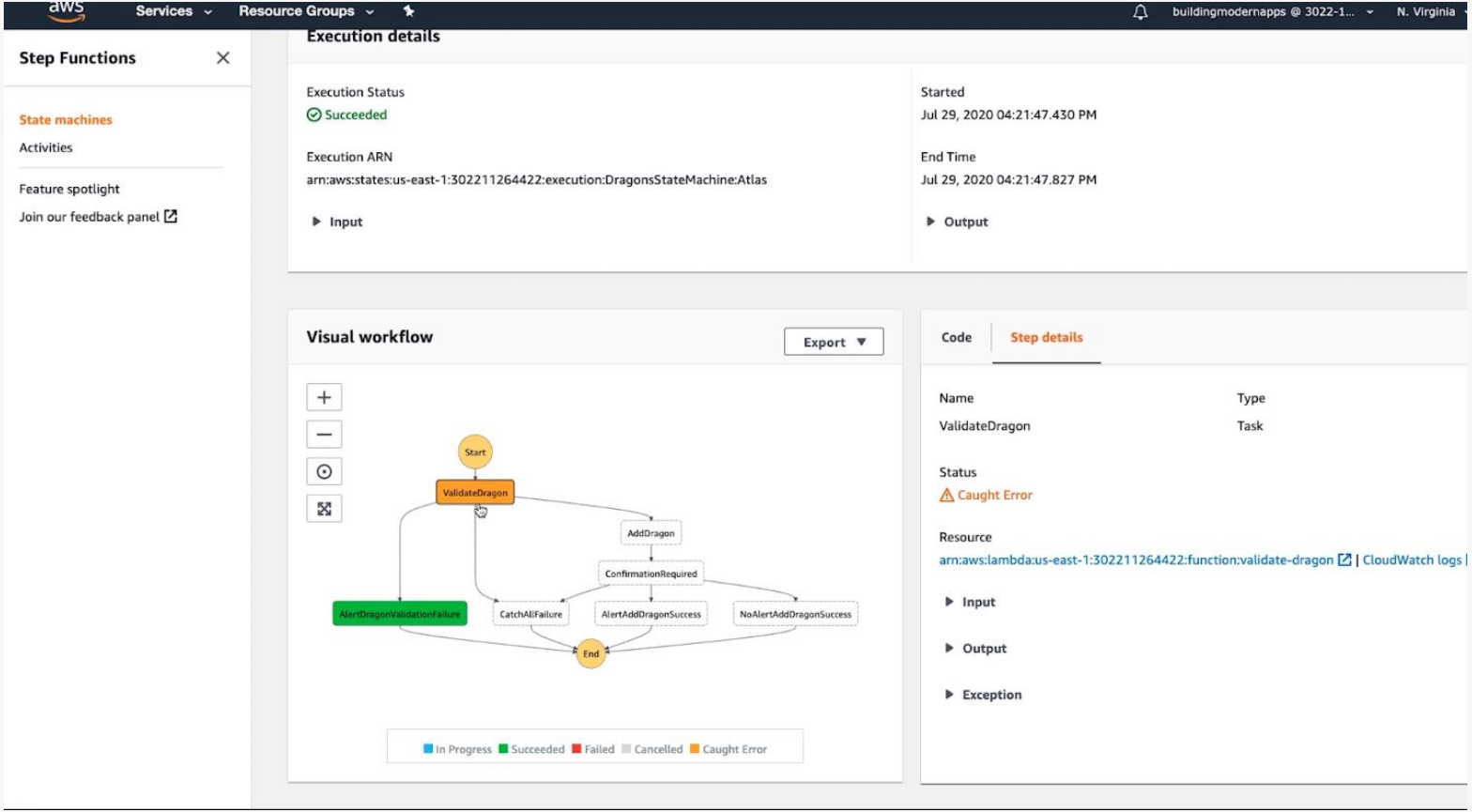


Scrolling down, you can also see the event history

Now let's a different path.



Add Atlas (already exist)



ValidateDragon step encountered an error, which changed the flow that was taken.

Good Resources:

Getting Started with AWS Step Functions

InputPath, ResultPath, and OutputPath Examples - AWS Step Functions

Sample projects for Step Functions

AWS Step Function Service Integrations

recap:

Let's say you have an input JSON data like this:

```
{ "input": { "name": "John Doe", "address": "123 Main Street" } }
```

And you want to pass the name and address as parameters to a Lambda function.

```
{
"StartAt": "MyState",
"States":
      { "MyState":
            { "Type": "Task",
            "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
            "Parameters":
                  { "Name.$": "$.input.name",
                    "Address.$": "$.input.address"
                  },
            "End": true

}}}
```

and you invoke an AWS Step Function with input data:

```
aws stepfunctions start-execution --state-machine-arn arn:aws:states:REGION:ACCOUNT_ID:stateMachine:STATE_MACHINE_NAME
--input '{"name": "John Doe", "address": "123 Main Street"}'
```

One feature of Step Functions is its ability to integrate with other AWS services without needing to write a Lambda function or a remote program to call AWS APIs.

In order to integrate with another AWS service from Step Functions,
all you need is a task state.

One common example is you can invoke a Lambda function from a state.

```
"Invoke Lambda function": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "Parameters": {
             "FunctionName":
             "arn:aws:lambda:<REGION>:<ACCTNUMBER>:function:<FUNCTIONNAME>",
             "Payload": {
                    "Input.$": "$"
                }
        },
        "Next": "NEXT_STATE"
 }
```

Here's a snippet showing a state that invokes a Lambda function.

```
"Invoke Lambda State": {
      "Type": "Task",
      "Resource":"arn:aws:lambda:<REGION>:<ACCTNUMBER>:function:<FUNCTIONNAME>"
      "Next": "NEXT_STATE"
 }
```

This is a bit of a simpler way to invoke a Lambda function from a state machine.

```
"Put item into DynamoDB": {
        "Type": "Task",
        "Resource": "arn:aws:states:::dynamodb:putItem",
        "Parameters": {
              "TableName": "MyDynamoDBTable",
              "Item": {
                      "DragonName": {
                            "S": "Atlas"
                      },
                      “Family” : {
                              “S”: “Red”
                      }
               }
        },
        "Next": "NEXT_STATE"
}
```

Integration with DynamoDB: a state that inserts an item into a DynamoDB table.
The parameters section is where you specify the parameters the AWS API needs.

```
"Amazon SNS: Publish a message": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
            "Message": "Hello from Step Functions!",
            "PhoneNumber": "+15555555555"
      },
      "Next": "NEXT_STATE"
}
```

integration with SNS

Step Functions integrates with a lot of other AWS services as well:

- Running an AWS batch job
- Running a task on Amazon elastic container service
- Sending a message to Amazon simple queue service
- Managing a job for AWS Glue or Amazon SageMaker
- Building workflows for executing an Amazon EMR job
- Running another step function workflow

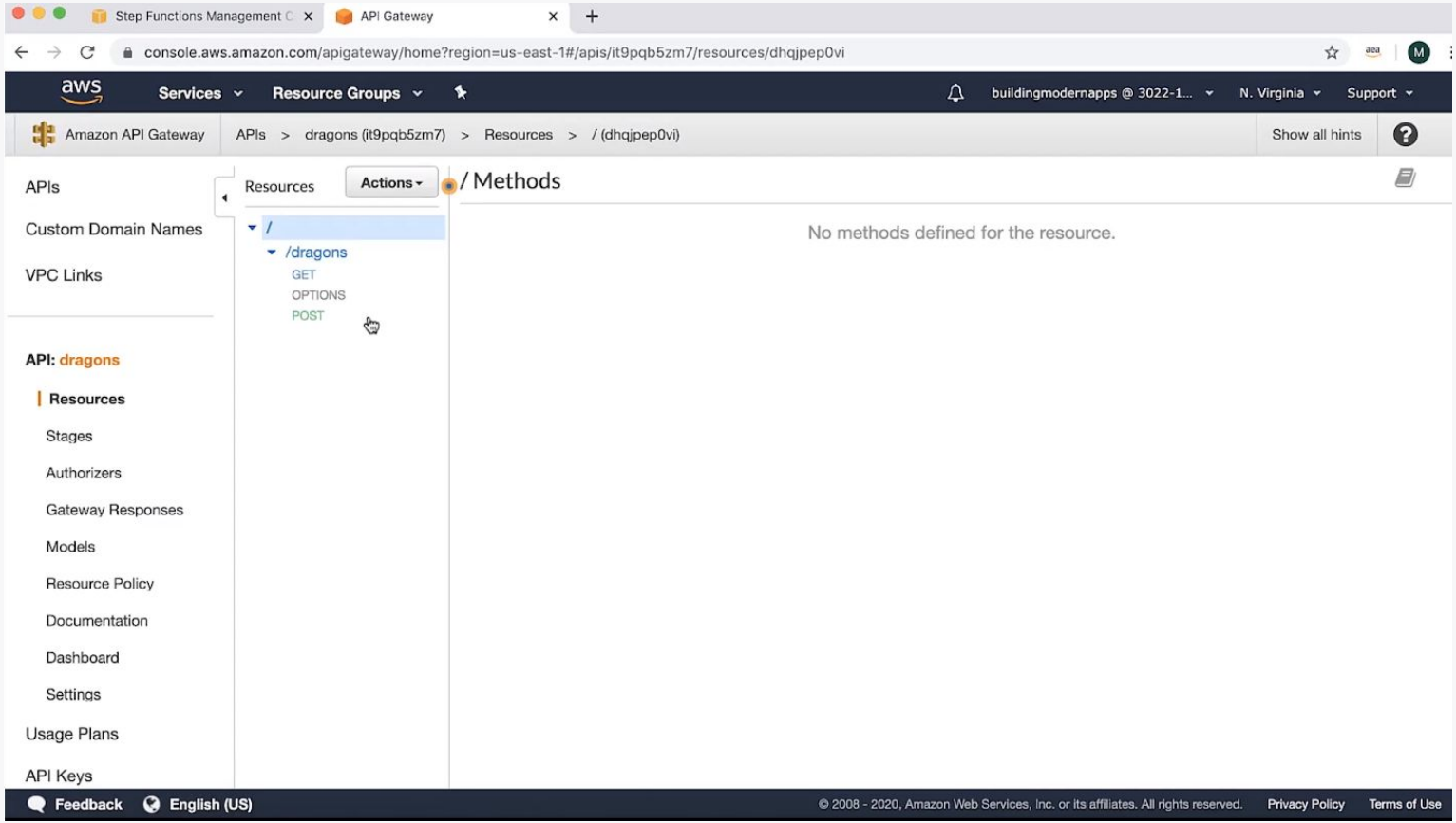
API Gateway and Step Function Integration Demo

Now that the state machine for reporting a new Dragon is created,
we want to configure API Gateway to start a new execution
every time a POST request comes into the dragon API.

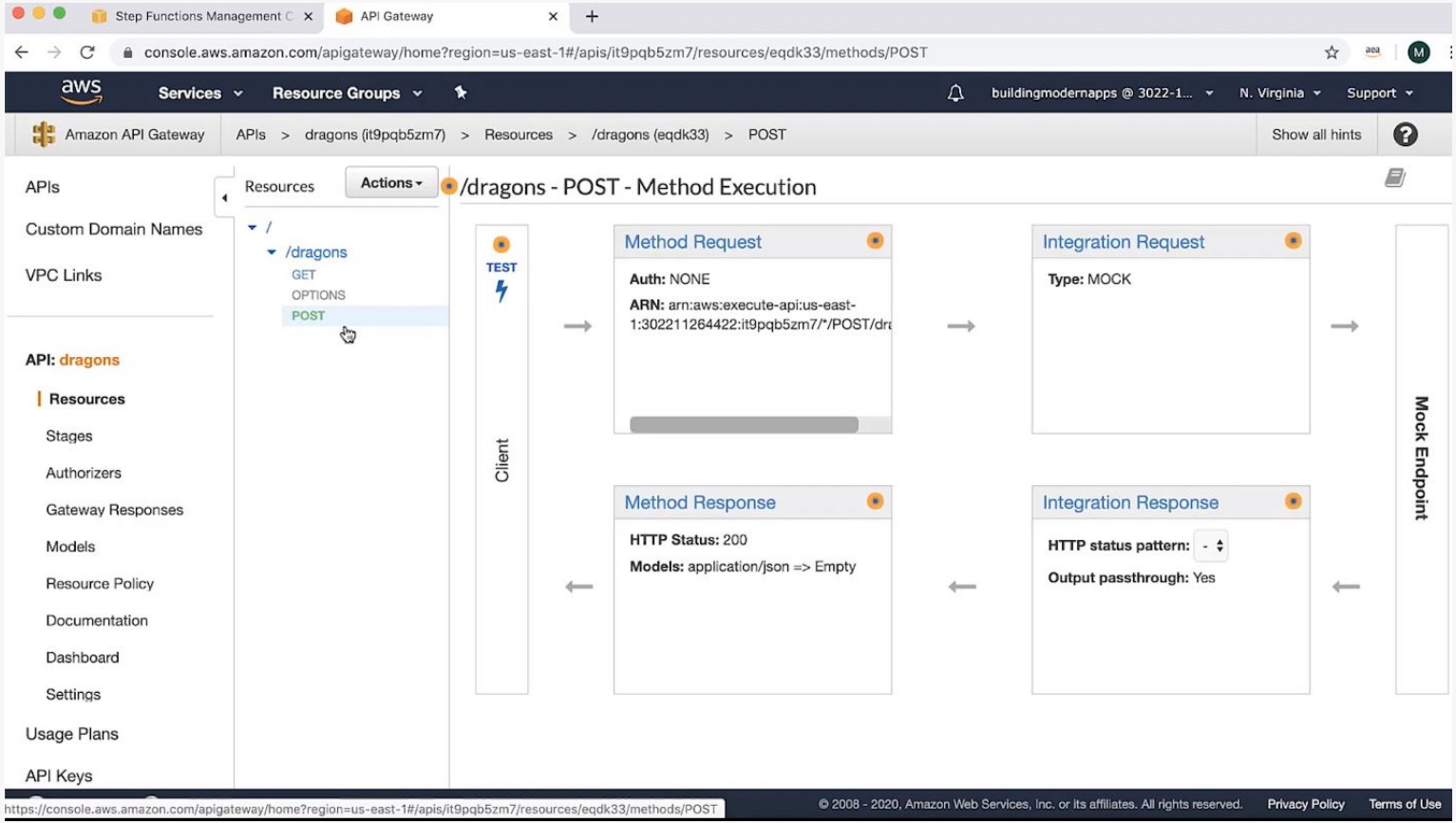
We use the API Gateway Integration Request Mapping
to change the format of the incoming data to match the format that Step Functions needs.

```
{
  "dragon_name_str": "Peter",
  "description_str": "New dragon",
  "family_str": "green",
  "location_city_str": "Houston",
  "location_country_str": "USA",
  "location_state_str": "TX",
  "location_neighborhood_str": "Downtown",
  "reportingPhoneNumber": "15555555555",
  "confirmationRequired": false
}
```

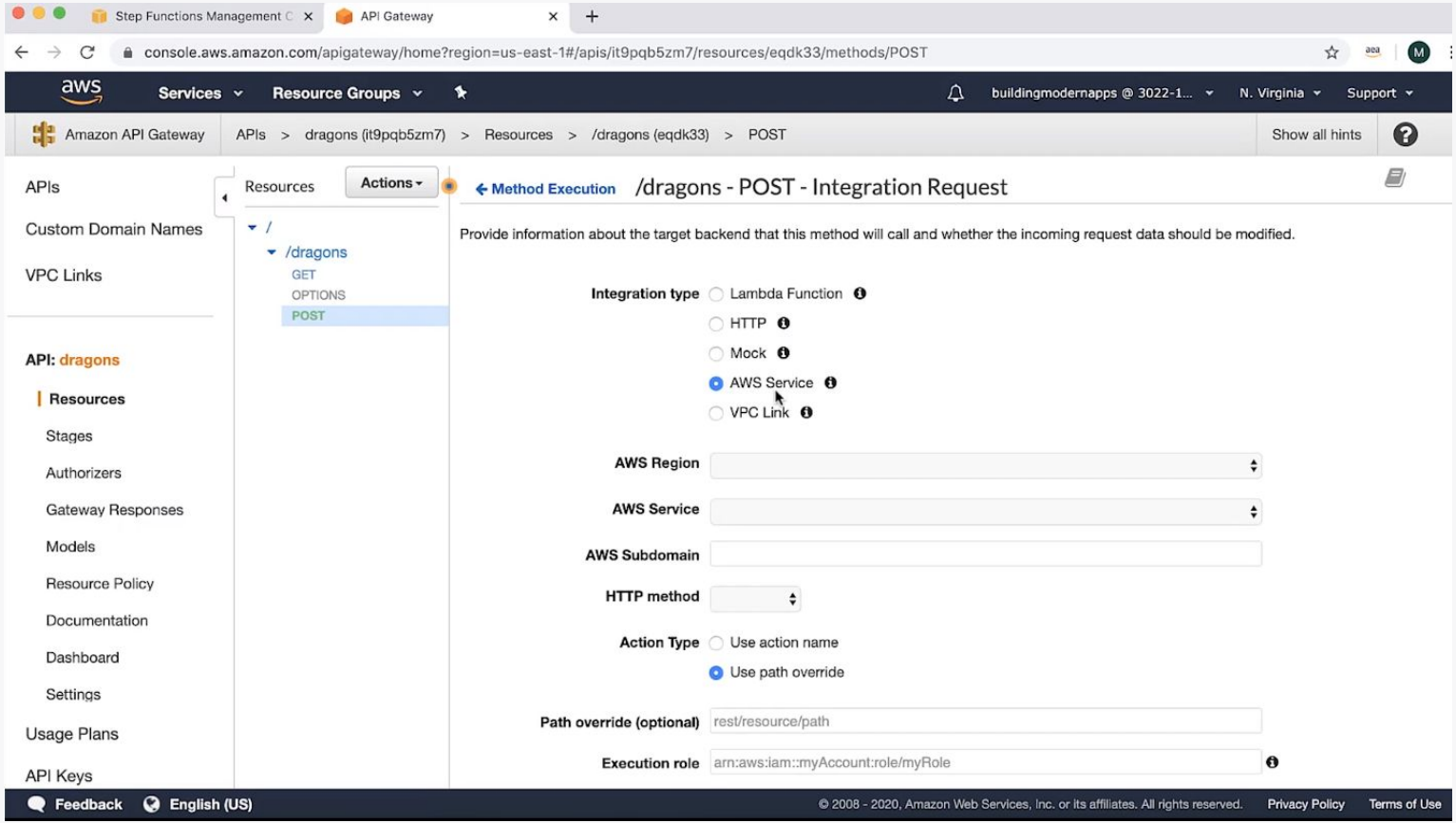

This is a valid new dragon data. This JSON payload should pass request validation because it matches the Dragon model.

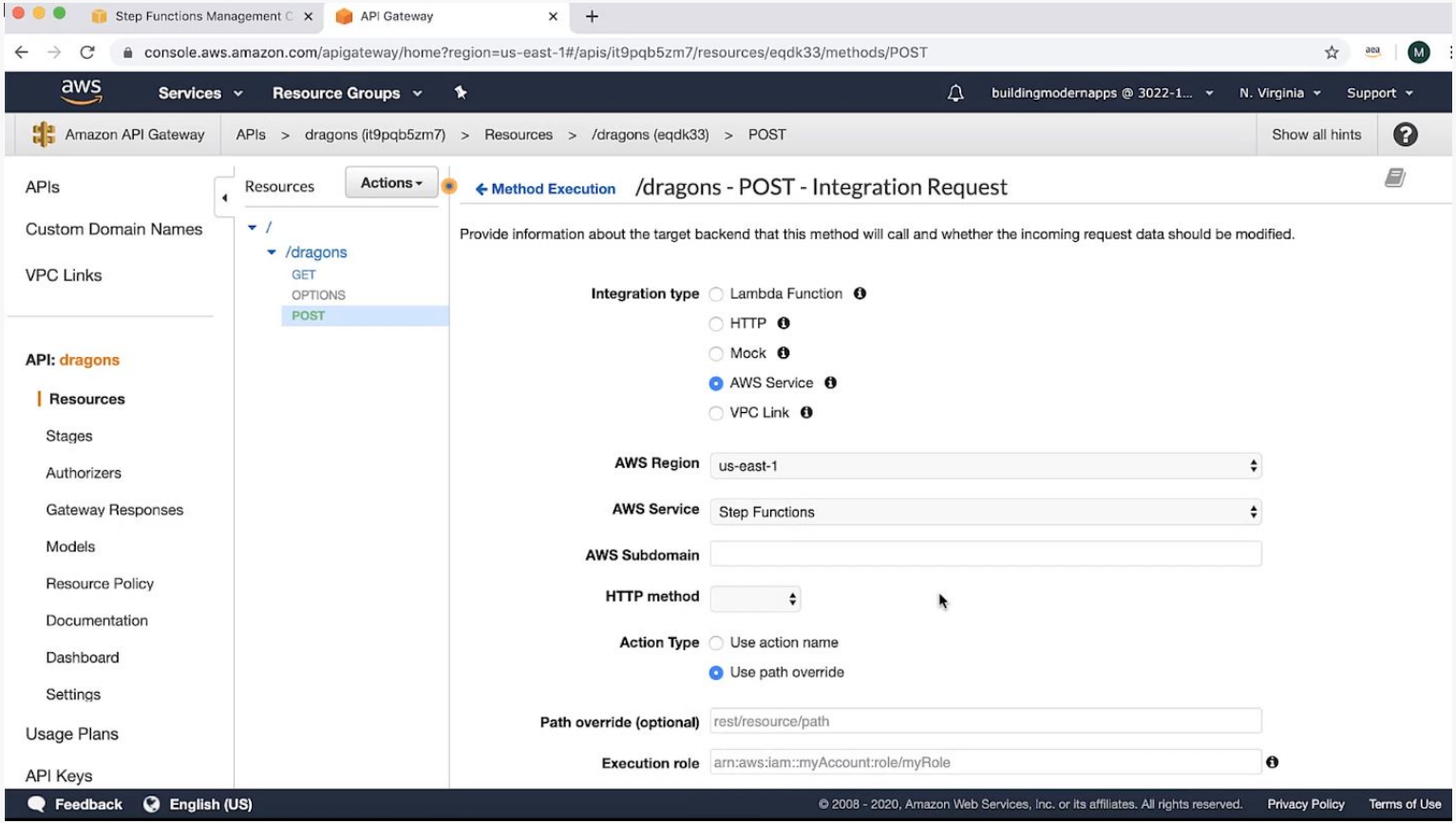


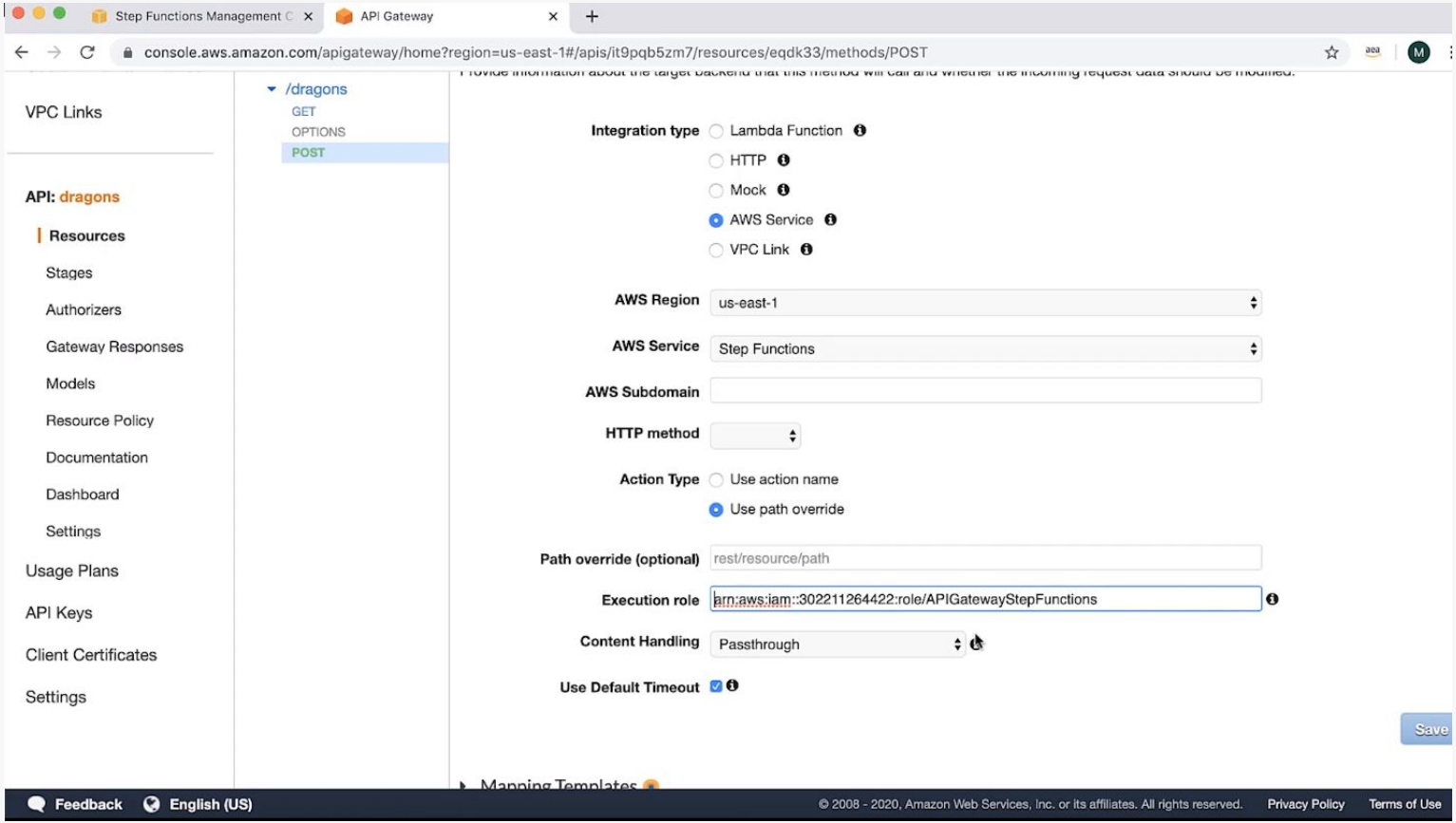
AWS API Console



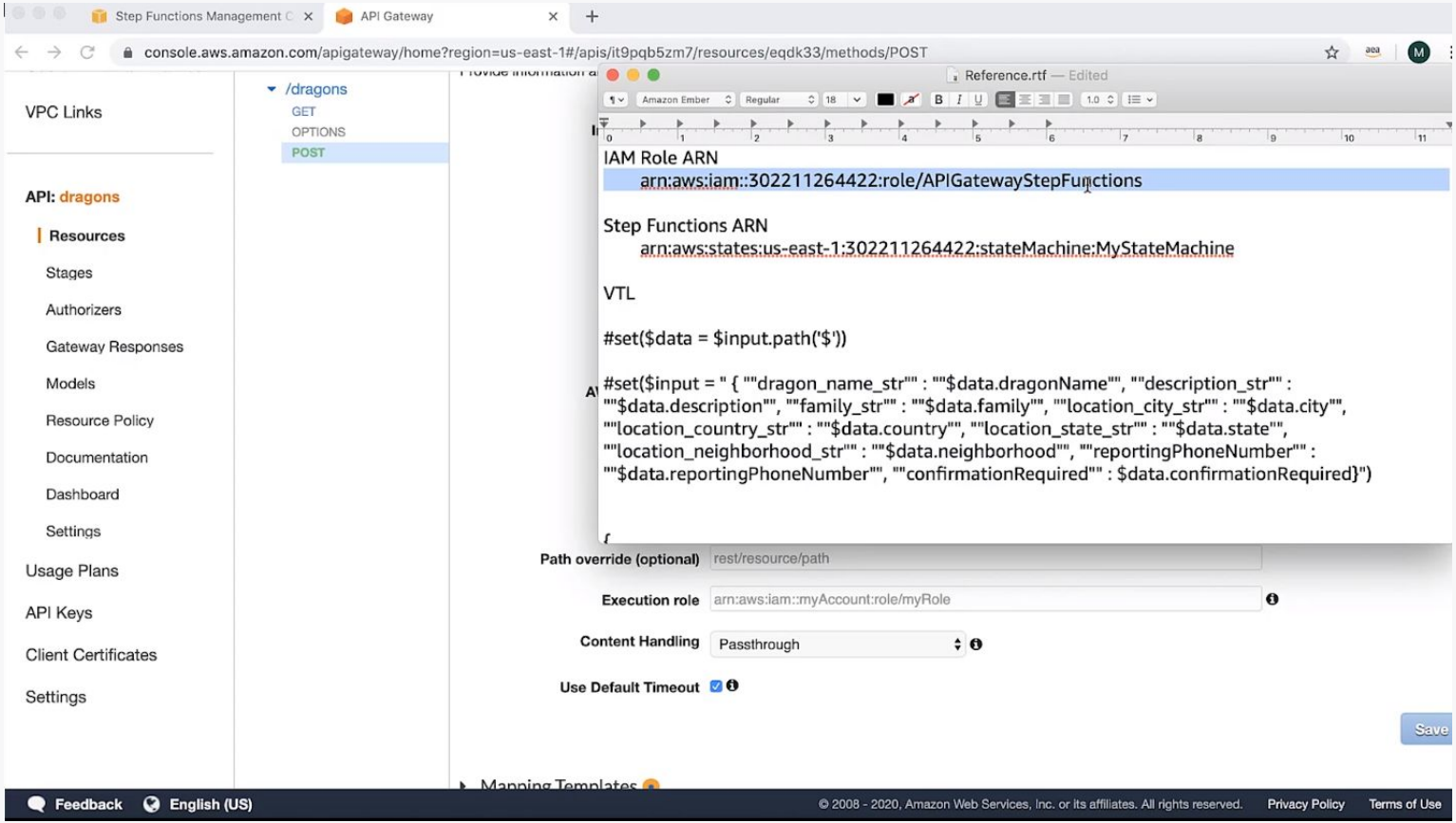
Post method. Navigate to the integration request.


Step Functions Management C
API Gateway
console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST
aws
Services
Resource Groups
buildingmodernapps @ 3022-1...
N. Virginia
Support
Amazon API Gateway
APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > POST
Show all hints
APIs
Custom Domain Names
VPC Links
API: dragons
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Resources
Actions
/
/dragons
GET
OPTIONS
POST
Method Execution
/dragons - POST - Integration Request
Provide information about the target backend that this method will call and whether the incoming request data should be modified.
Integration type
Lambda Function
HTTP
Mock
AWS Service
VPC Link
AWS Region
AWS Service
AWS Subdomain
HTTP method
Action Type
Use action name
Use path override
Path override (optional)
rest/resource/path
Execution role
arn:aws:iam::myAccount:role/myRole
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

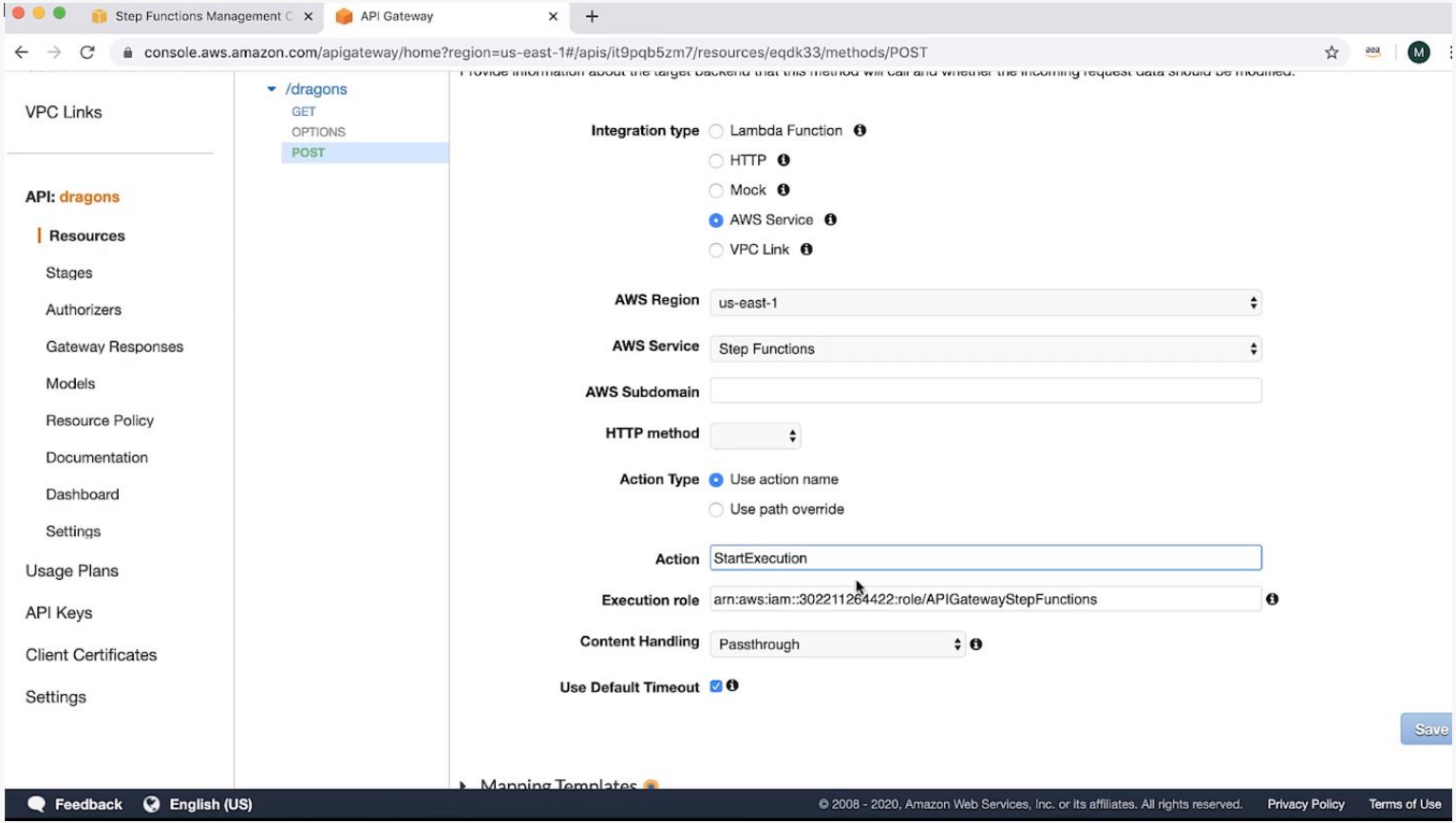

Step Functions Management C
API Gateway
console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST
aws
Services
Resource Groups
buildingmodernapps @ 3022-1...
N. Virginia
Support
Amazon API Gateway
APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > POST
Show all hints
APIs
Custom Domain Names
VPC Links
API: dragons
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Resources
Actions
/
/dragons
GET
OPTIONS
POST
Method Execution
/dragons - POST - Integration Request
Provide information about the target backend that this method will call and whether the incoming request data should be modified.
Integration type
Lambda Function
HTTP
Mock
AWS Service
VPC Link
AWS Region
us-east-1
AWS Service
Step Functions
AWS Subdomain
HTTP method
Action Type
Use action name
Use path override
Path override (optional)
rest/resource/path
Execution role
arn:aws:iam::myAccount:role/myRole
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use



This execution role is what API Gateway is going to assume in order to invoke the state machine. That means that we have to already have an IAM Role created in this account that has the necessary permissions that API Gateway will need to execute the state machine or start a new execution.



Keep all this information in a file



StartExecution is the AWS API for starting a new execution.

Notice that we didn’t insert ARN for the state machine.

So looking at this, how a customer send in a POST request,
and then invoke that specific state machine?

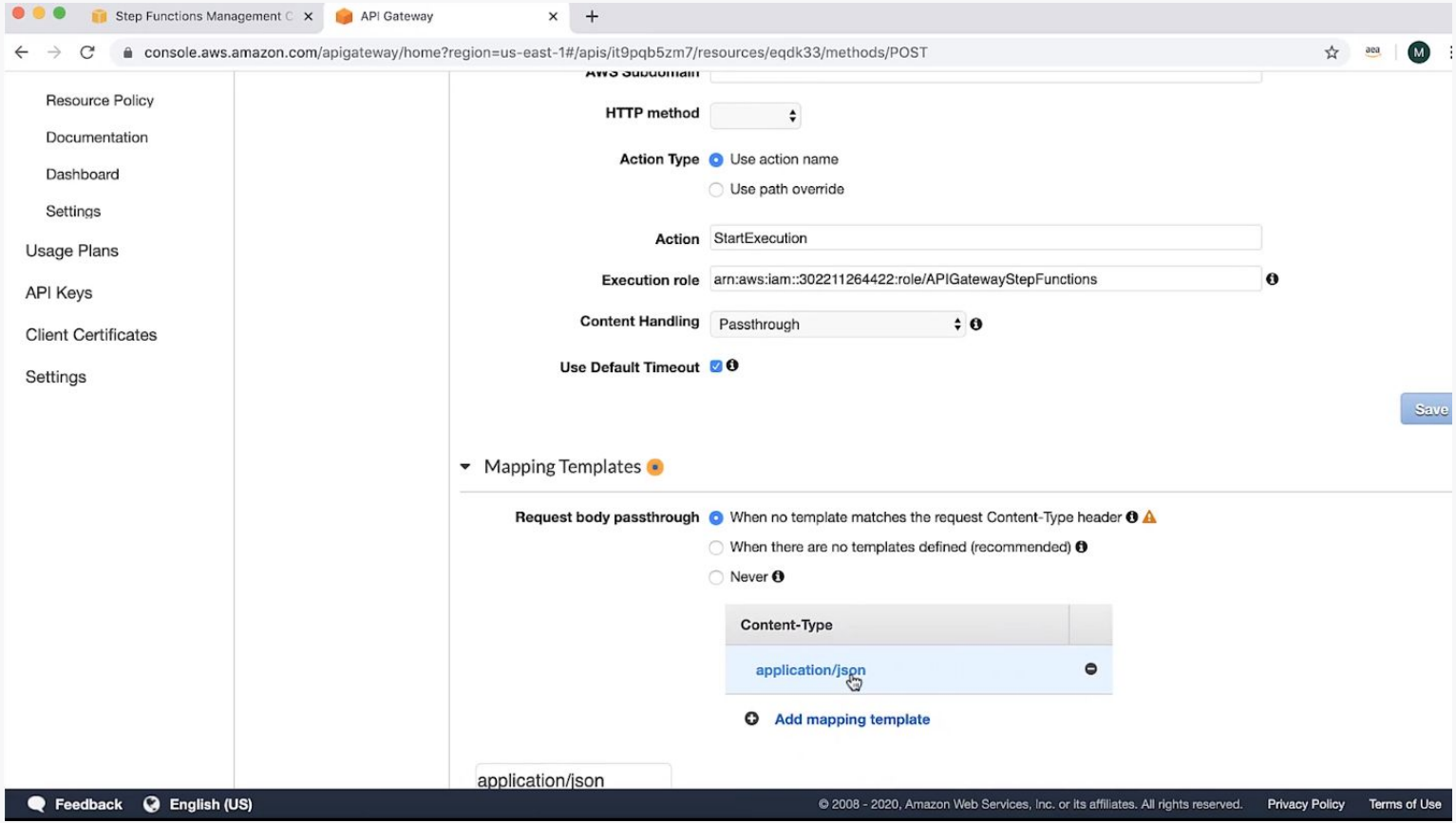
One way is that Client have to include the ARN for that state machine on the request.

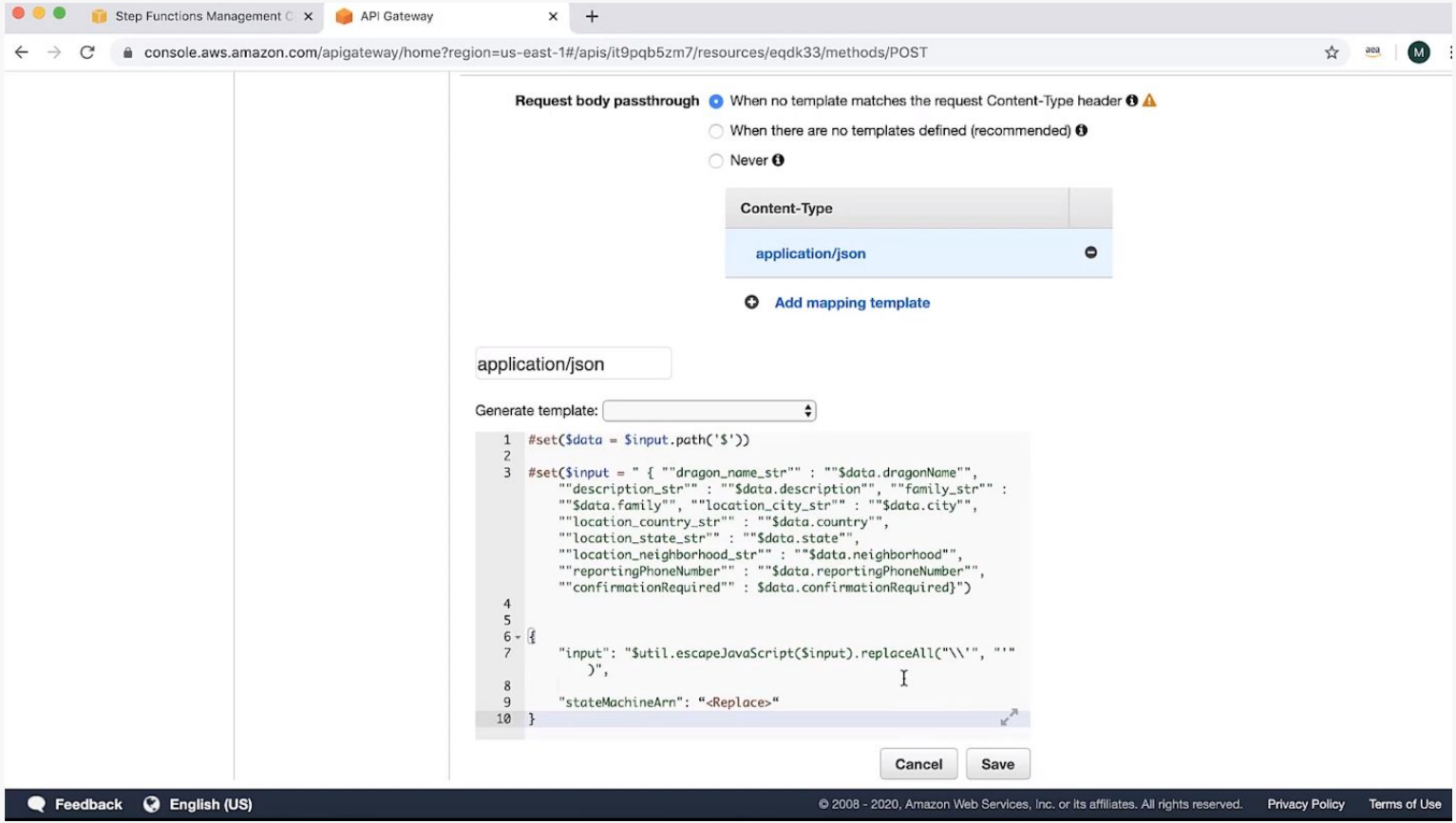
But, it's not a good idea to expose that ARN to clients.

Instead, we need to find a way to inject that ARN
into this payload without them having to provide it.

We can take advantage of the mapping templates.

Mapping templates are written in VTL and they allow you to change the payload.


Step Functions Management C
API Gateway
console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST
Resource Policy
Documentation
Dashboard
Settings
Usage Plans
API Keys
Client Certificates
Settings
HTTP method
Action Type
Use action name
Use path override
Action
StartExecution
Execution role
arn:aws:iam::302211264422:role/APIGatewayStepFunctions
Content Handling
Passthrough
Use Default Timeout
Save
Mapping Templates
Request body passthrough
When no template matches the request Content-Type header
When there are no templates defined (recommended)
Never
Content-Type
application/json
Add mapping template
application/json
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use



This is a mapping VTL which also add state machine ARN to the end of request.

Step Functions Management C | API Gateway

console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST

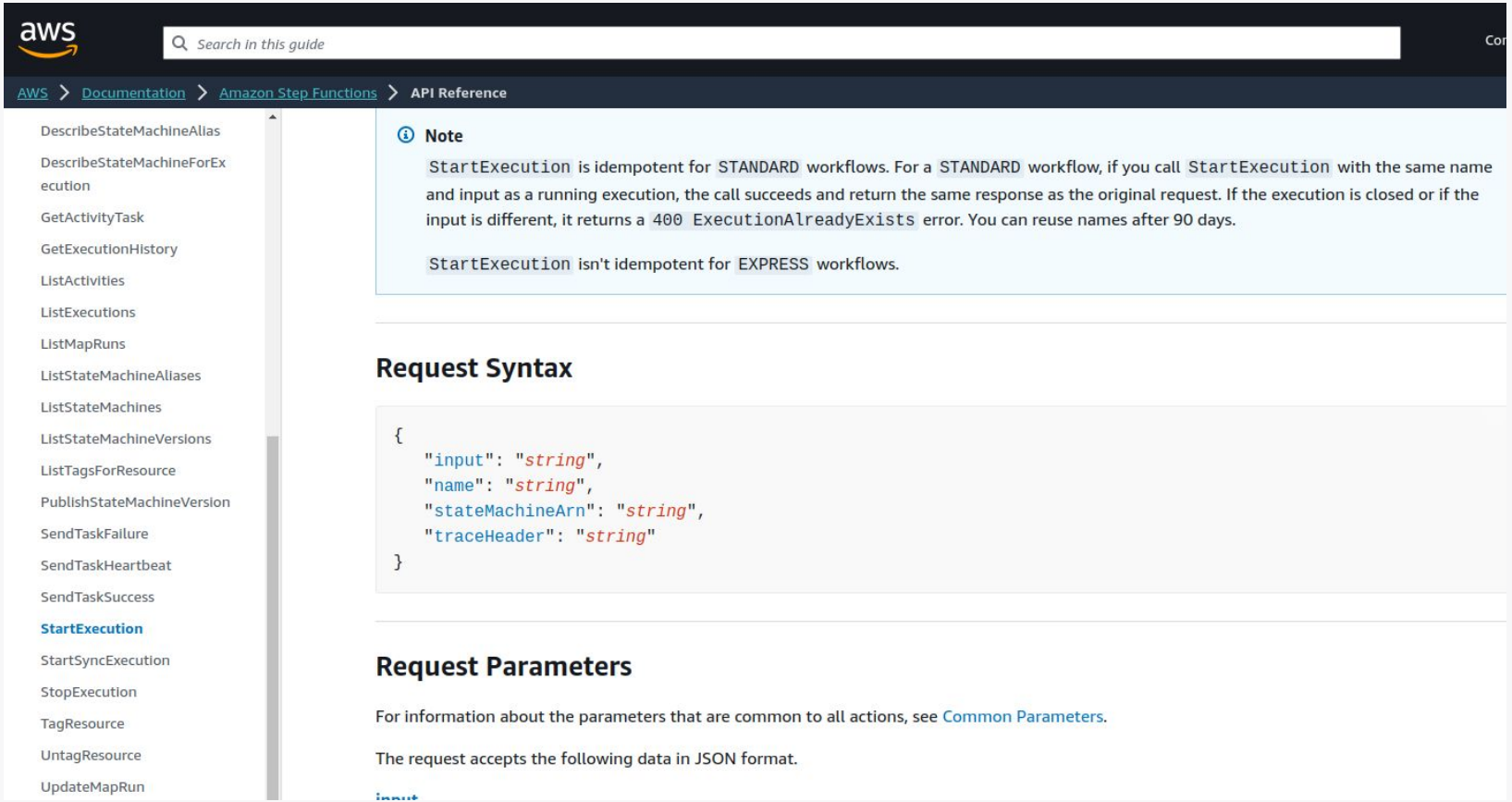
Generate template:

```
#set($data = $input.path('$'))

#set($input = " { ""dragon_name_str"" : ""$data.dragonName"", ""description_str"" : ""$data.description"", ""family_str"" : ""$data.family"", ""location_city_str"" : ""$data.city""
    ""location_country_str"" : ""$data.country"", ""location_state_str"" : ""$data.state"", ""location_neighborhood_str"" : ""$data.neighborhood"", ""reportingPhoneNumber"" : ""$d
    .reportingPhoneNumber"", ""confirmationRequired"" : $data.confirmationRequired}")


{
    "input": "$util.escapeJavaScript($input).replaceAll("\\'", "'")",

    "stateMachineArn": “<Replace>“
}
```



this is the StartExecution request syntax. We only used input and stateMachineArn in this example

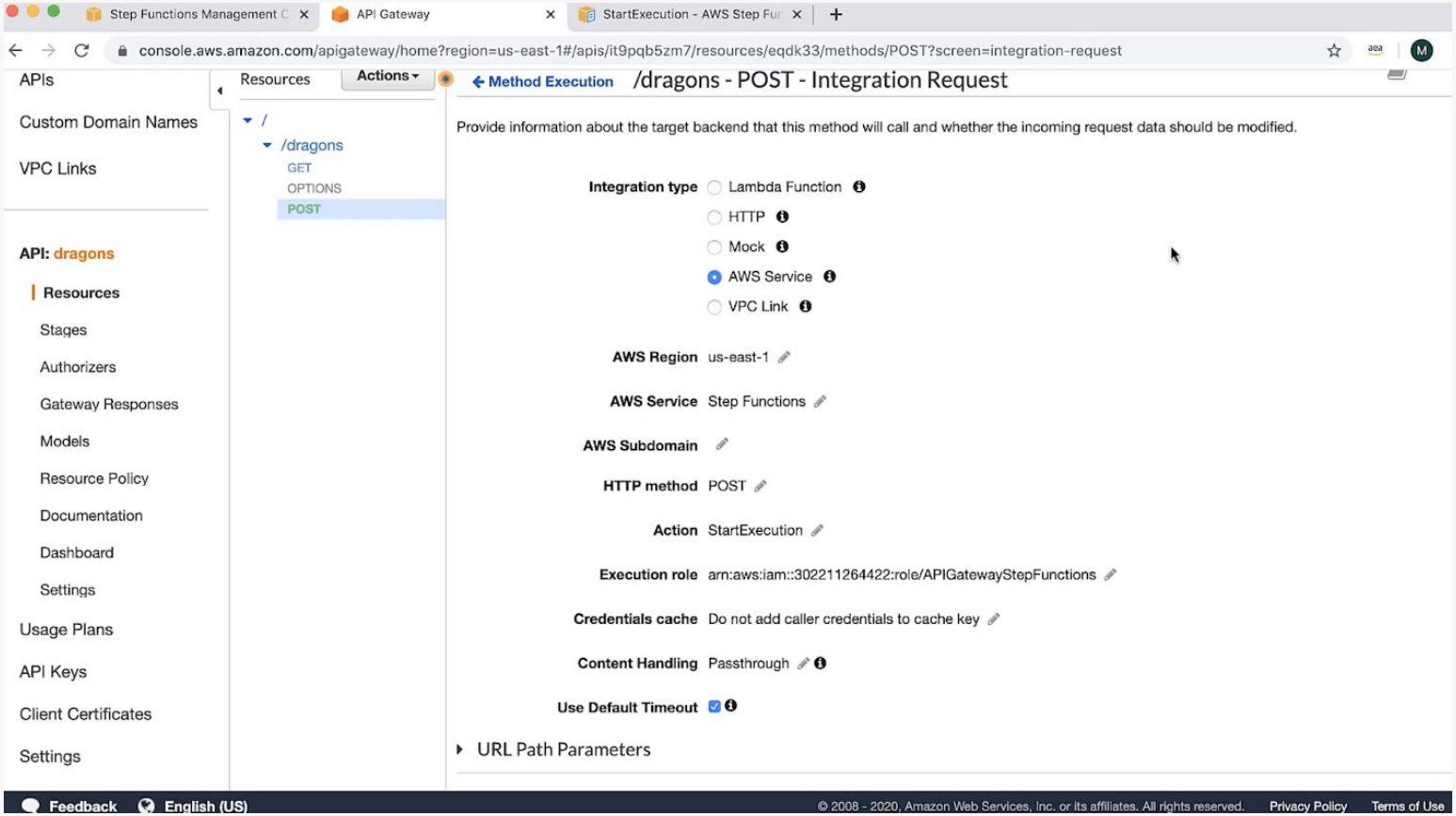
Client Format

```
{
  "dragonName": "Henry",
  "description": "New dragon",
  "family": "green",
  "city": "Seattle",
  "country": "USA",
  "state": "WA",
  "neighborhood": "Downtown",
  "reportingPhoneNumber": "5555555555",
  "confirmationRequired": true
}
```

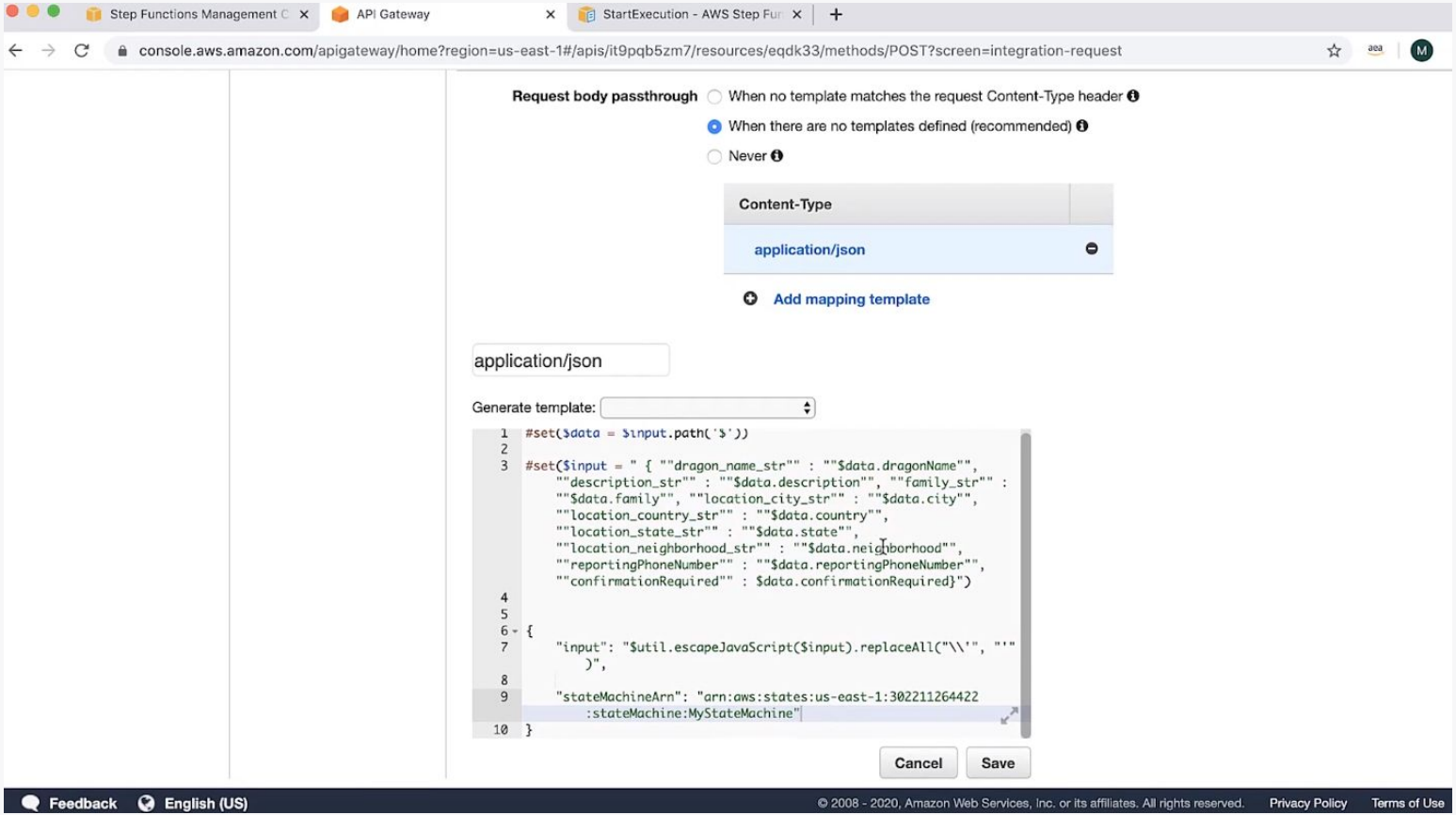
Backend Format

```
{
  "description_str": "New Dragon",
  "dragon_name_str": "Henry",
  "family_str": "green",
  "location_city_str": "Seattle",
  "location_country_str": "USA",
  "location_neighborhood_str":
  "Downtown",
  "location_state_str": "WA"
}
```

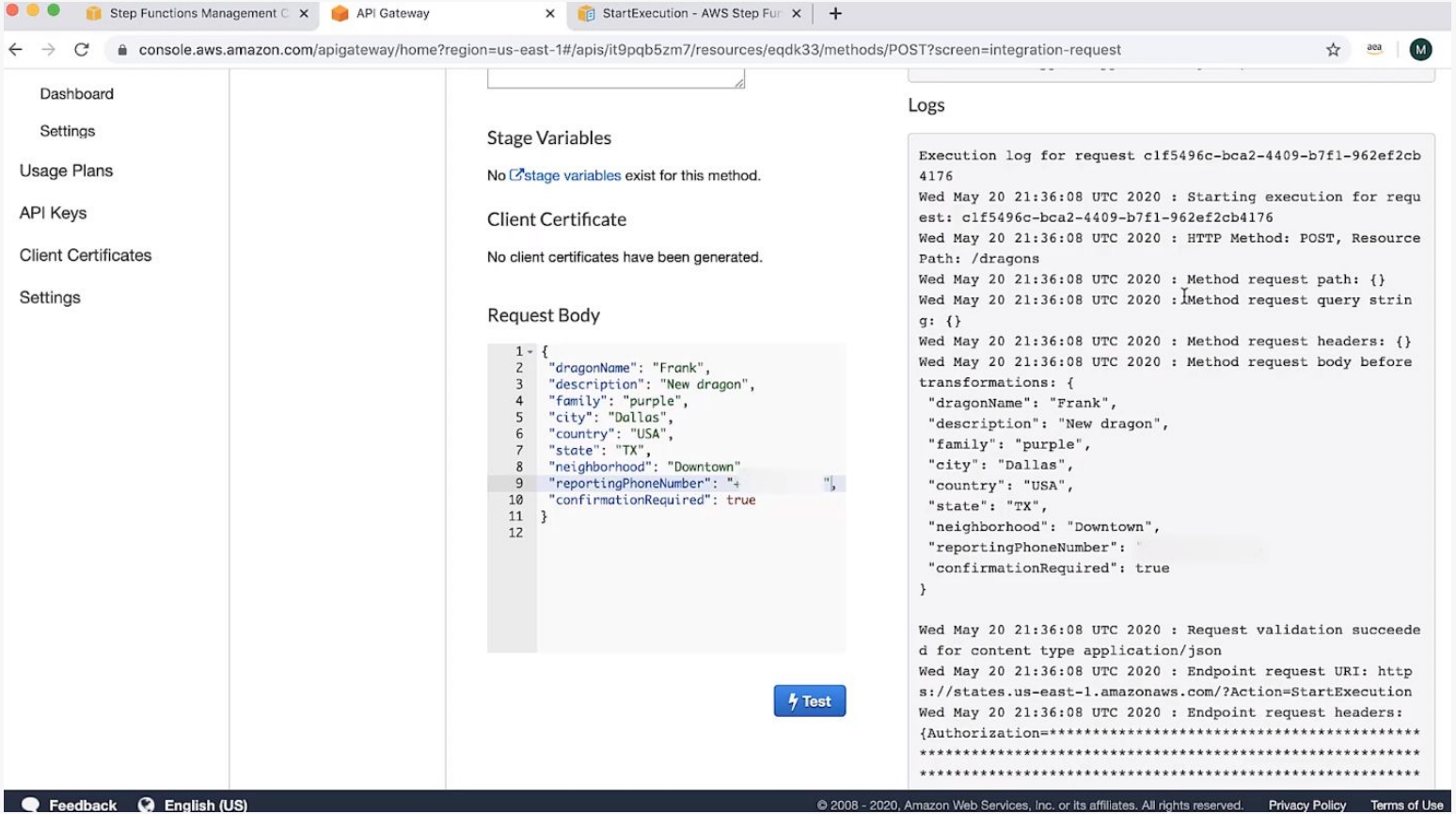
This is what client send, and what the API mapping does with input to send to the backend.



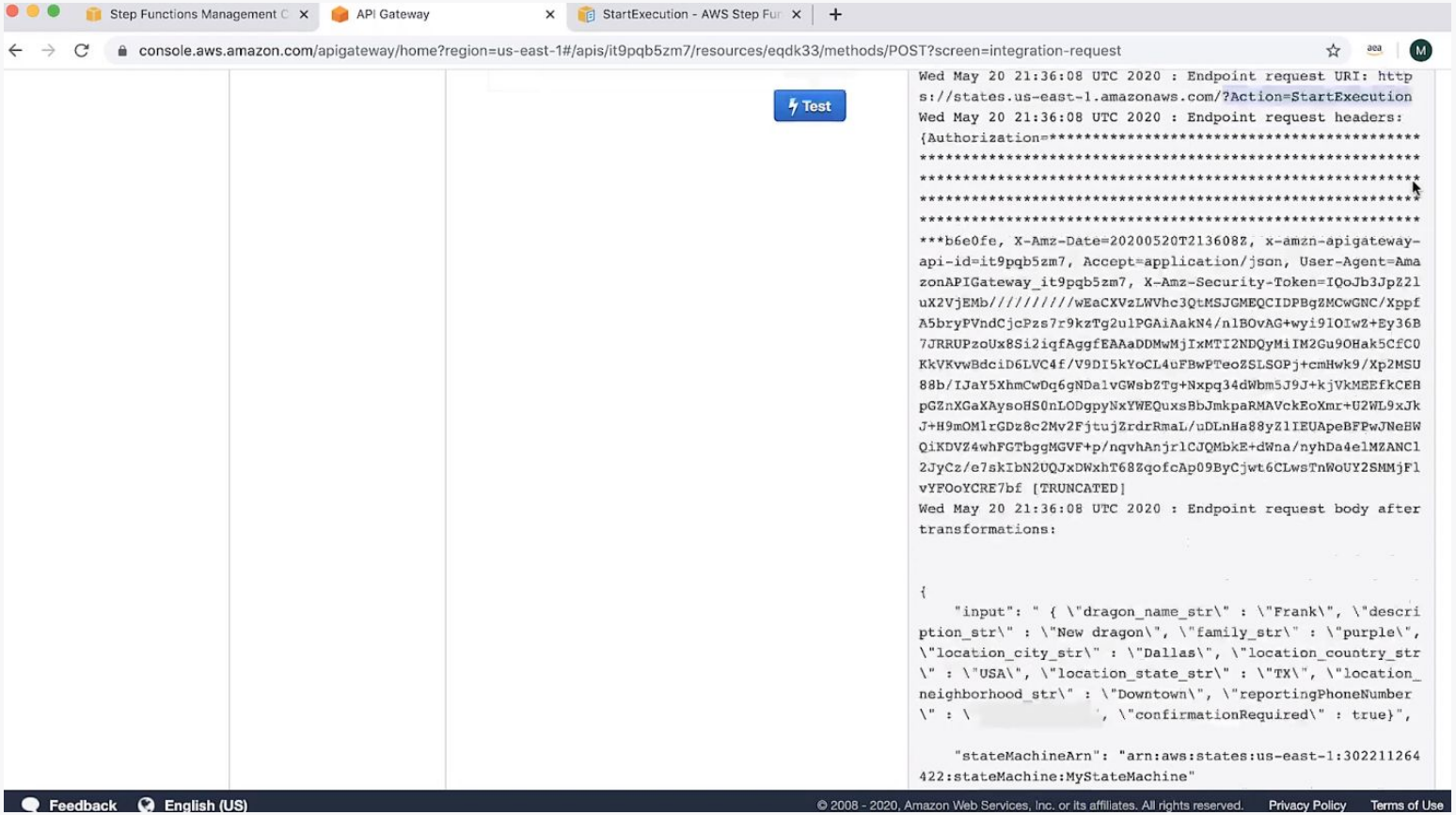
Everything is ready to test



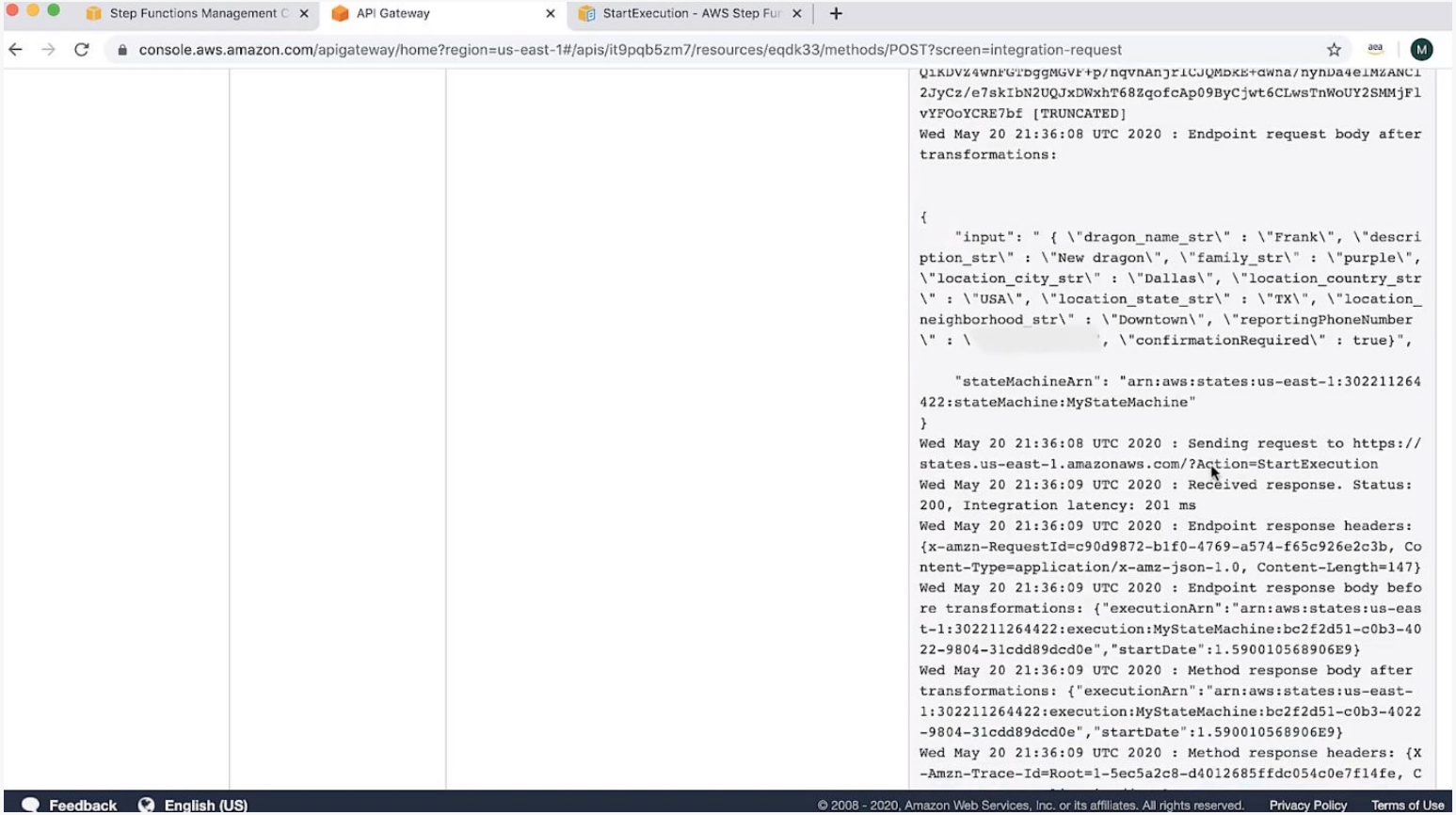
Everything is ready to test (cont.)


Step Functions Management C
API Gateway
StartExecution - AWS Step Fur
console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST?screen=integration-request
Dashboard
Settings
Usage Plans
API Keys
Client Certificates
Settings
Stage Variables
No stage variables exist for this method.
Client Certificate
No client certificates have been generated.
Request Body
{
"dragonName": "Frank",
"description": "New dragon",
"family": "purple",
"city": "Dallas",
"country": "USA",
"state": "TX",
"neighborhood": "Downtown"
"reportingPhoneNumber": "+ ",
"confirmationRequired": true
}
Test
Logs
Execution log for request c1f5496c-bca2-4409-b7f1-962ef2cb4176
Wed May 20 21:36:08 UTC 2020 : Starting execution for request: c1f5496c-bca2-4409-b7f1-962ef2cb4176
Wed May 20 21:36:08 UTC 2020 : HTTP Method: POST, Resource Path: /dragons
Wed May 20 21:36:08 UTC 2020 : Method request path: {}
Wed May 20 21:36:08 UTC 2020 : Method request query string: {}
Wed May 20 21:36:08 UTC 2020 : Method request headers: {}
Wed May 20 21:36:08 UTC 2020 : Method request body before transformations: {
"dragonName": "Frank",
"description": "New dragon",
"family": "purple",
"city": "Dallas",
"country": "USA",
"state": "TX",
"neighborhood": "Downtown",
"reportingPhoneNumber":
"confirmationRequired": true
}
Wed May 20 21:36:08 UTC 2020 : Request validation succeeded for content type application/json
Wed May 20 21:36:08 UTC 2020 : Endpoint request URI: https://states.us-east-1.amazonaws.com/?Action=StartExecution
Wed May 20 21:36:08 UTC 2020 : Endpoint request headers: {Authorization=**********
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

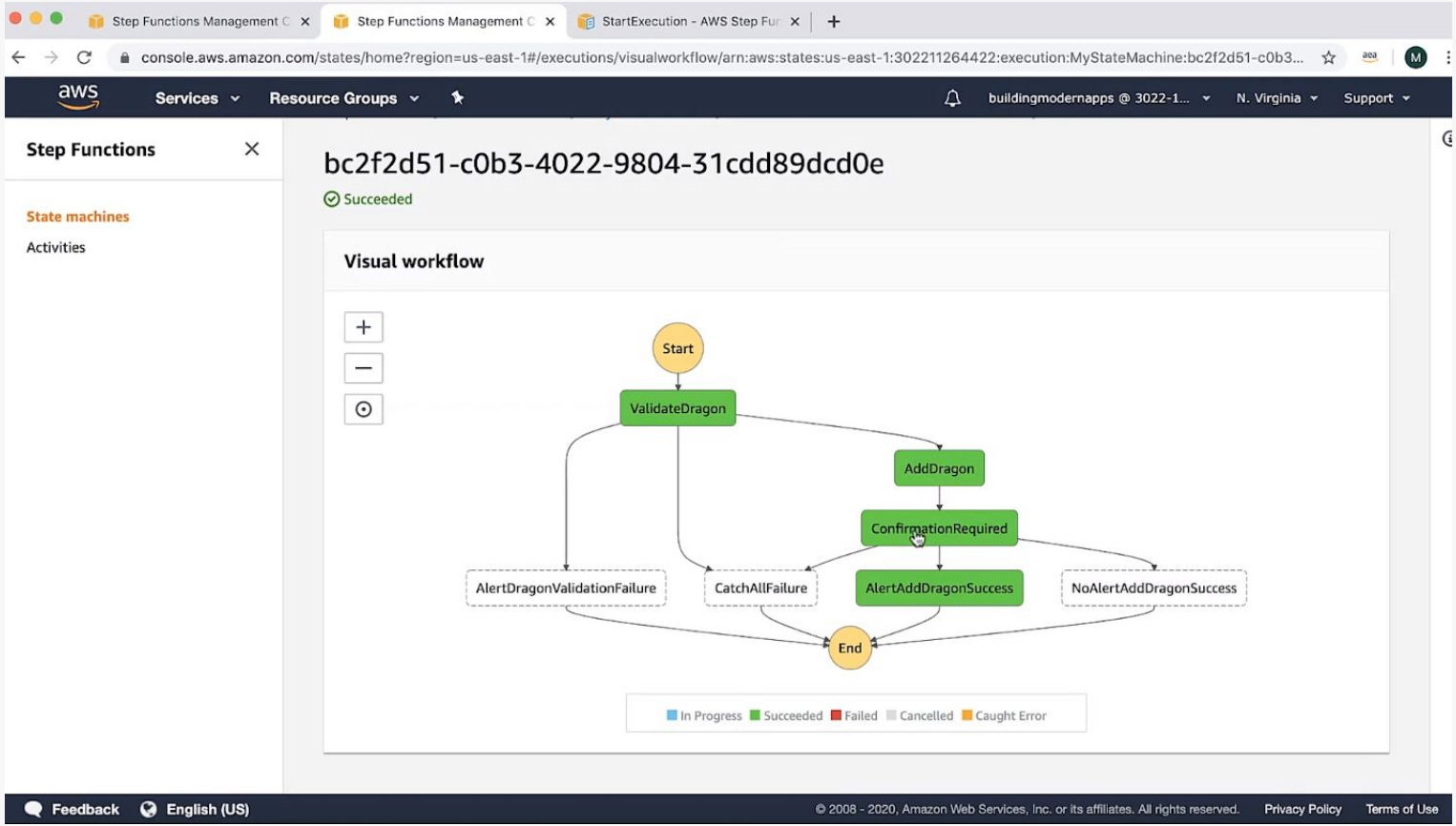
test


Step Functions Management C
API Gateway
StartExecution - AWS Step Fur
console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST?screen=integration-request
Test
Wed May 20 21:36:08 UTC 2020 : Endpoint request URI: https://states.us-east-1.amazonaws.com/?Action=StartExecution
Wed May 20 21:36:08 UTC 2020 : Endpoint request headers:
{Authorization=**********
***b6e0fe, X-Amz-Date=20200520T213608Z, x-amzn-apigateway-api-id=it9pqb5zm7, Accept=application/json, User-Agent=AmazonAPIGateway_it9pqb5zm7, X-Amz-Security-Token=IQoJb3JpZ2luX2VjEMb/////////wEaCXVzLWVhc3QtMSJGMEQCIDPBgZMCwGNC/XppfA5bryPVndCjcPzs7r9kzTg2ulPGAiAakN4/nlBOvAG+wyi9lOIw2+Ey36B7JRRUPzoUx8Si2iqfAggfEAAaDDMwMjIxMTI2NDQyMiIM2Gu9OHak5CfC0KkVKvwBdciD6LVC4f/V9DI5kYoCL4uFBwPTeoZSLSOPj+cmHwk9/Xp2MSU88b/IJaY5XhmCwDq6gNDalvGWsbZTg+Nxpq34dWbm5J9J+kjVkMEEfkCEBpGZnXGaXAysoHS0nLODgpyNxYWEQuxsBbJmkpaRMAVckEoXmr+U2WL9xJkJ+H9mOM1rGDz8c2Mv2FjtujZrdrRmaL/uDLnHa88yZ1IEUApeBFPwJNeBWQiKDVZ4whFGTbggMGVF+p/nqvhAnjrlCJQMbkE+dWna/nyhDa4elMZANC12JyCz/e7skIbN2UQJxDWxhT68ZqofcAp09ByCjwt6CLwsTnWoUY2SMMjFlvYFOoYCRE7bf [TRUNCATED]
Wed May 20 21:36:08 UTC 2020 : Endpoint request body after transformations:
{
"input": " { \"dragon_name_str\" : \"Frank\", \"description_str\" : \"New dragon\", \"family_str\" : \"purple\", \"location_city_str\" : \"Dallas\", \"location_country_str\" : \"USA\", \"location_state_str\" : \"TX\", \"location_neighborhood_str\" : \"Downtown\", \"reportingPhoneNumber\" : \ , \"confirmationRequired\" : true}",
"stateMachineArn": "arn:aws:states:us-east-1:302211264422:stateMachine:MyStateMachine"
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

test result



We can see the mapping here



visual workflow of this execution

Good Resource for learning VTL

Resolver mapping template programming guide - AWS AppSync

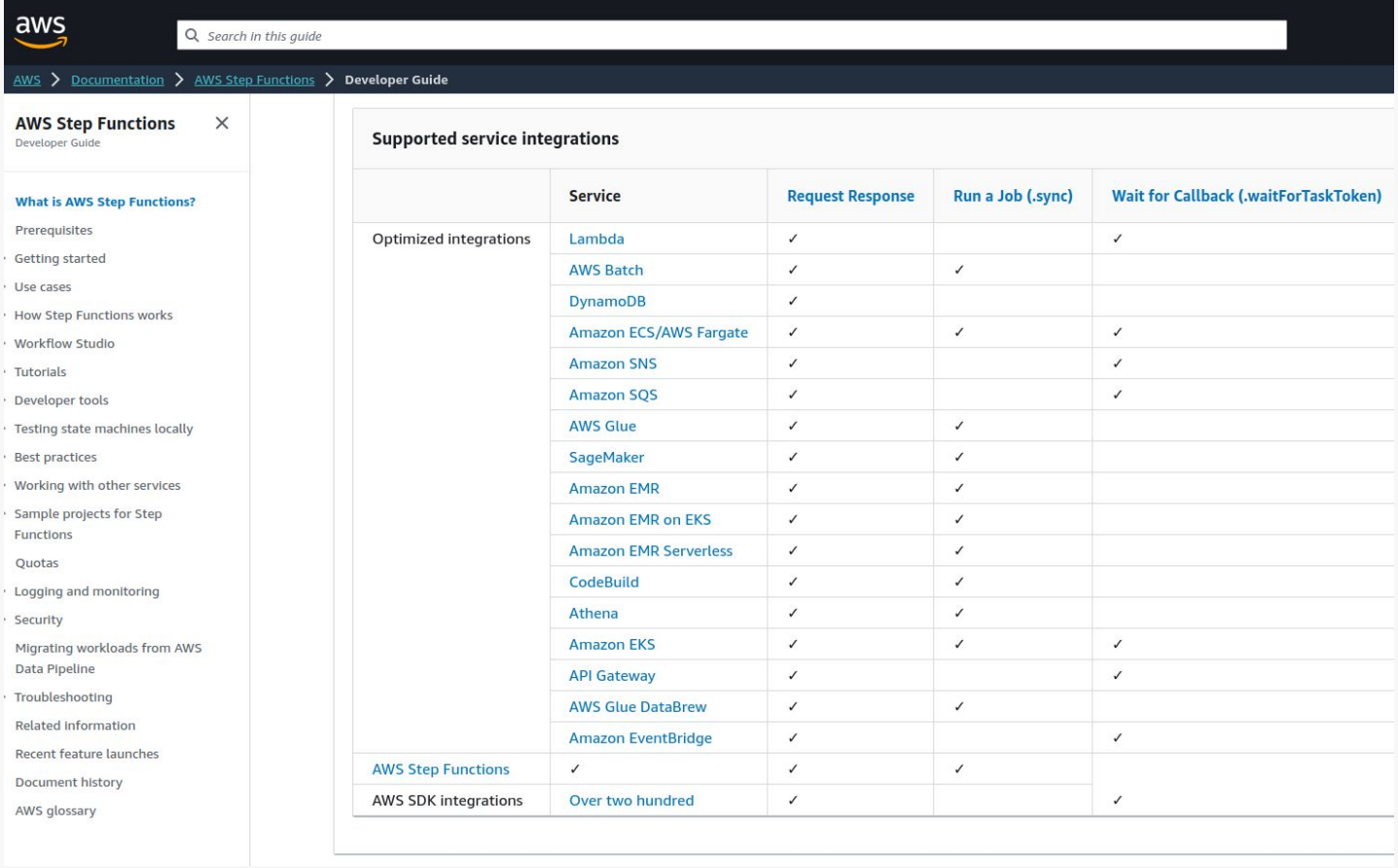
Apache Velocity Engine VTL Reference

Run a Job and Wait for Callback Patterns

In the service integration of Step Functions,
we can call different AWS services directly.

There are three patterns for this integration:

Request Response, Run a Job, and Wait for Callback



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

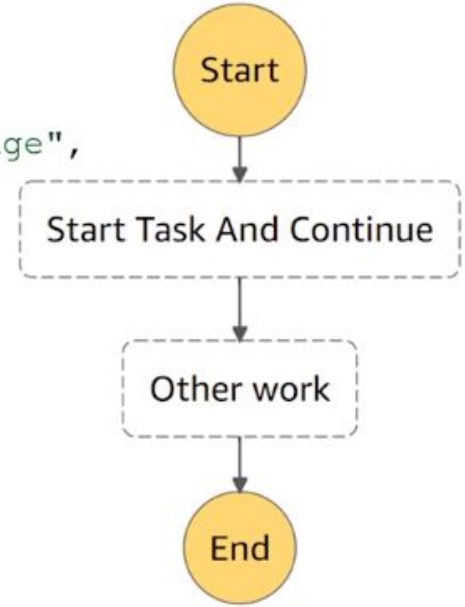
So far, we been looking at the first pattern.

In this section, we want to go in more details on the other two patterns.

Let's take an example, with the Amazon Simple Queue Service (SQS).

This is a queue service, which you can access with the HTTPS Protocol to send messages and pull message out of that queue.

```
"Start Task And Continue": {
   "Type": "Task",
   "Resource": "arn:aws:states:::sqs:sendMessage",
   "Parameters": {
      "QueueUrl": "<SQS QUEUE URL>",
      "MessageBody.$": "$.input.message"
   }
}
```

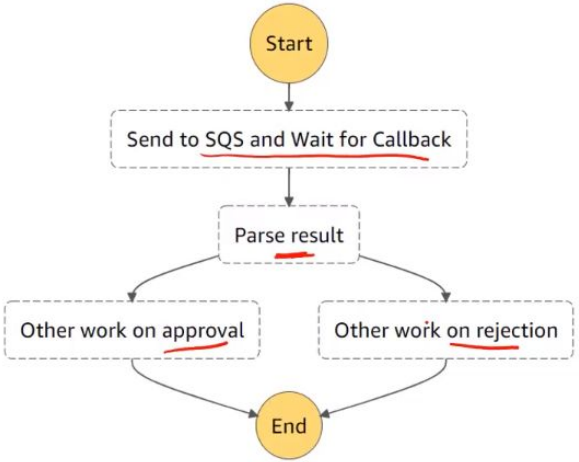


The way we are calling SQS here is via the request response pattern.

This means, that Step Functions will send a message in the queue,
wait to make sure it was successfully accepted by the SQS Service,
and if so, it will proceed with the task “Other work”.

This means that it doesn't matter what's happening with the message in the queue,
the state machine we'll just keep going.

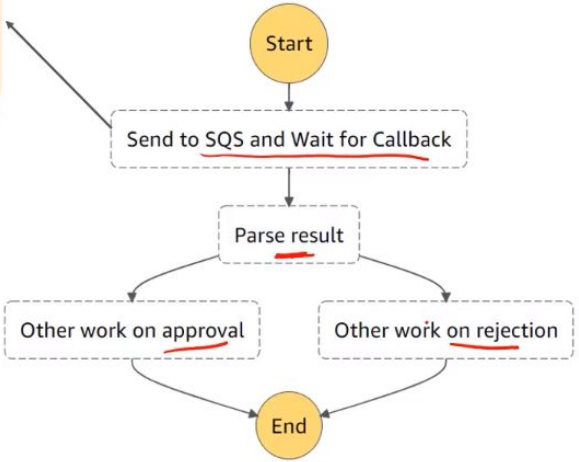
This is a valid pattern, but sometimes you may want to wait for that message to be pooled and be processed.



in this example, we choose a path based on the result from the SQS

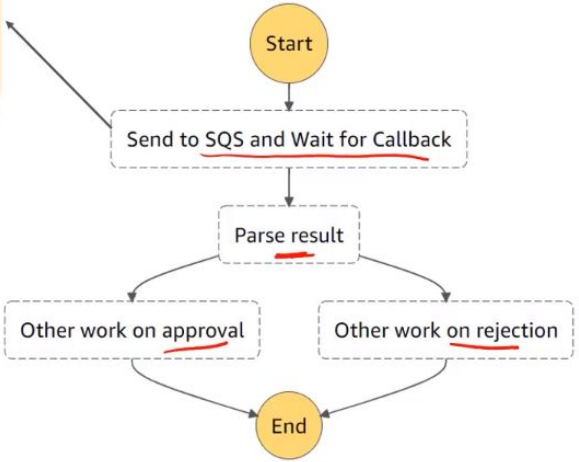
Task calls SQS with task token

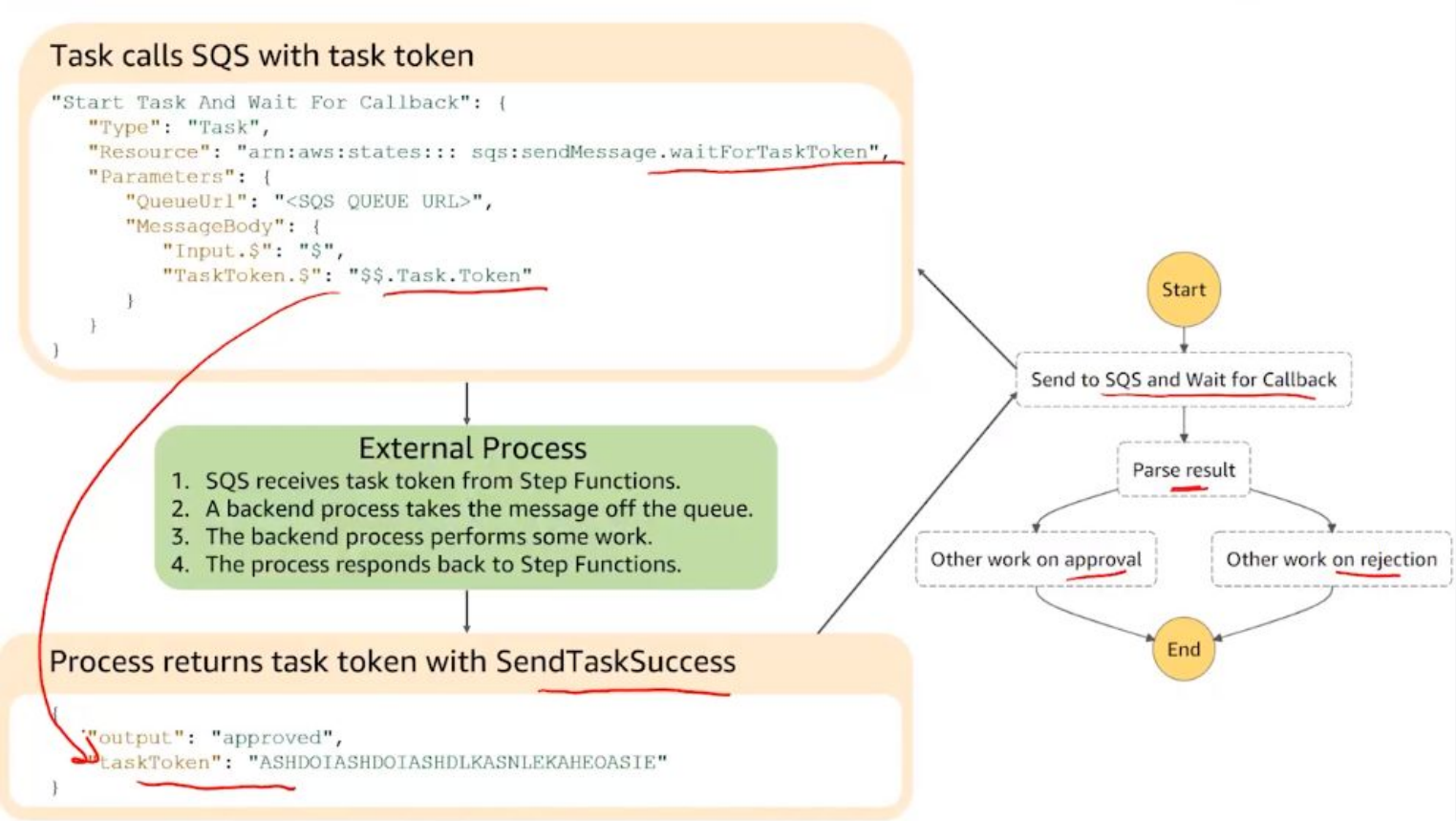
```
"Start Task And Wait For Callback": {
   "Type": "Task",
   "Resource": "arn:aws:states::: sqs:sendMessage.waitForTaskToken",
   "Parameters": {
      "QueueUrl": "<SQS QUEUE URL>",
      "MessageBody": {
         "Input.$": "$",
         "TaskToken.$": "$$.Task.Token"
      }
   }
}
```



Task calls SQS with task token

```
"Start Task And Wait For Callback": {
   "Type": "Task",
   "Resource": "arn:aws:states::: sqs:sendMessage.waitForTaskToken",
   "Parameters": {
      "QueueUrl": "<SQS QUEUE URL>",
      "MessageBody": {
         "Input.$": "$",
         "TaskToken.$": "$$.Task.Token"
      }
   }
}
```





call, wait, and callback processes

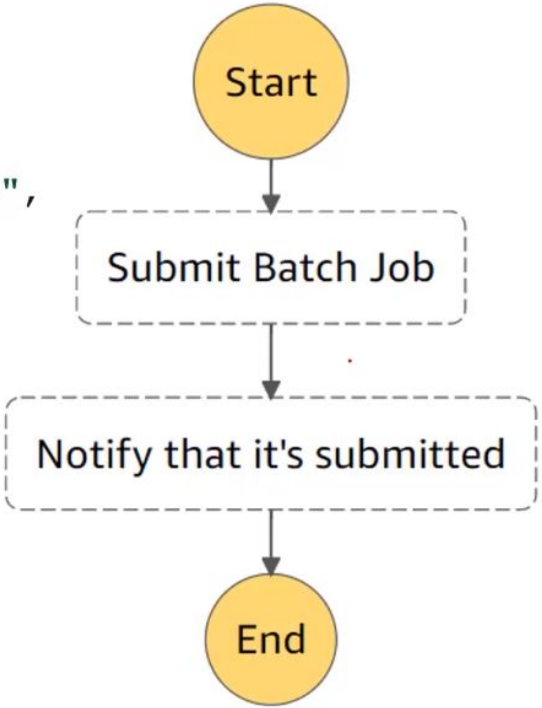
The third pattern is the RunAJob or .SyncCommand

This is for an integration with a service where you want Step Functions to wait for a request to complete before progressing to the next state.

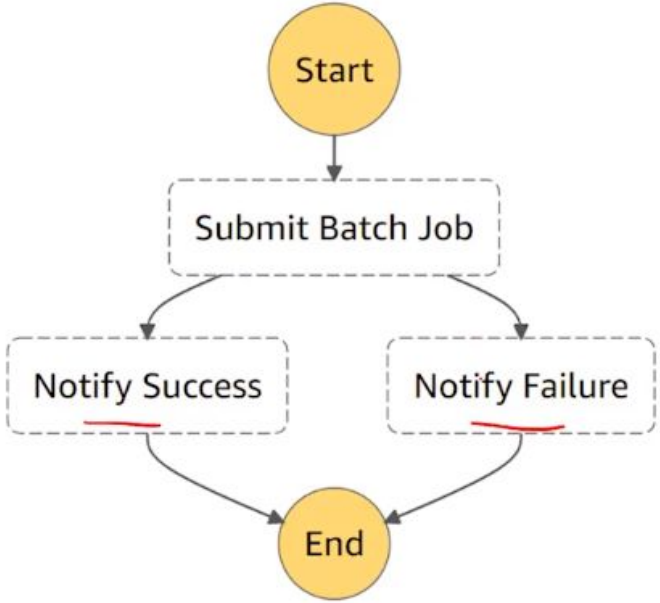
Let's take an example with the AWS Batch Service,
which allows to spin up some Servers,
execute scripts on those Servers to perform some work,
and tears down all of those Servers down afterward.

when executing this call, if all you cared about is to start AWS Batch (successful and not), then run that service as a request response.

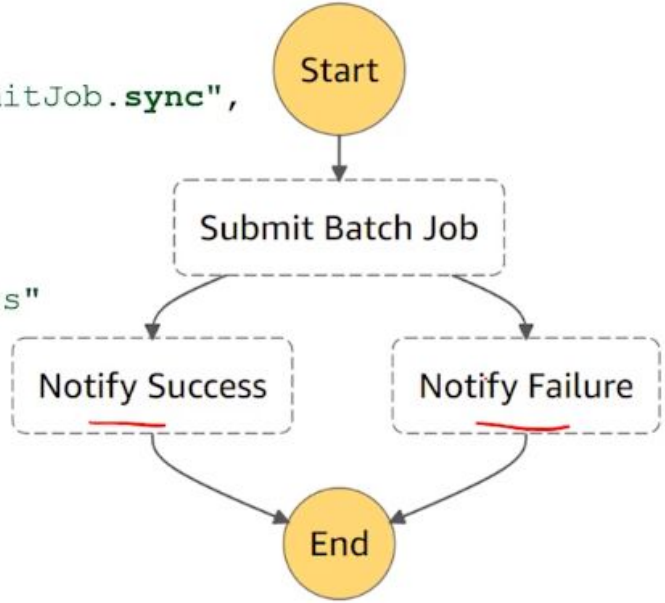
```
"Submit Batch Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::batch:submitJob",
    "Parameters": {
        "JobDefinition": "preprocessing",
        "JobName": "preprocessingBatchJob",
        "JobQueue": "primaryQueue",
        "Parameters.$": "$.batchjob.parameters"
    }
}
```



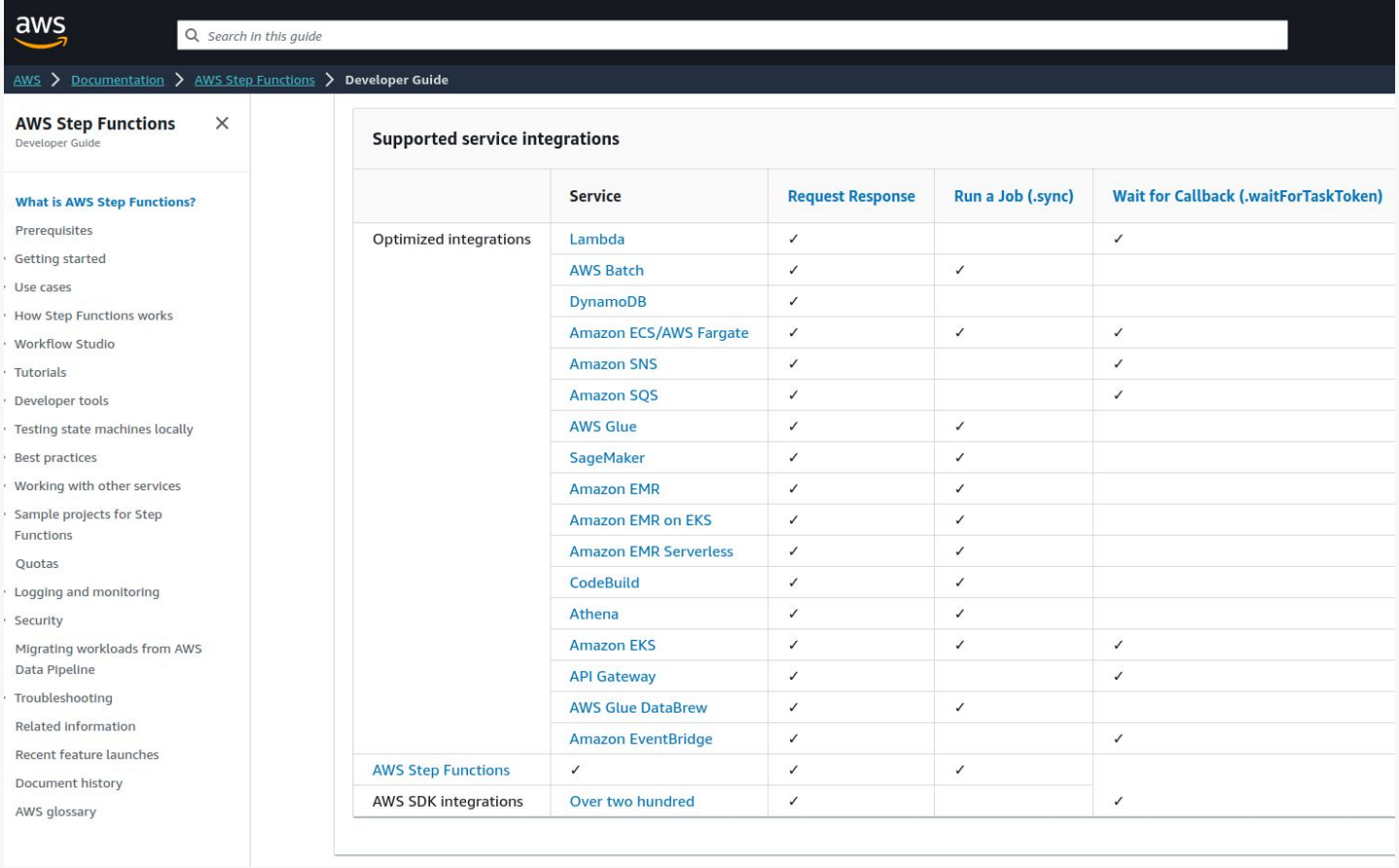
However, if you wanted Step Functions to wait
for the execution of that job to finish in AWS batch,
to know if it ran successfully, or if it errored out,
that workflow does not help.


Start
Submit Batch Job
Notify Success
Notify Failure
End

```
"Submit Batch Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::batch:submitJob.sync",
    "Parameters": {
        "JobDefinition": "preprocessing",
        "JobName": "preprocessingBatchJob",
        "JobQueue": "primaryQueue",
        "Parameters.$": "$.batchjob.parameters"
    }
}
```



we need to modify the Resource with adding .sync



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

(recap)

Standard vs Express Step Functions

Up until now, we have been discussing standard workflows in Step Functions.

But there are actually two types of workflows: Standard and Express.

	Standard workflow	Express workflow
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

	Standard workflow	Express workflow
State transition persistence	Persisted to disk	In memory only
Execution semantics	Exactly-once workflow execution	At-least-once workflow execution

	Standard workflow	Express workflow
Pricing	Priced per state transition	Priced by the number of executions you run, their duration, and memory consumption
Maximum duration	365 days	5 minutes

	Standard workflow	Express workflow
Execution history	Stored in Step Functions and can be sent to Amazon CloudWatch Logs	Sent to Amazon CloudWatch Logs

	Standard workflow	Express workflow
Service integrations	Supports all service integrations and patterns	Supports all service integrations. Does not support Job-run (.sync) or Callback (.waitForTaskToken) patterns.

Good Resources:

Choosing Standard or Express Workflows - AWS Step Functions

Event Driven Architectures

A client sends an event through API gateway
and lets step function take to responsibility for doing what it has to do.

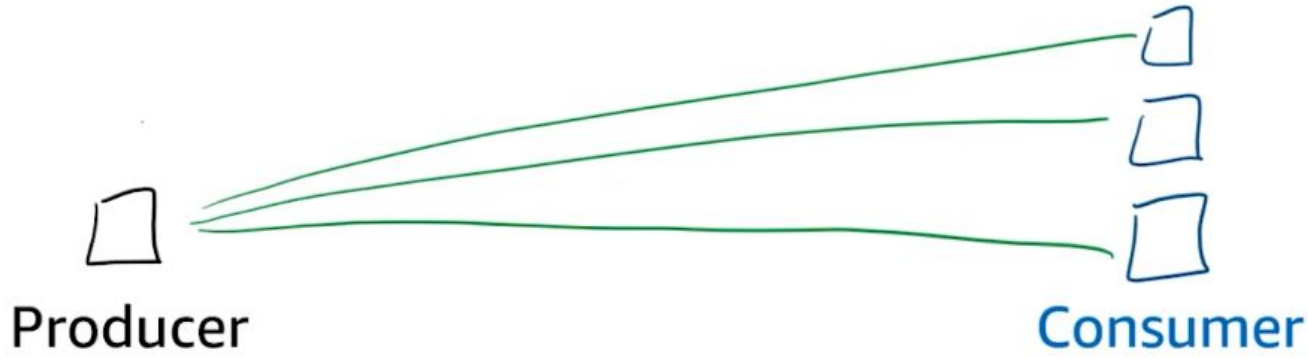
This is an event-driven architecture.

An event-driven architecture uses events to
trigger and communicate between decoupled services
and is common in modern applications, built with microservices.

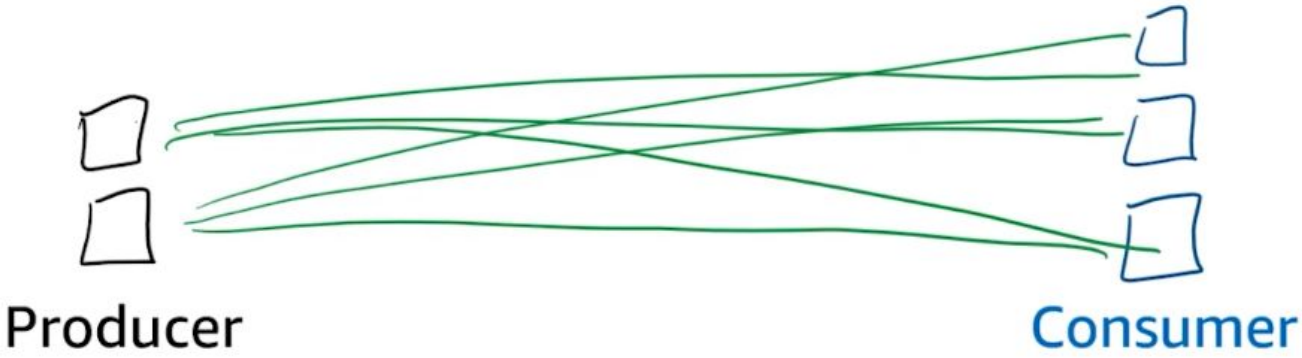
In an event-driven system, the flow of the program is determined by events such as user actions (clicks, keystrokes), sensor outputs, or messages from other programs or services.

When we talk about events,
there is always a producer and a consumer.

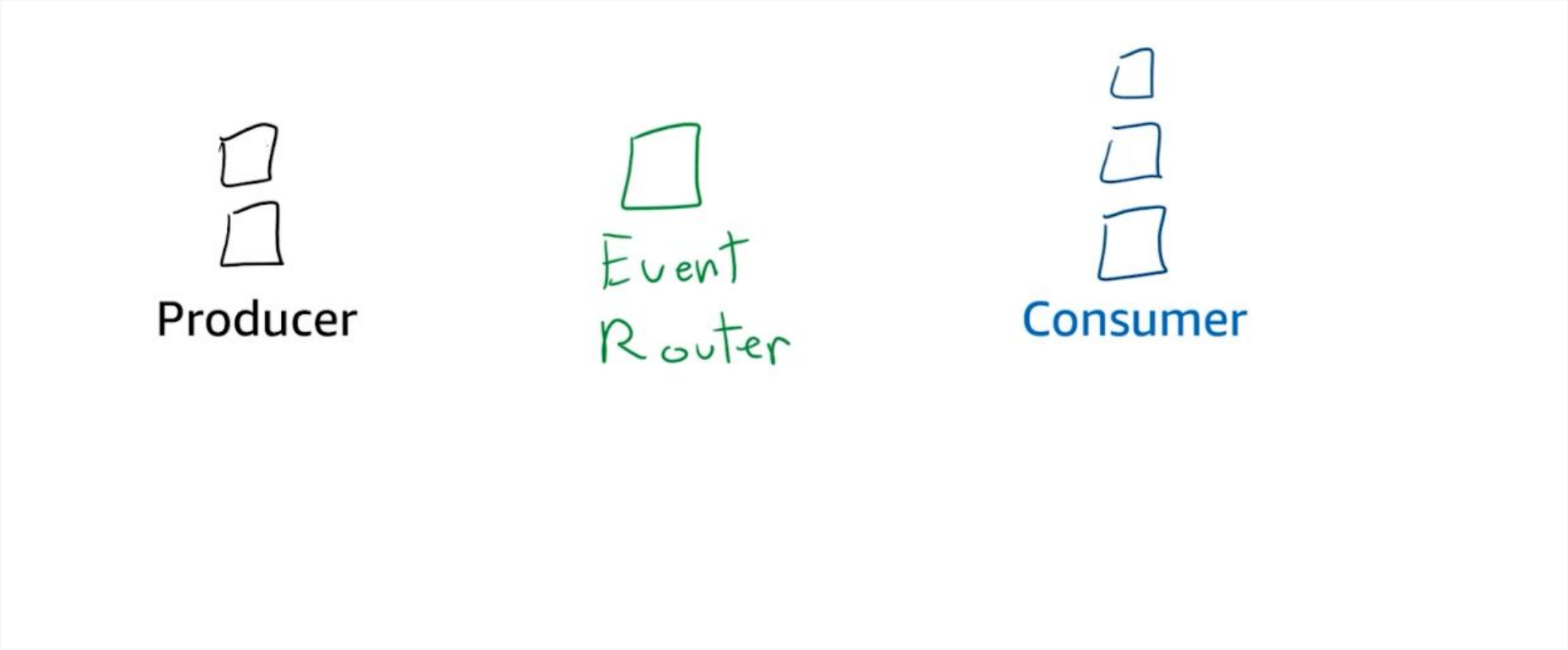
When there are only one producer and one consumer, it's not very difficult.



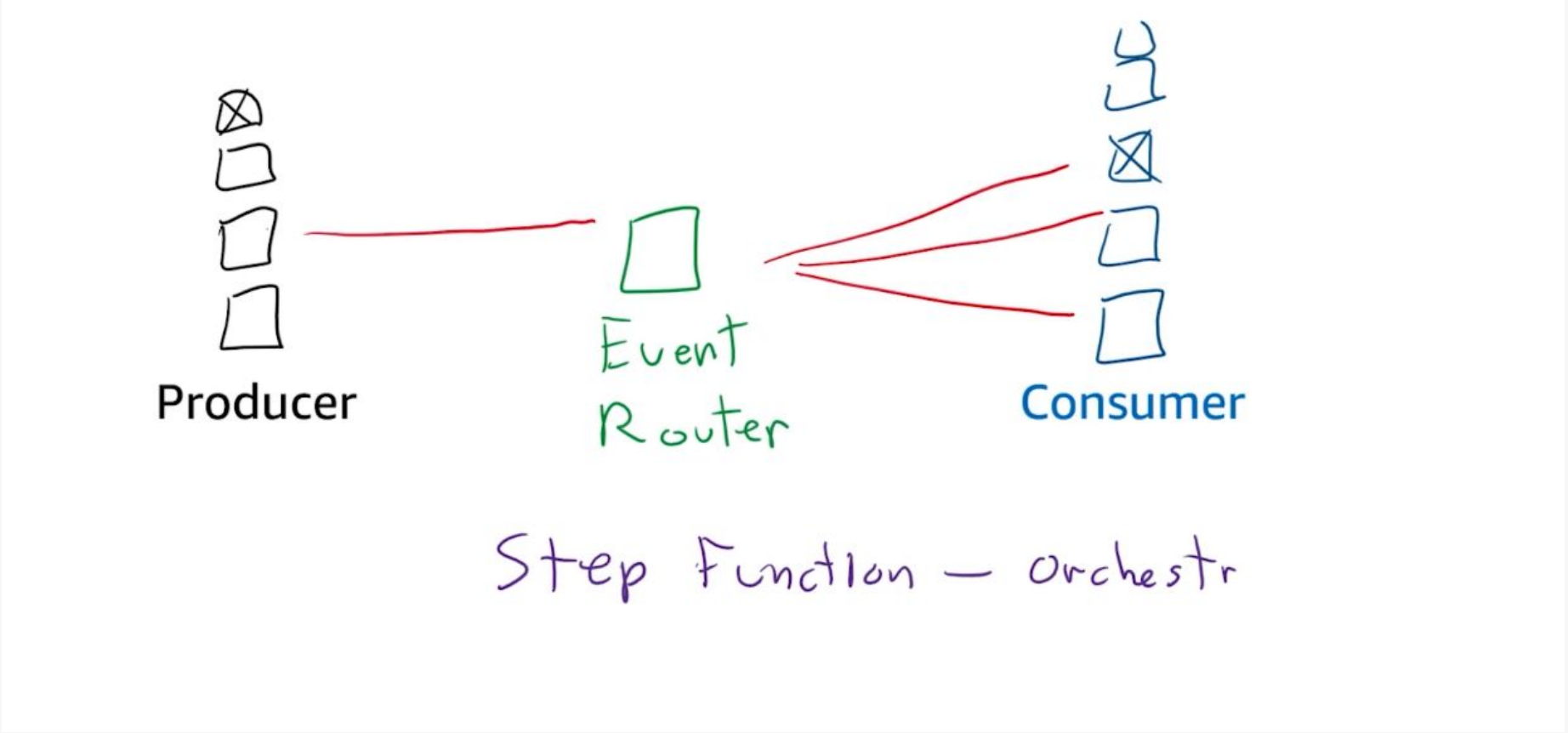
with more consumer, the producer need to provide service to more consumers



finally, you have to add more producer which your architecture become a mess



Event Router decouple producers and consumers



Step Functions could act as a router, as it orchestrates events.

There are three more AWS services
that can act as routers.

Use them if you don't want as much orchestration as what Step Functions provide.

The first is a fully managed messaging queue service:
Amazon Simple Queue Service or SQS.

Using SQS, you can send, store and receive messages
between producers and consumers at any volume
without losing messages or requiring other services to be available.

Messages will stay in the queue for up to 14 days.

SQS offers two types of message queues: standard queues and FIFO

Standard queue offers a maximum throughput
but with a best effort ordering and in at least once delivery.

SQS First-In-First-Out are designed to guarantee that messages
are processed exactly once and in the exact order that they really have been sent.

To send messages, your producer will use the HTTPS protocol
and ind to receive messages,
your consumer will need to pull from the queue using the HTTPS protocol as well.

if there are no messages in the queue,
consumer is staying there, idling, and waiting for a message to finally arrive.

That's actually how queues work.

If you like your consumer to be notified that something was sent by a producer, that's where you need a publish subscribe messaging service.

Amazon Simple Notification Service or SNS

SNS is a fully managed publish-subscribe messaging service
that's highly available, durable and secure.

It uses SNS topics for high-throughput, push-based, many-to-many messaging.

The third service is Amazon EventBridge service.

It's a service bus that makes it easy to connect producers and consumers together using data your own application or integrated software as a service application, or even AWS services.

Good Resources: SQS, SNS, EventBridge

Amazon Simple Queue Service

Getting started with Amazon SNS - Amazon Simple Notification Service

Amazon EventBridge Documentation

Exercise 5: AWS Step Functions