

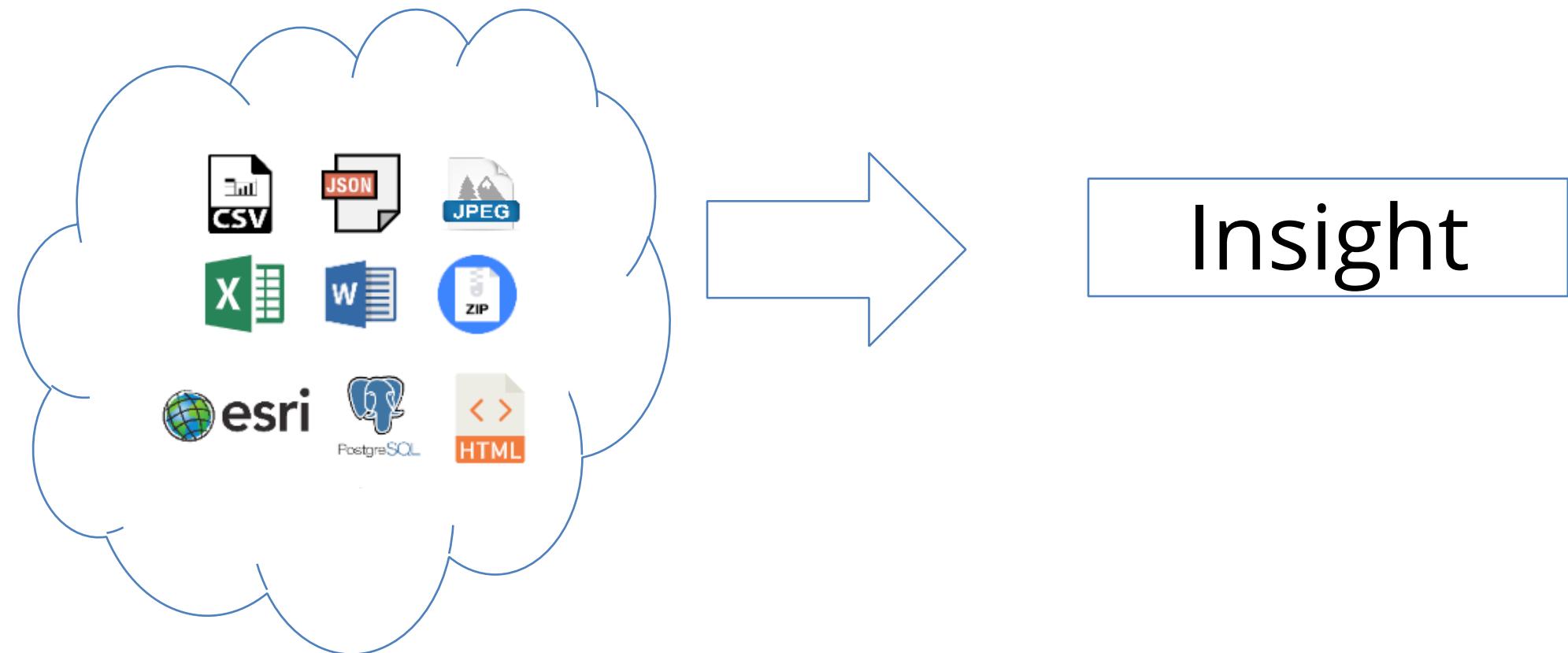


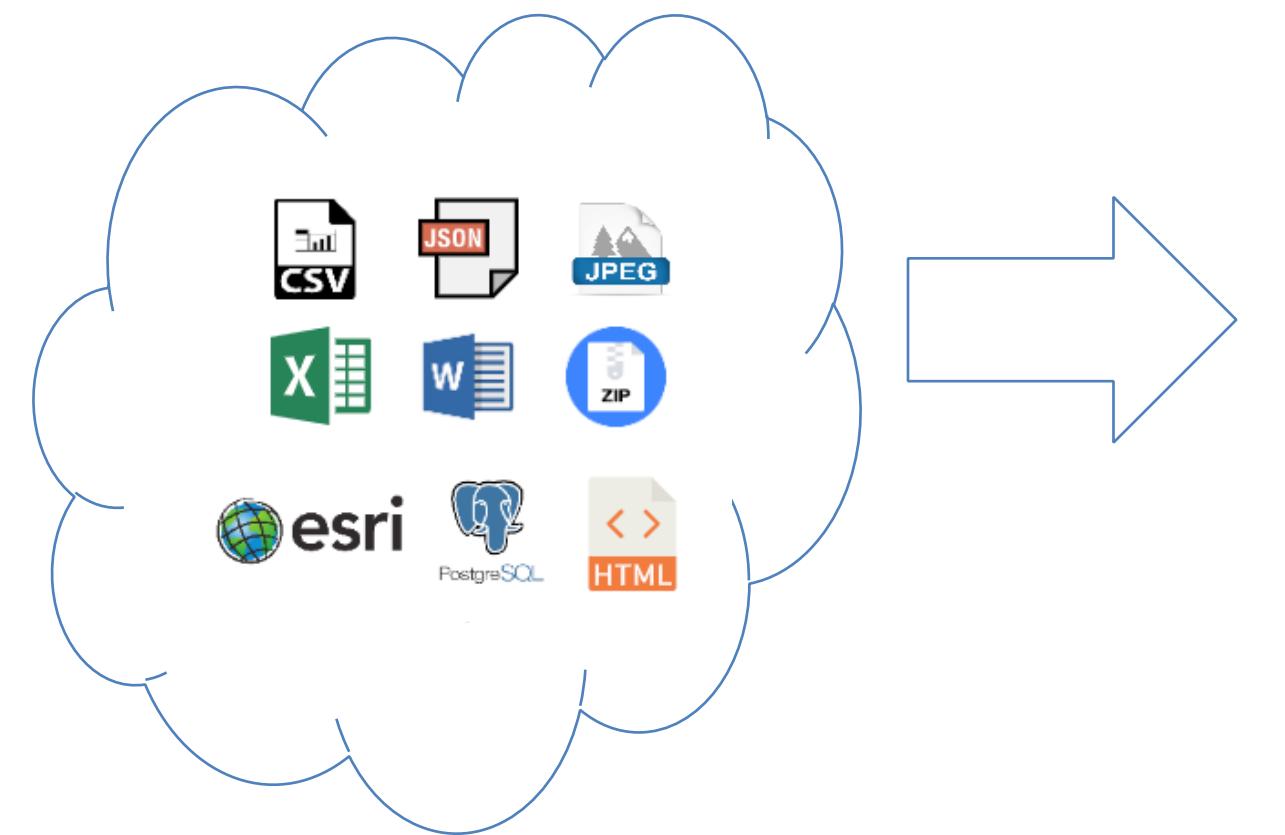
# Working with large numbers of non-trivial ETL pipelines.

OR I'VE ALREADY STUFFED IT UP SO YOU DON'T HAVE TO

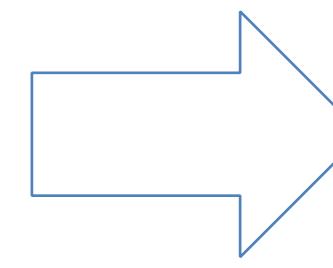
JESSICA FLANAGAN

# Our Business Model

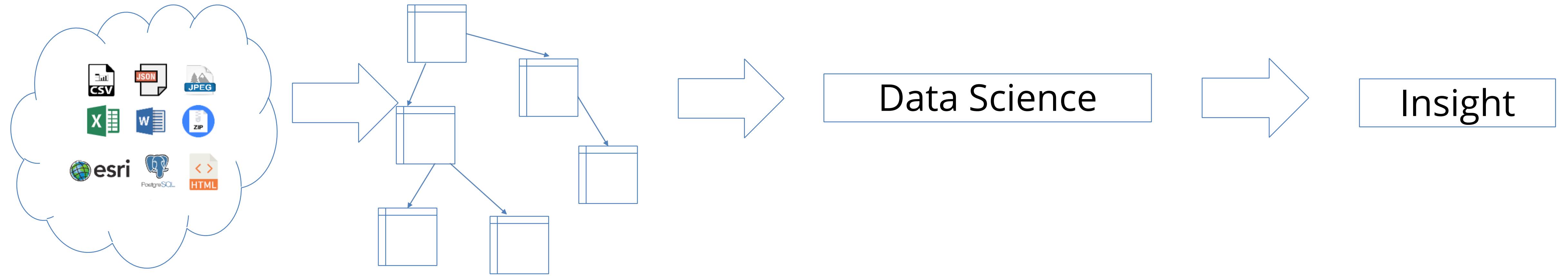




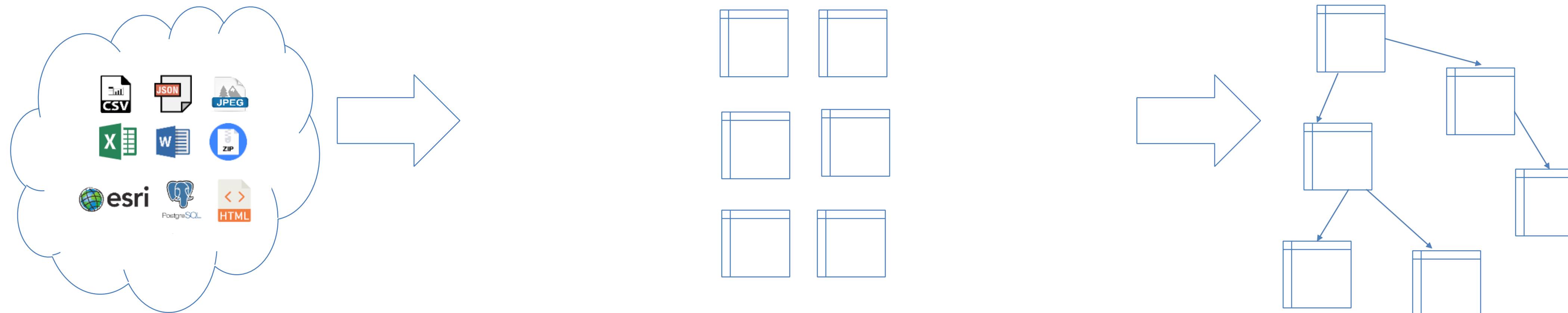
Data Science



Insight



# Our Basic Workflow



# Problem Space

## What is a non trivial ETL pipeline?

- If you just need to trigger a lambda when you put something in S3 you don't need a pipeline.
- This talk is about batch pipelines not streaming although some concepts may apply.

## What is a large number of pipelines?

- Without proper monitoring you have to treat pipelines like children deciding how many you can/want to look after is an exercise for the reader.

# Architecture Goals



**Create Pipelines that are easy to debug.  
When they fail we need to see where the  
failure occurred and any related logging.**

# Architecture Goals



**Create Pipelines that are easy to debug.  
When they fail we need to see where the  
failure occurred and any related logging.**



**We need a whole of system view so we  
can see that our pipelines are  
running and which succeed or fail.**

# Architecture Goals



**Create Pipelines that are easy to debug.**  
When they fail we need to see where the failure occurred and any related logging.



**Monitoring so we can see if our pipelines are degrading or producing less results over time.**



**We need a whole of system view so we can see that our pipelines are running and which succeed or fail.**

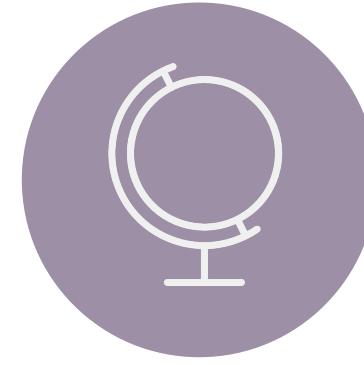
# Architecture Goals



**Create Pipelines that are easy to debug.**  
When they fail we need to see where the failure occurred and any related logging.



**Monitoring so we can see if our pipelines are degrading or producing less results over time.**

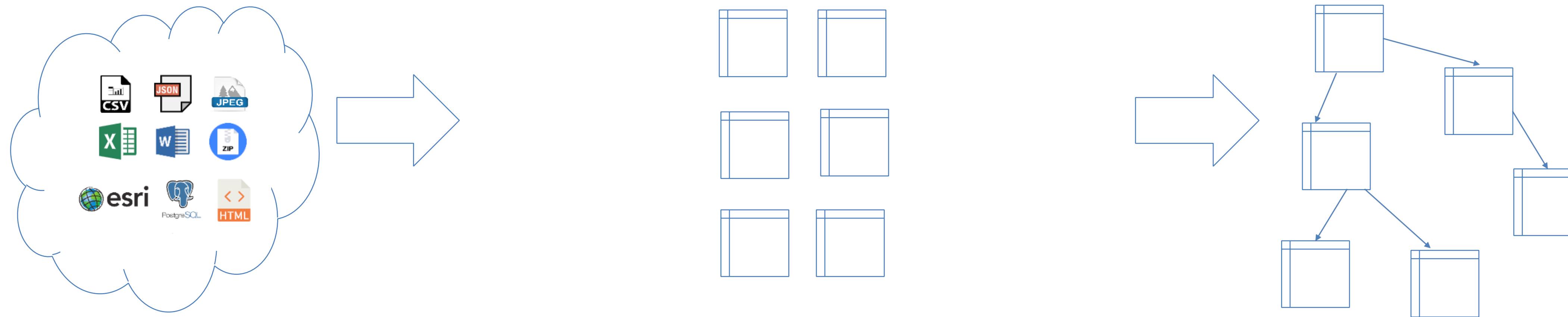


We need a whole system view so we can see that our pipelines are running and which succeed or fail.

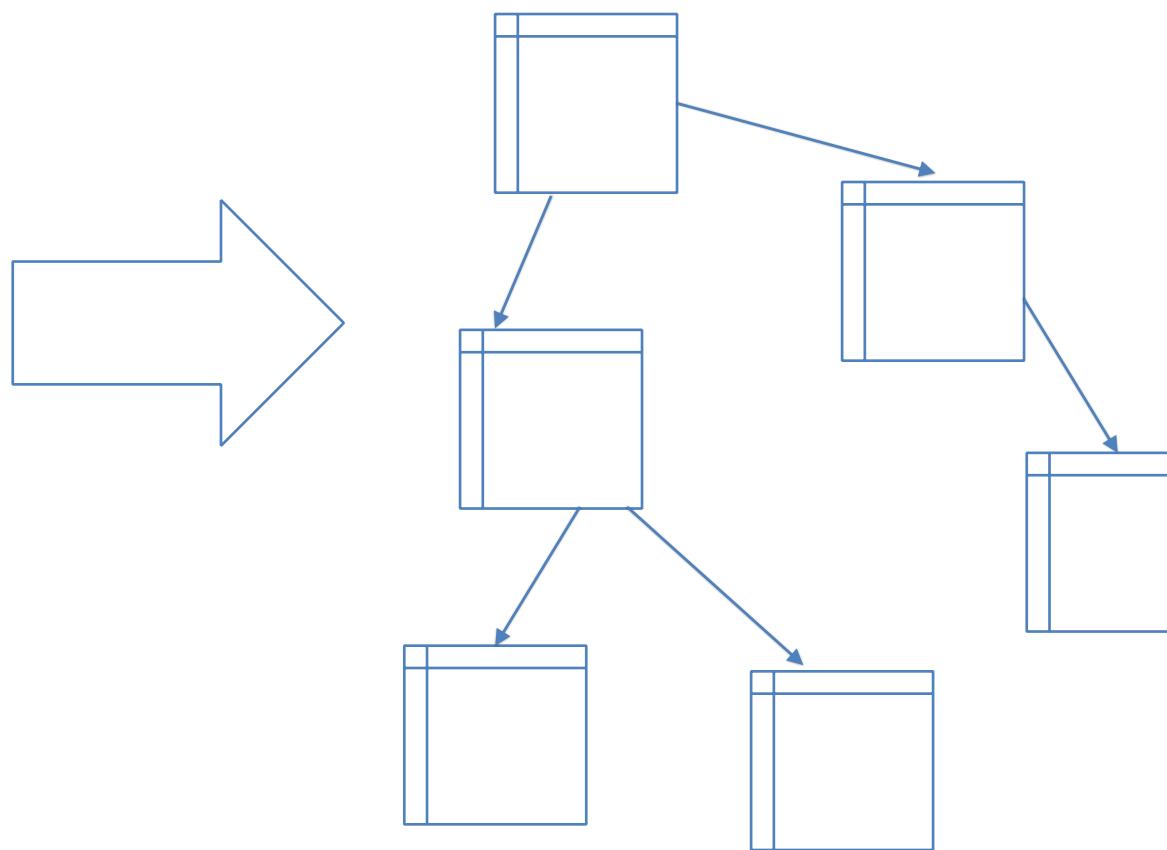
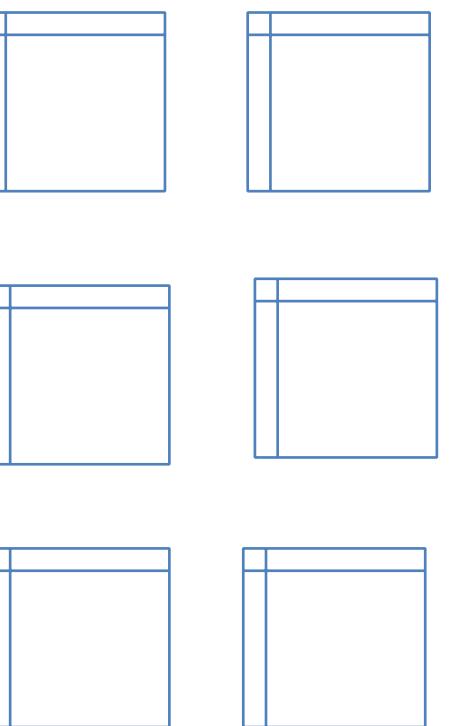
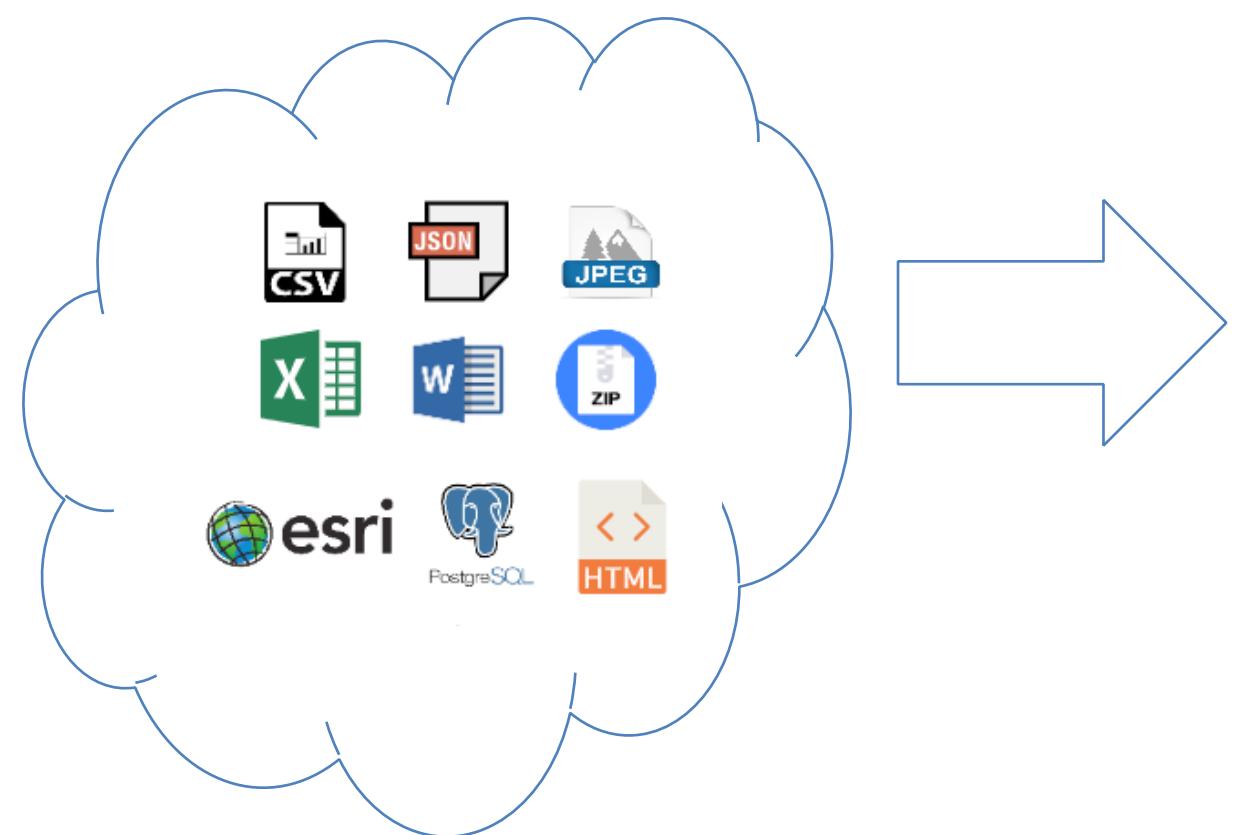


**A way of making sure that if our pipelines fail that someone is tasked with fixing the problem.**

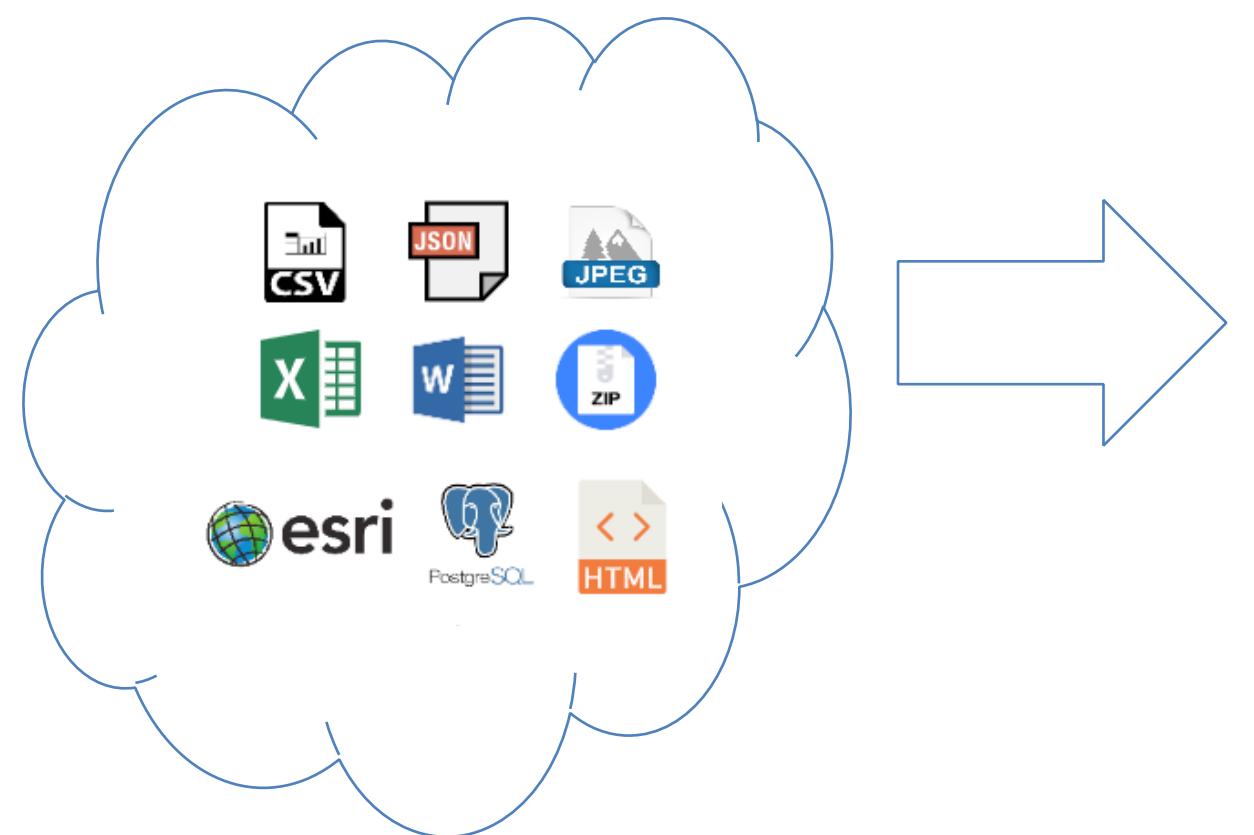
# Our Basic Workflow



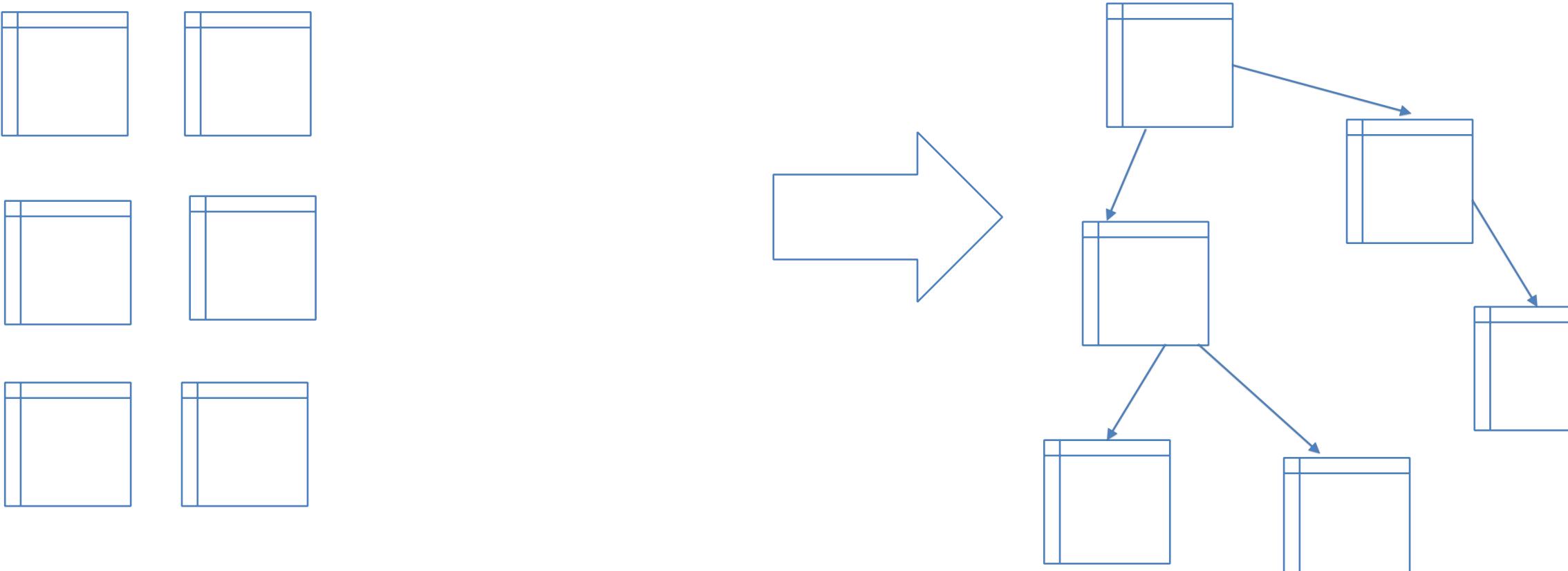
# Extract



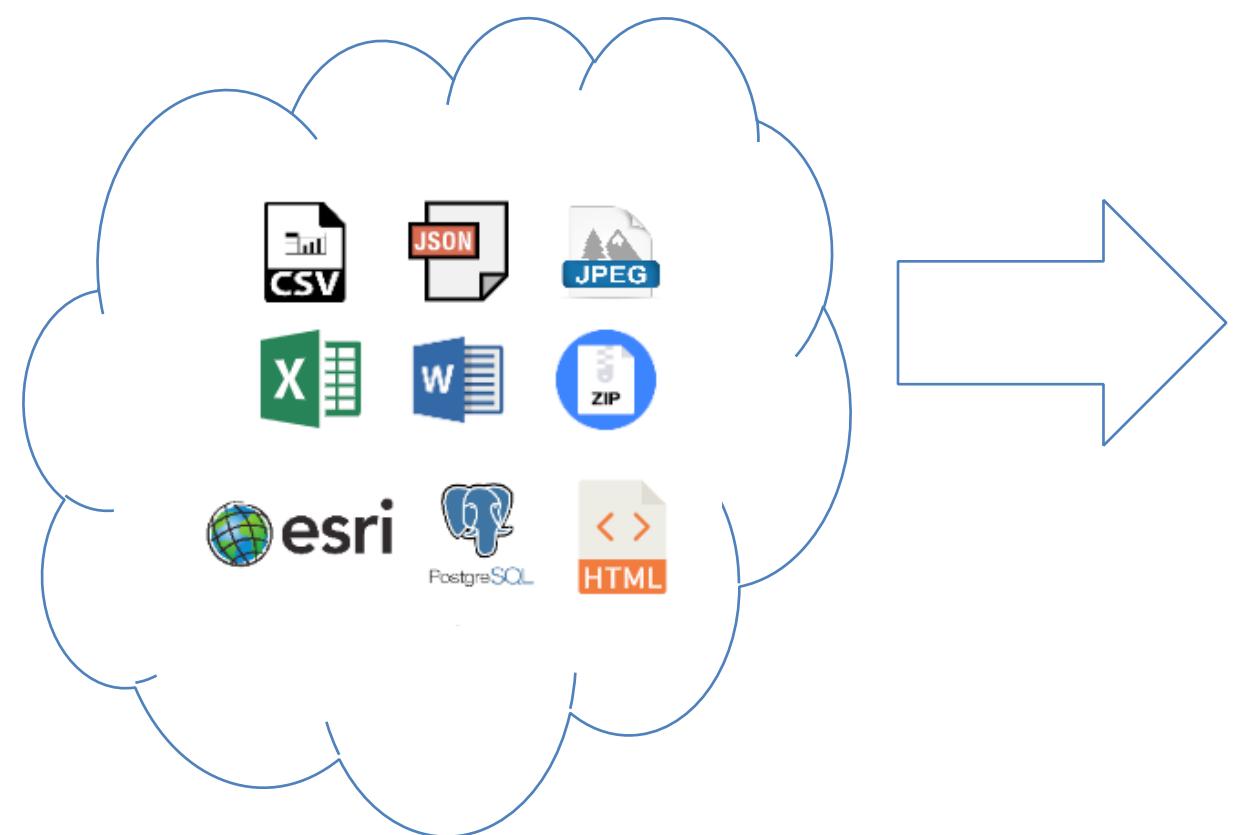
# Extract



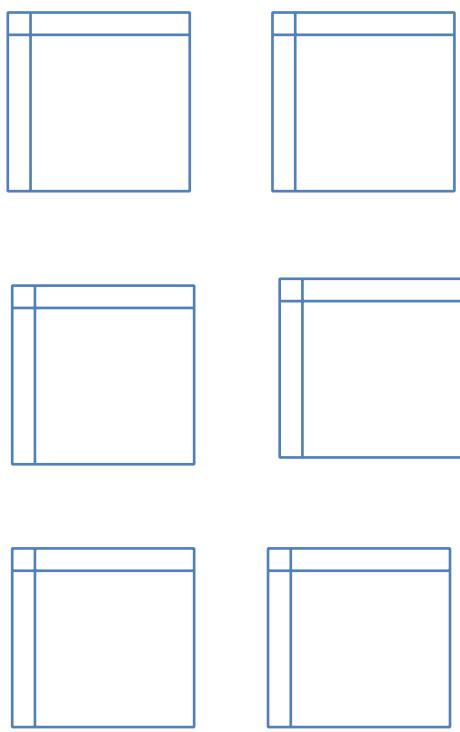
# Transform



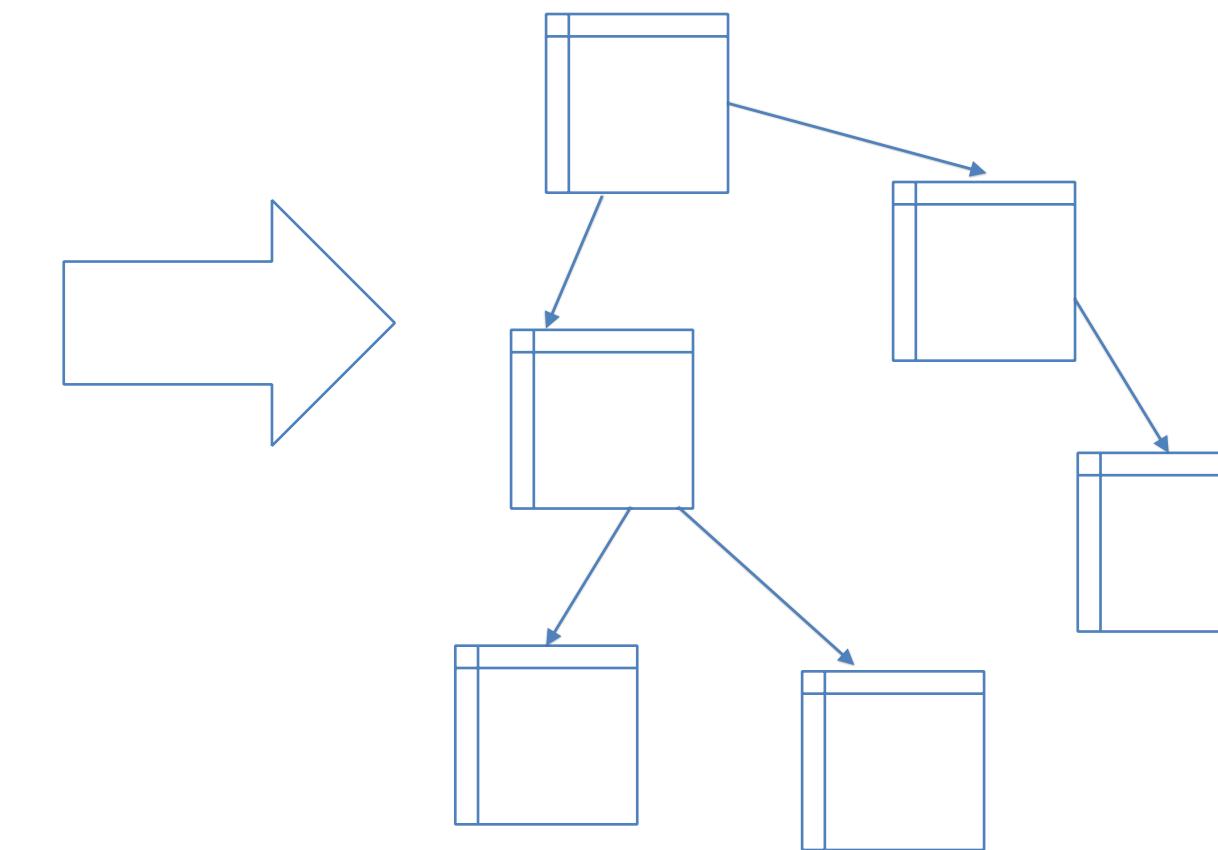
# Extract



# Transform



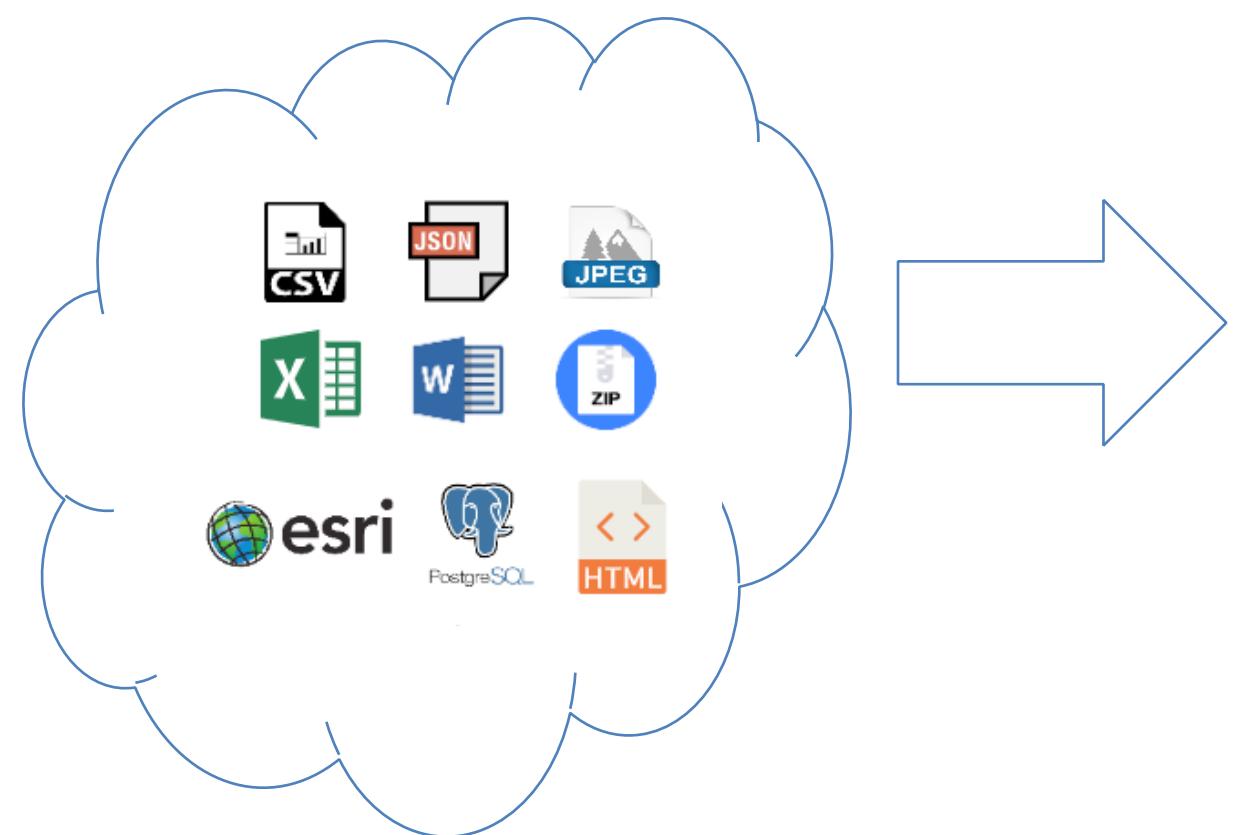
# Load



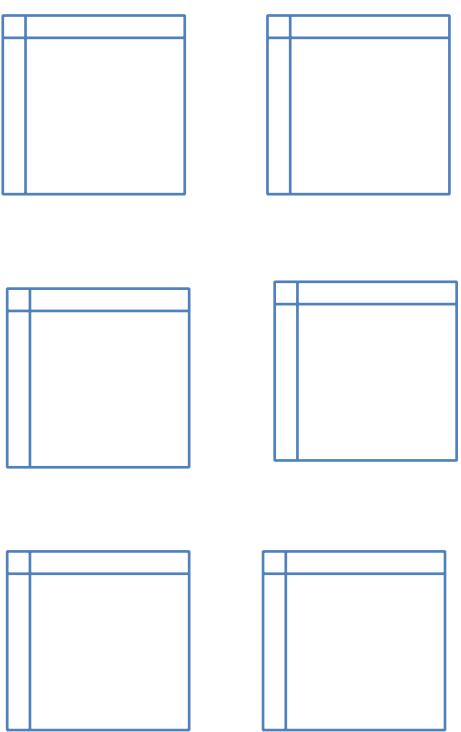


**Wherever possible show your working.  
Allow data scientists to access the data  
before and after transformation.**

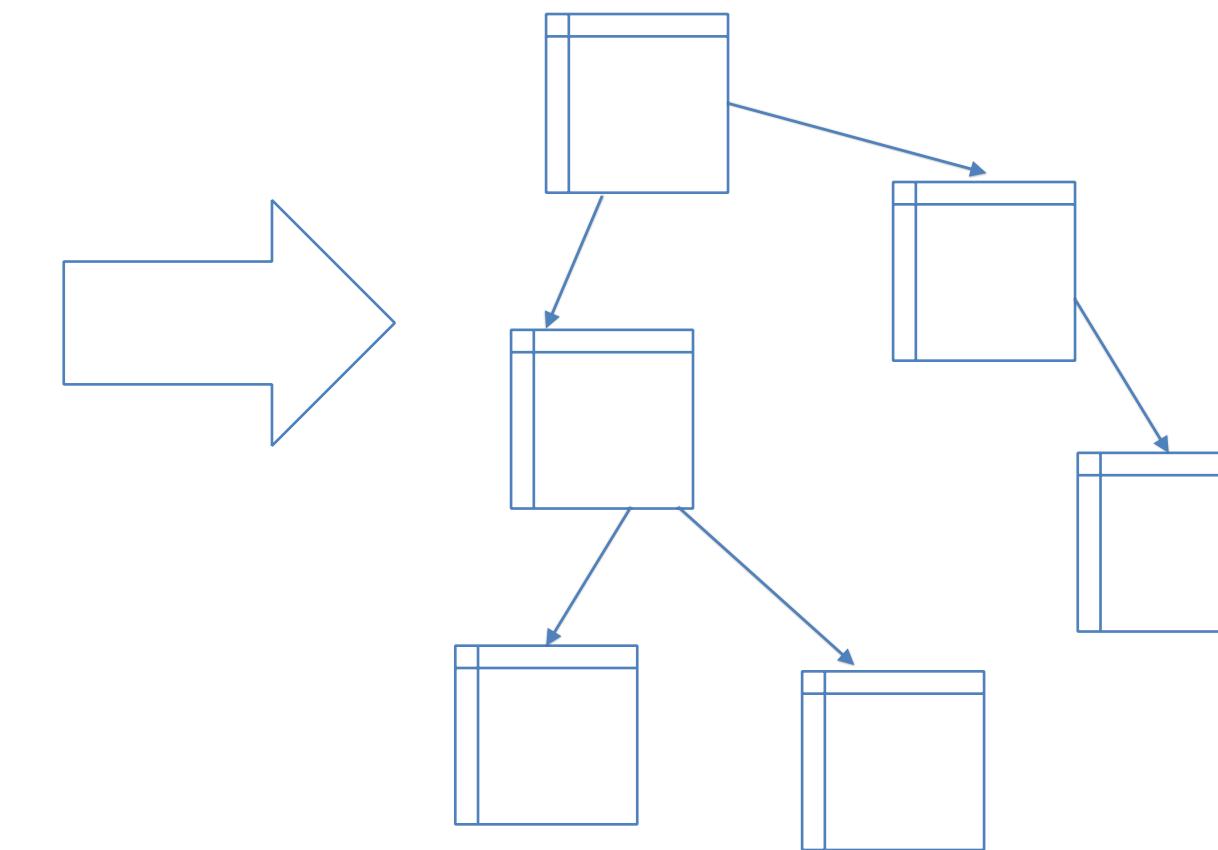
# Extract



# Transform



# Load



# Extract



OR



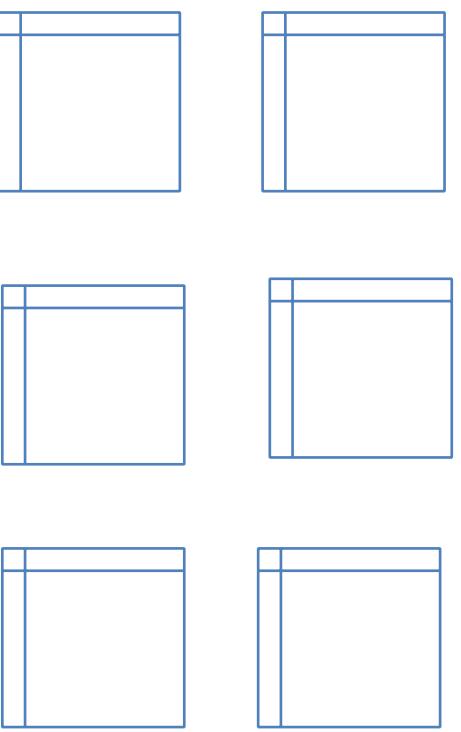
Amazon  
Lambda



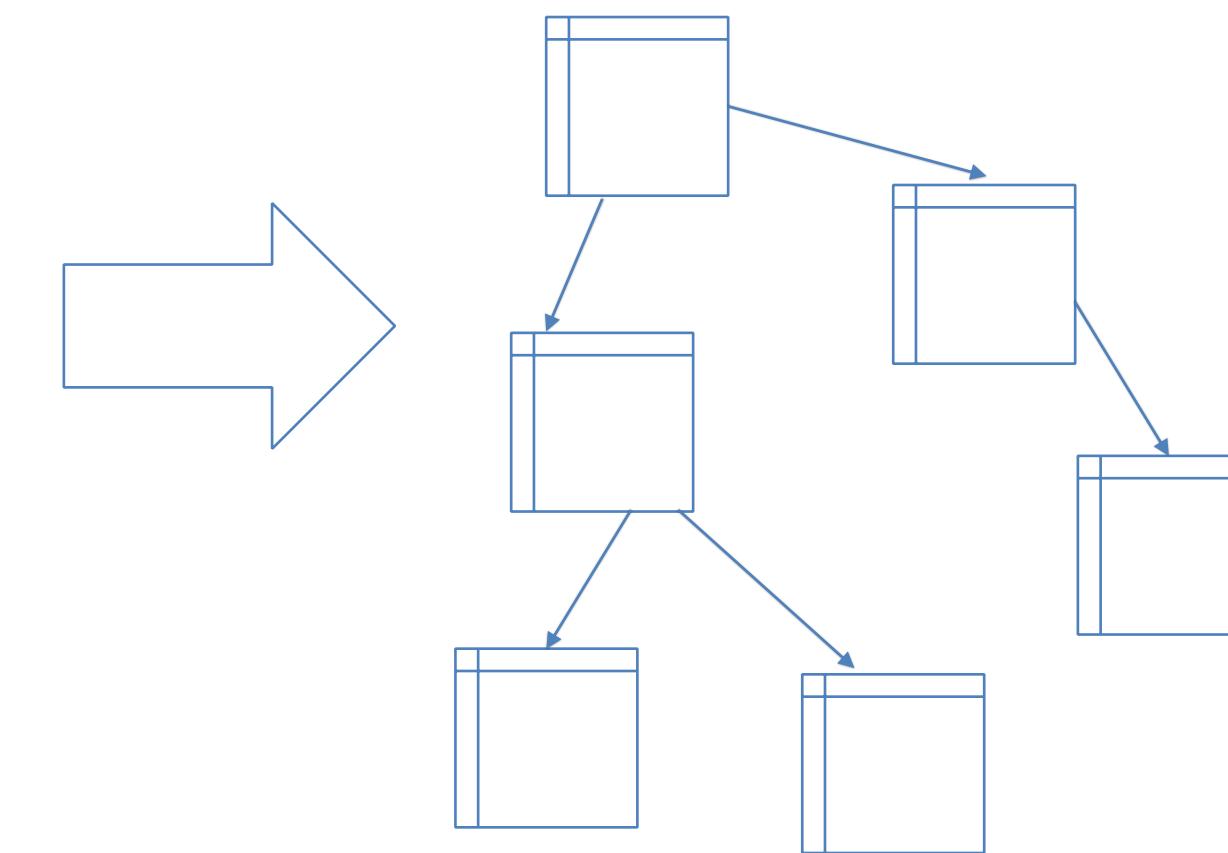
S3 - Extract



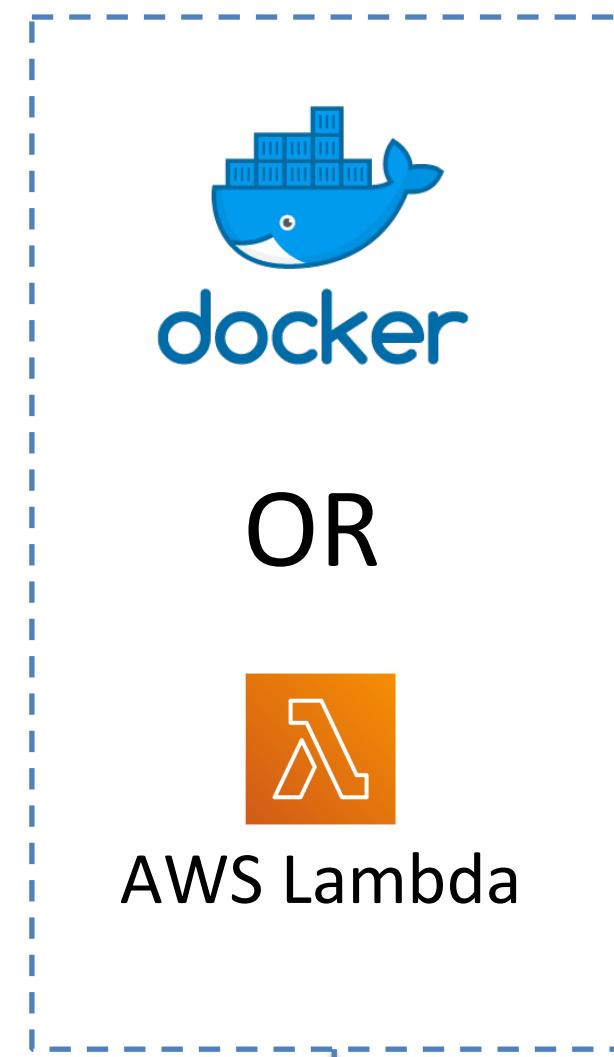
# Transform



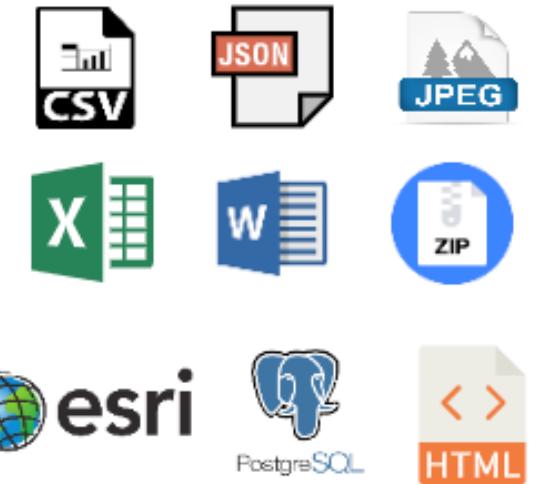
# Load



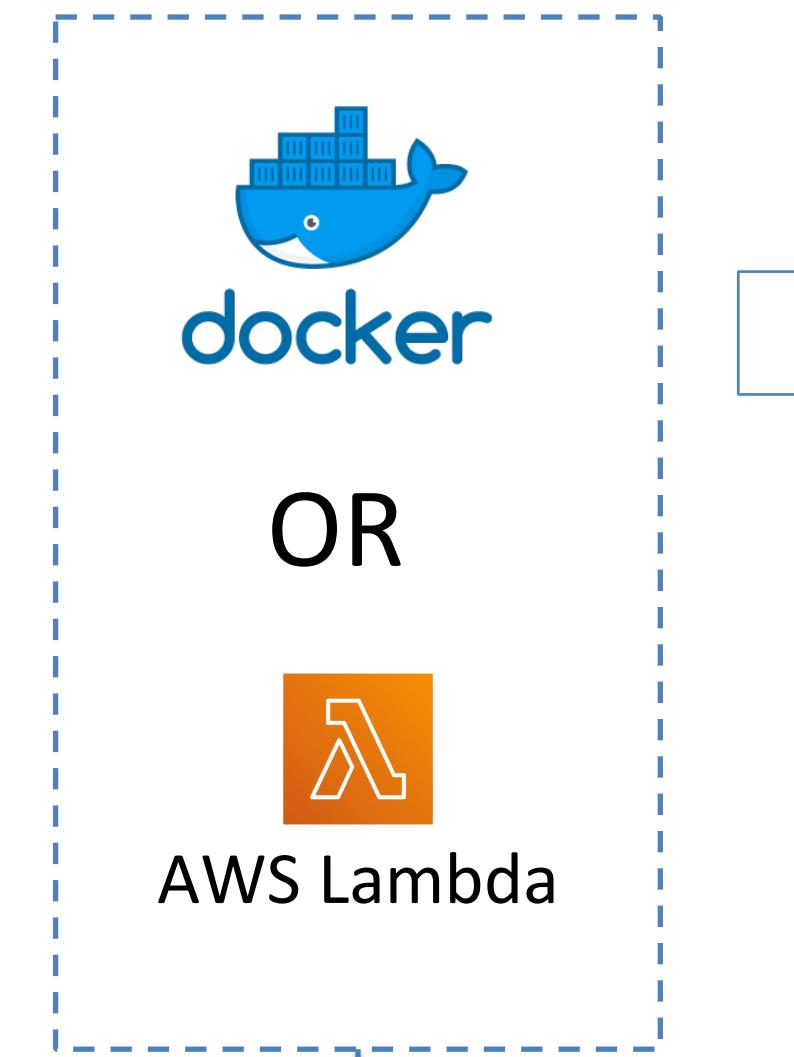
# Extract



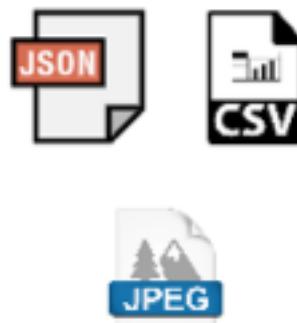
S3 - Extract



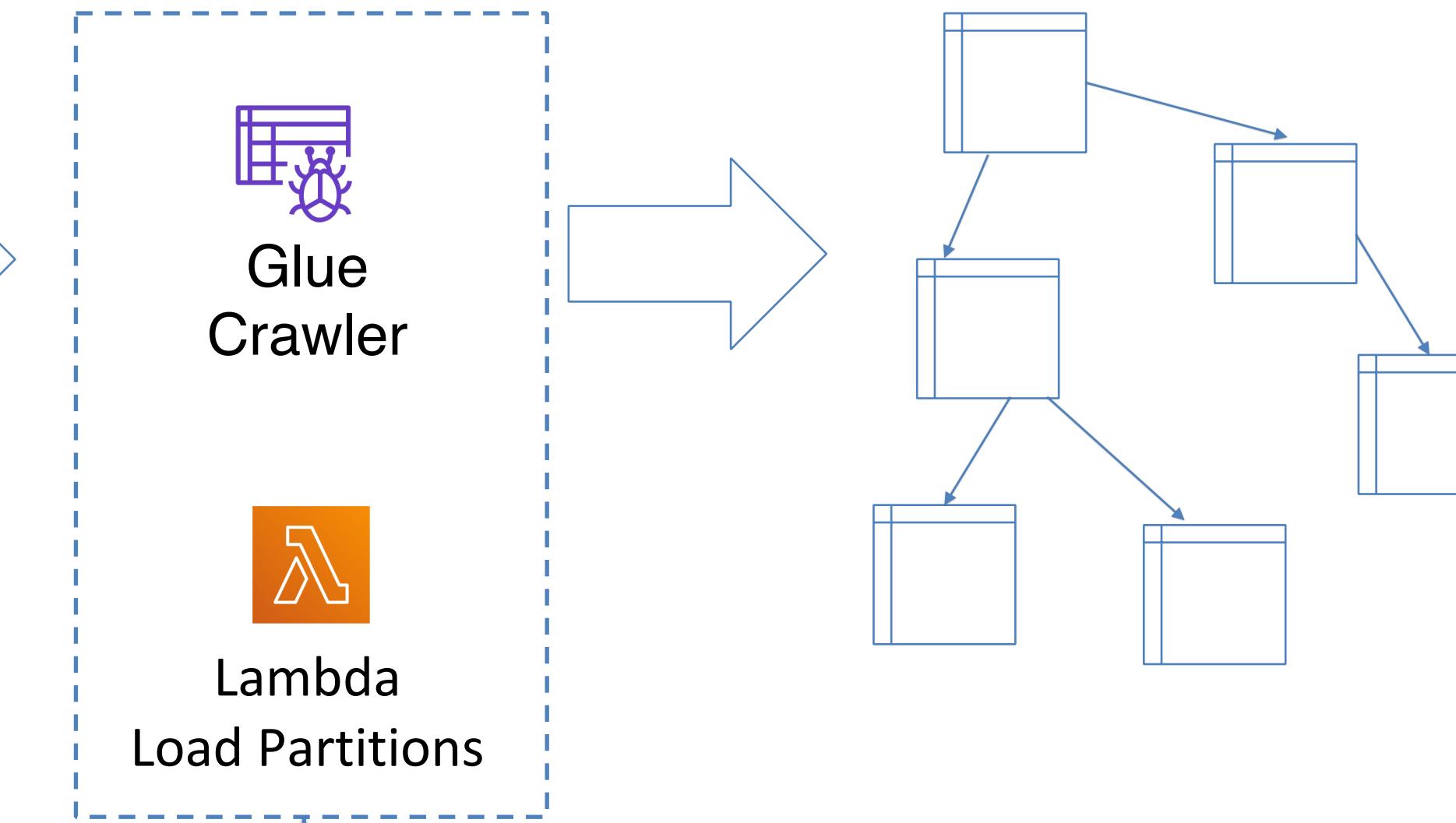
# Transform



S3 - Structured



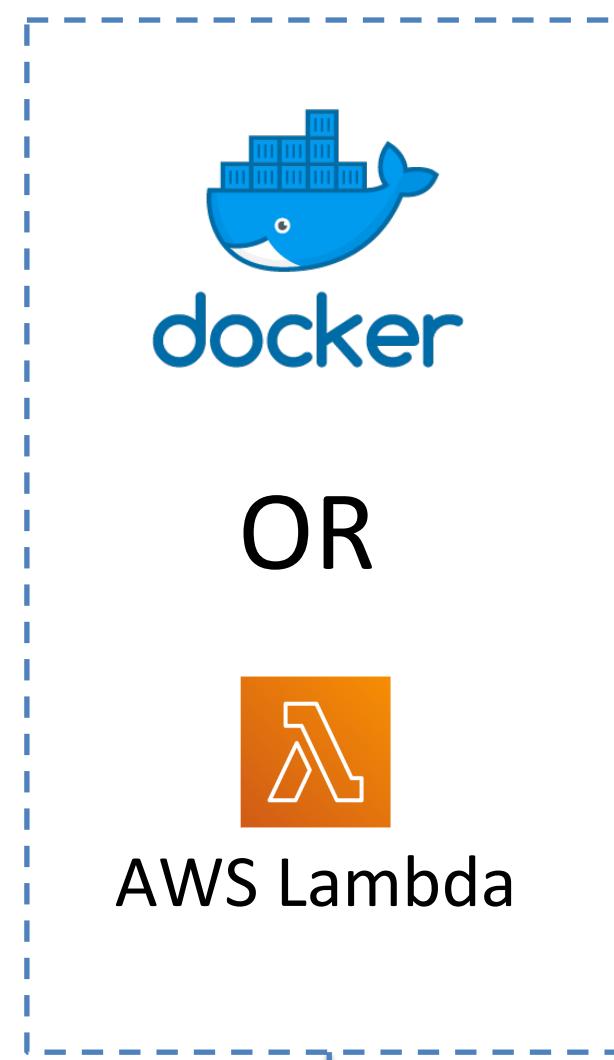
# Load



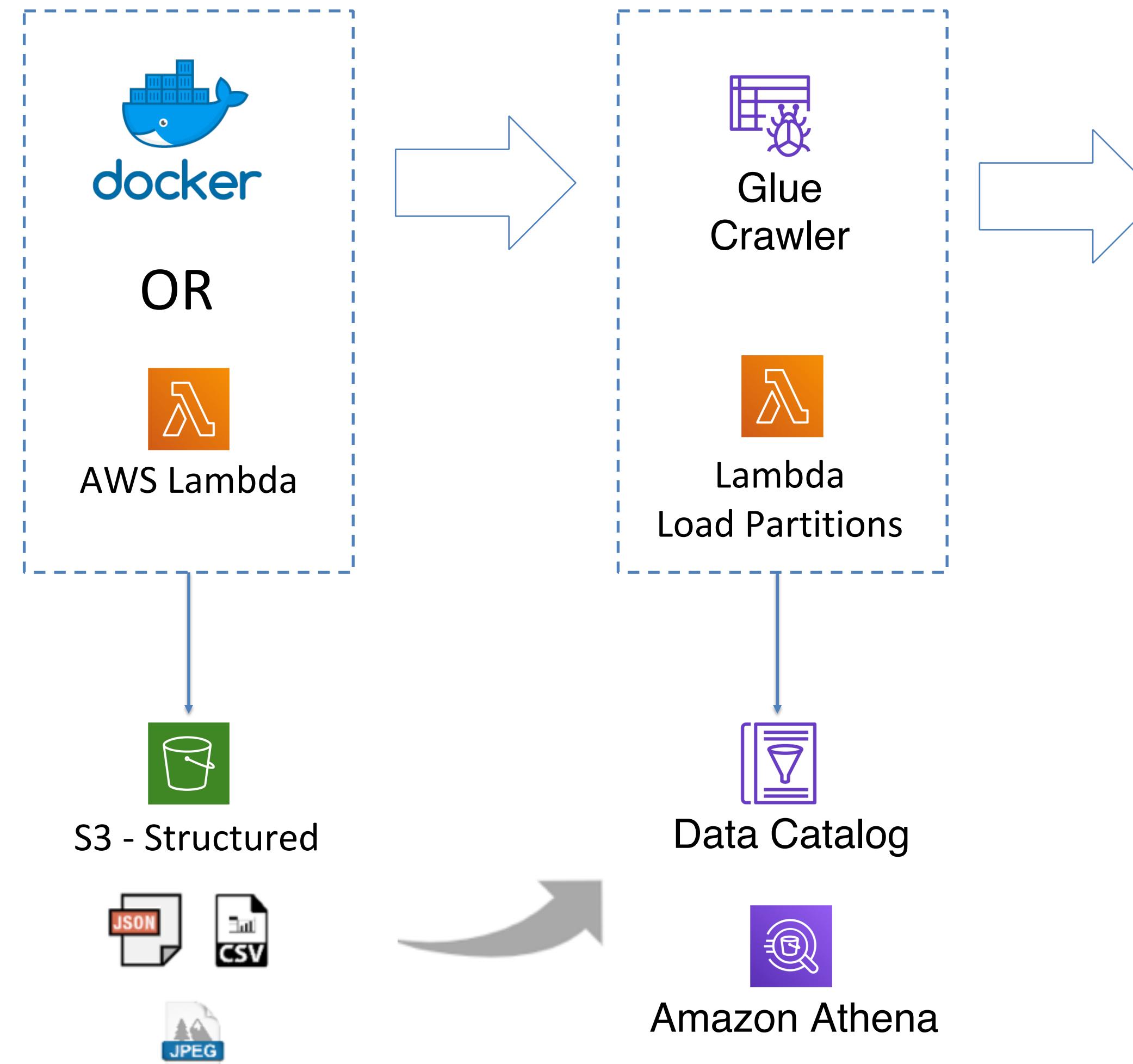
Data Catalog



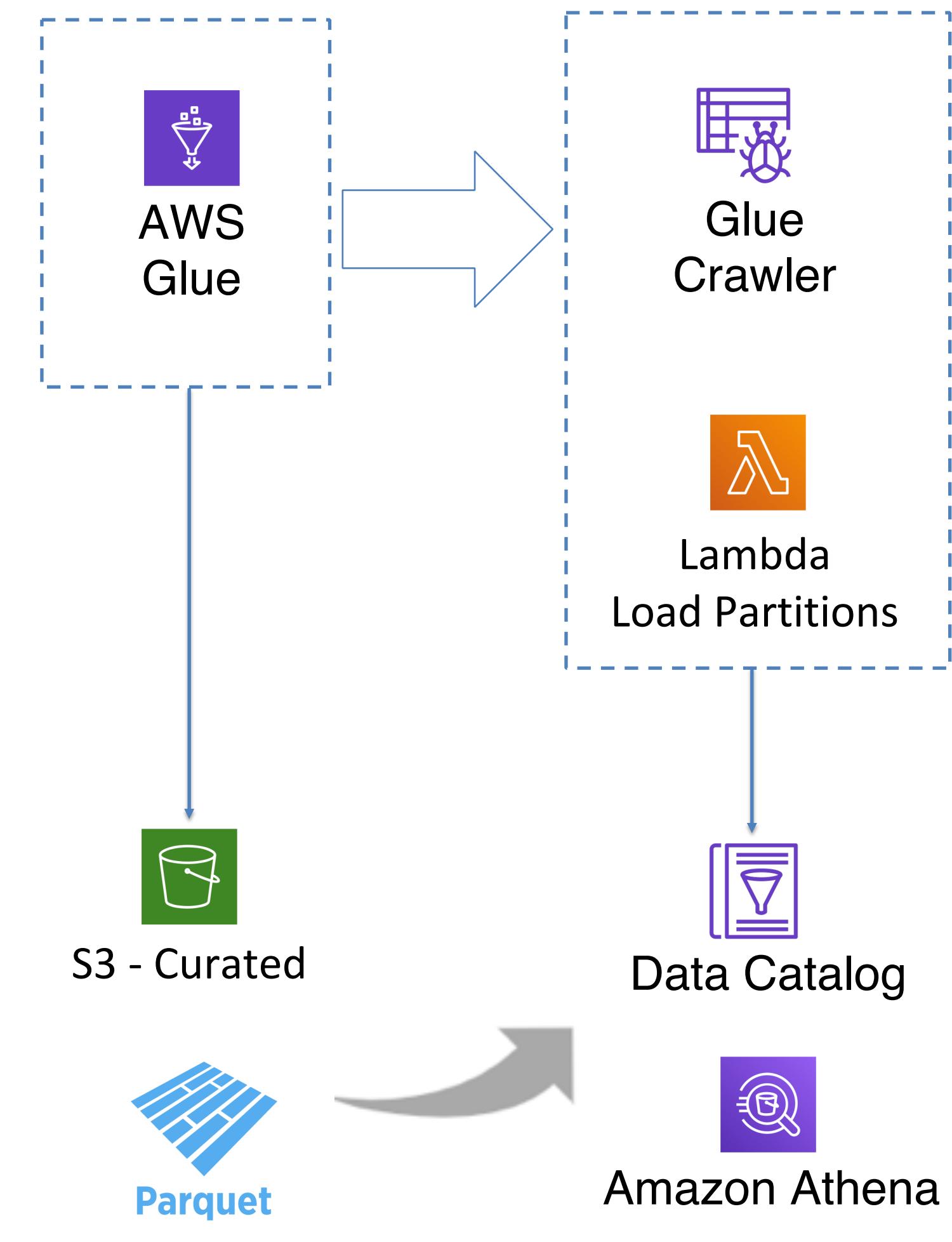
# Extract



# Transform



# Load





**Don't put extract in the same s3 bucket  
as the other stages it makes it hard to  
set different storage and backup  
options.**

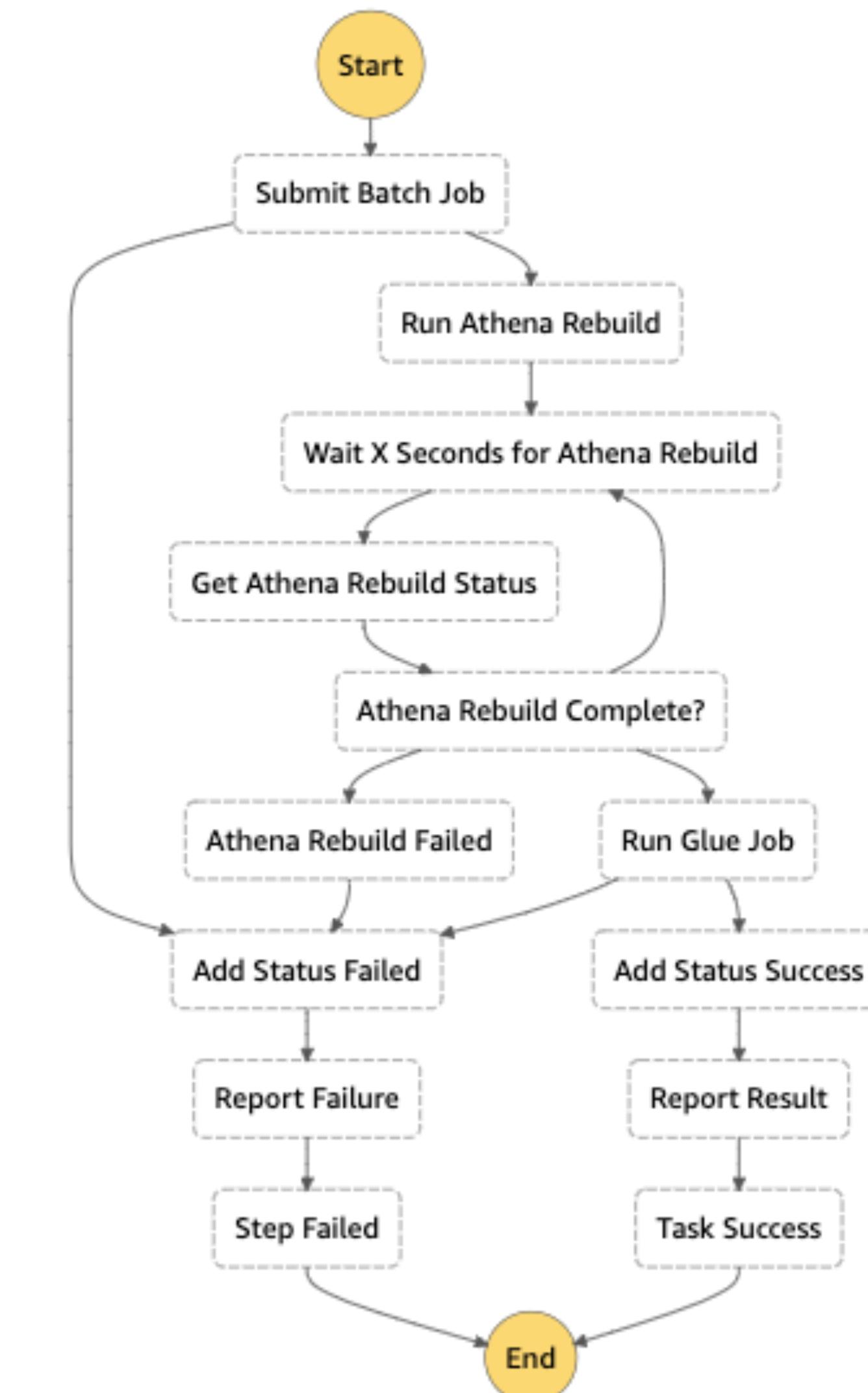
# Data Pipeline Tools Comparison

	Data Specific	Flexible
AWS	<b>AWS Data pipelines</b>	<b>AWS Step Functions</b>
Non AWS	<b>Apache Nifi</b> <b>Azkaban</b> <b>Oozie</b>	<b>Apache Airflow</b> <b>Luigi</b>

# AWS Step Functions

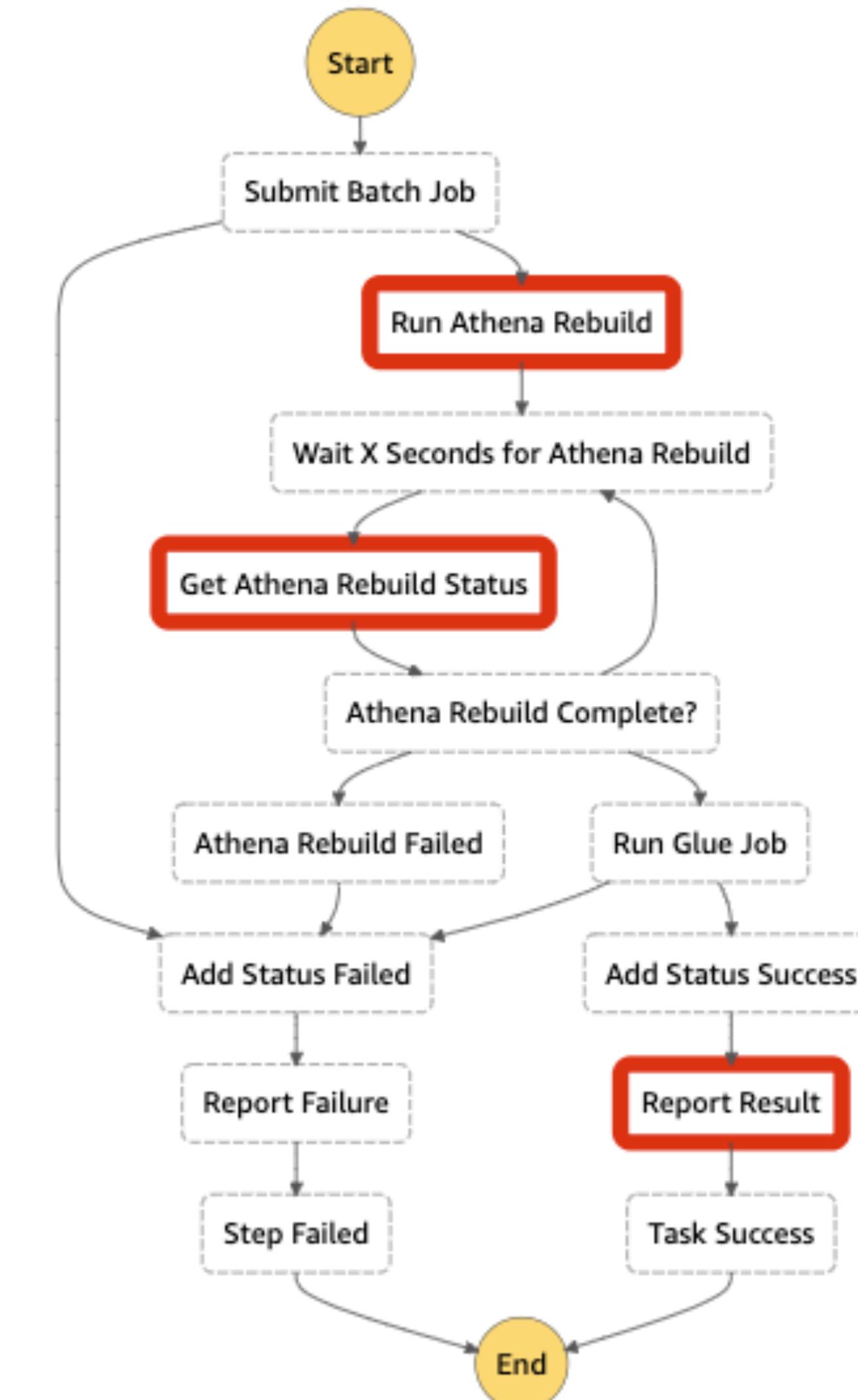
- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel

## State machine definition



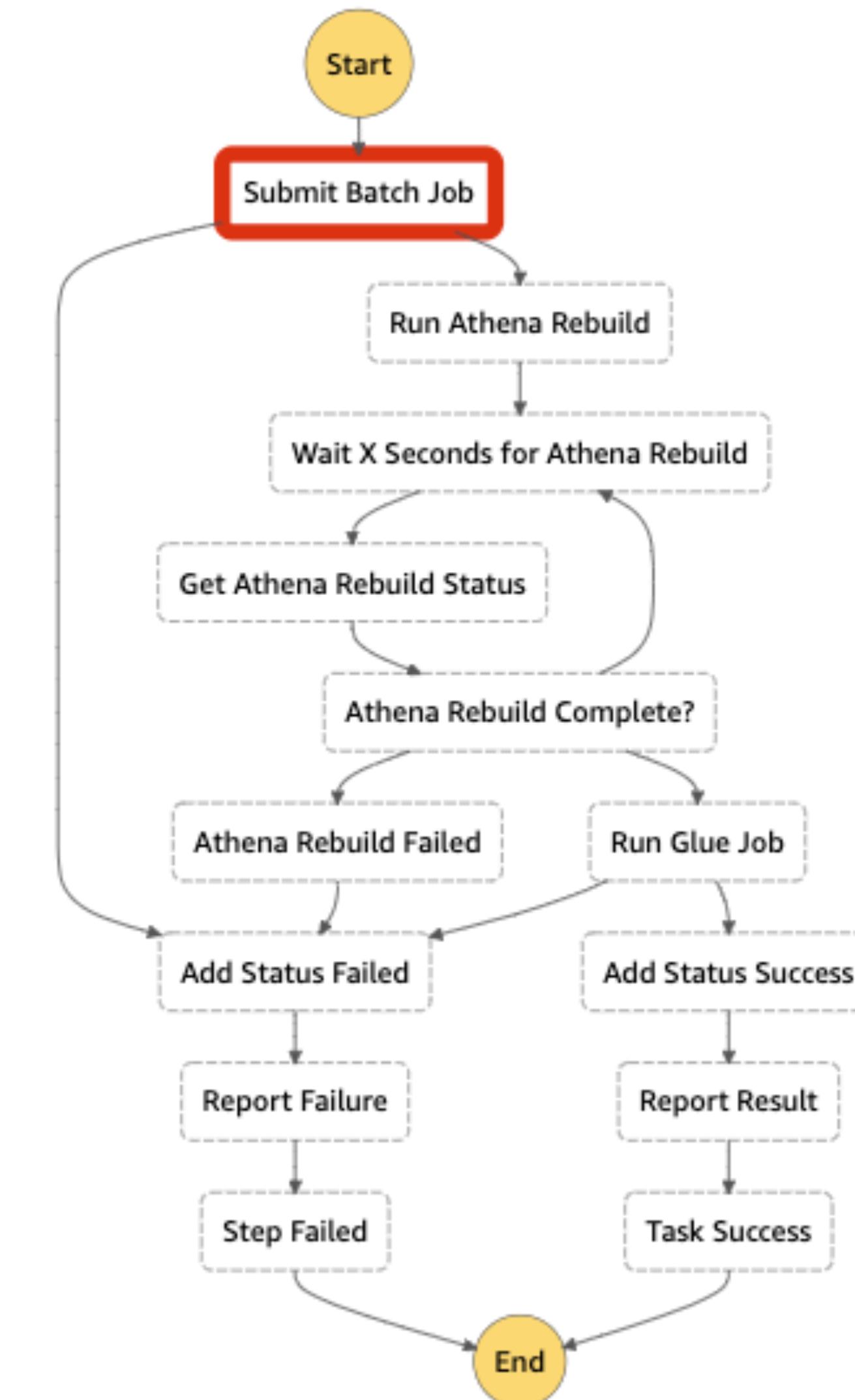
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



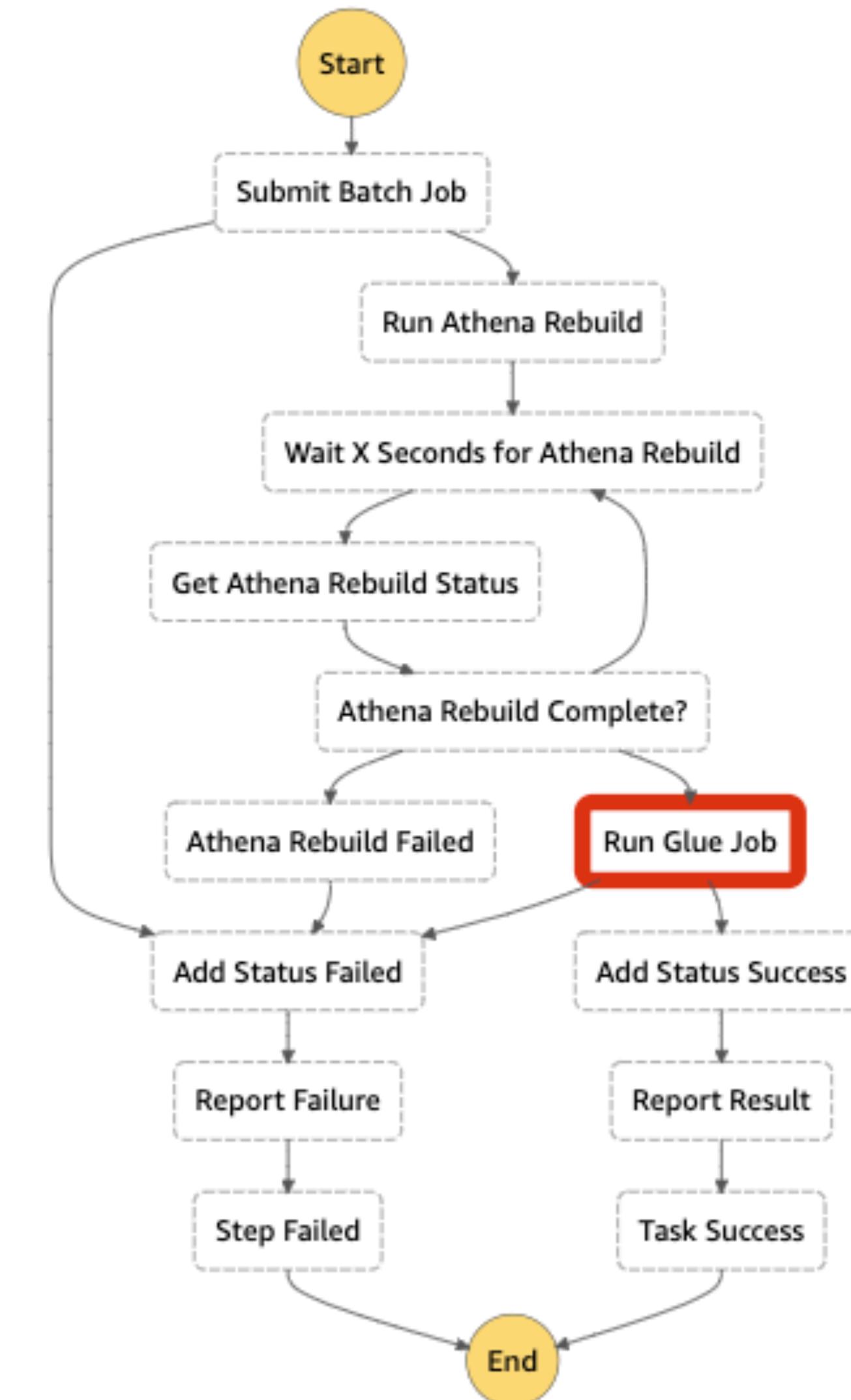
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



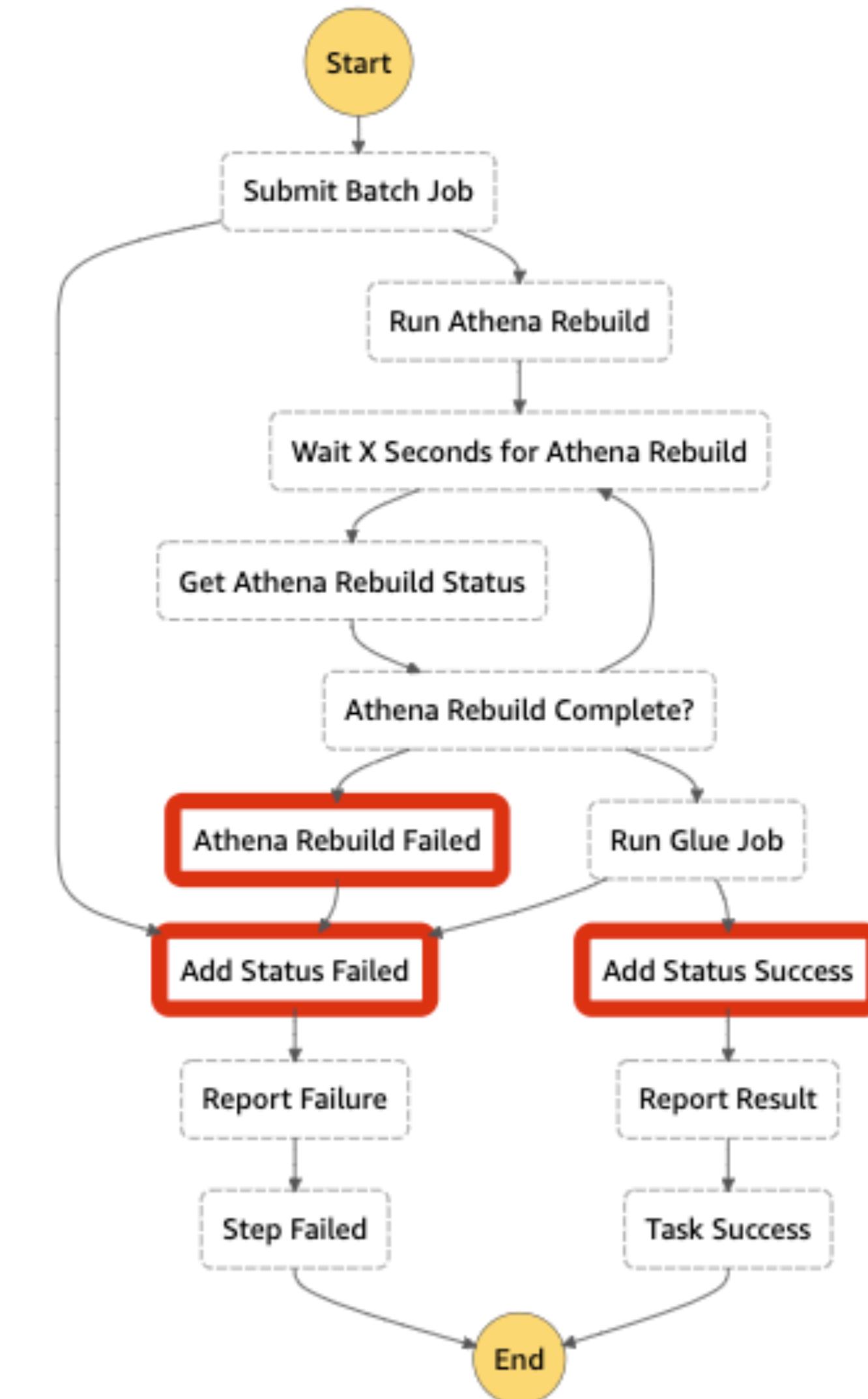
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



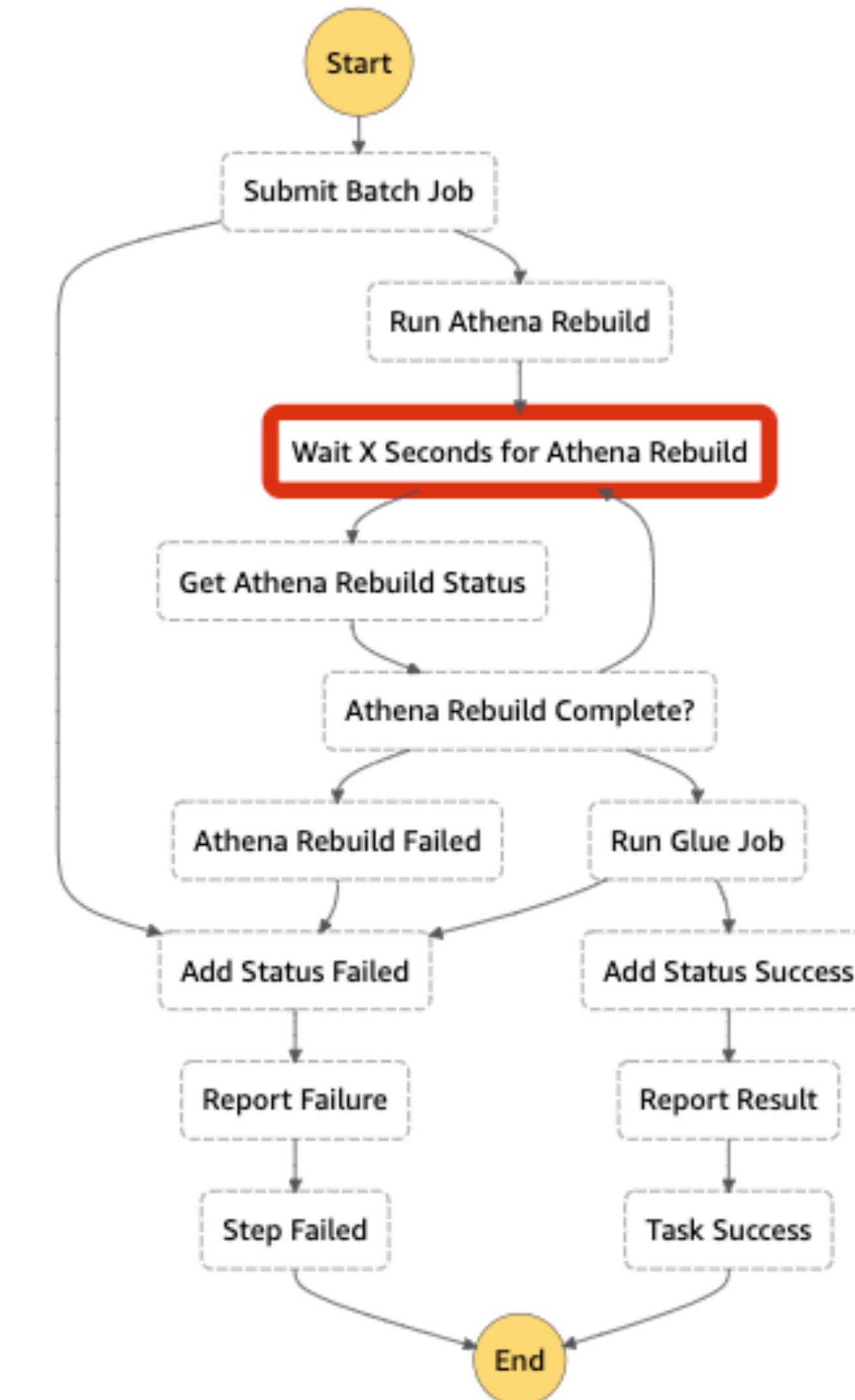
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



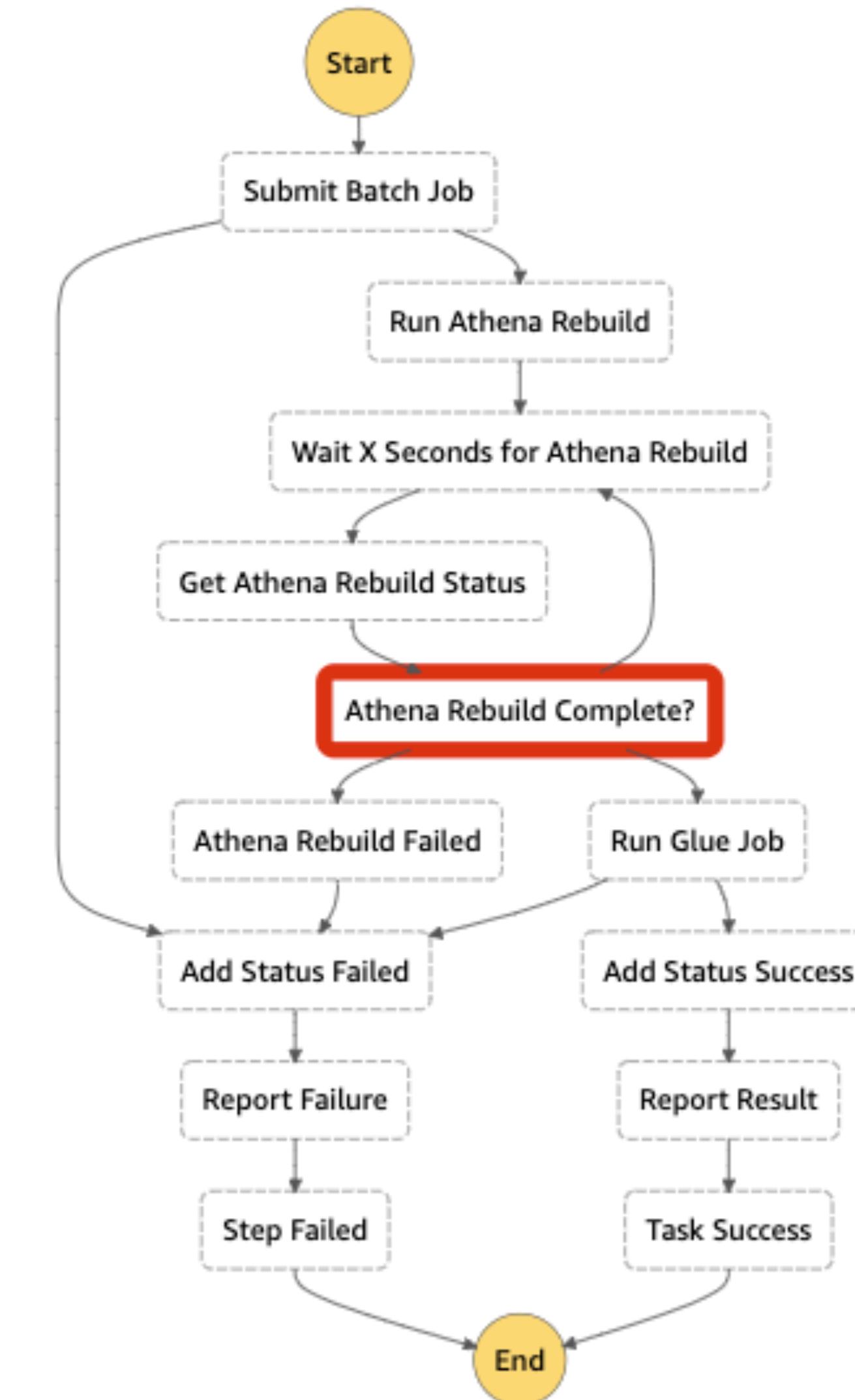
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



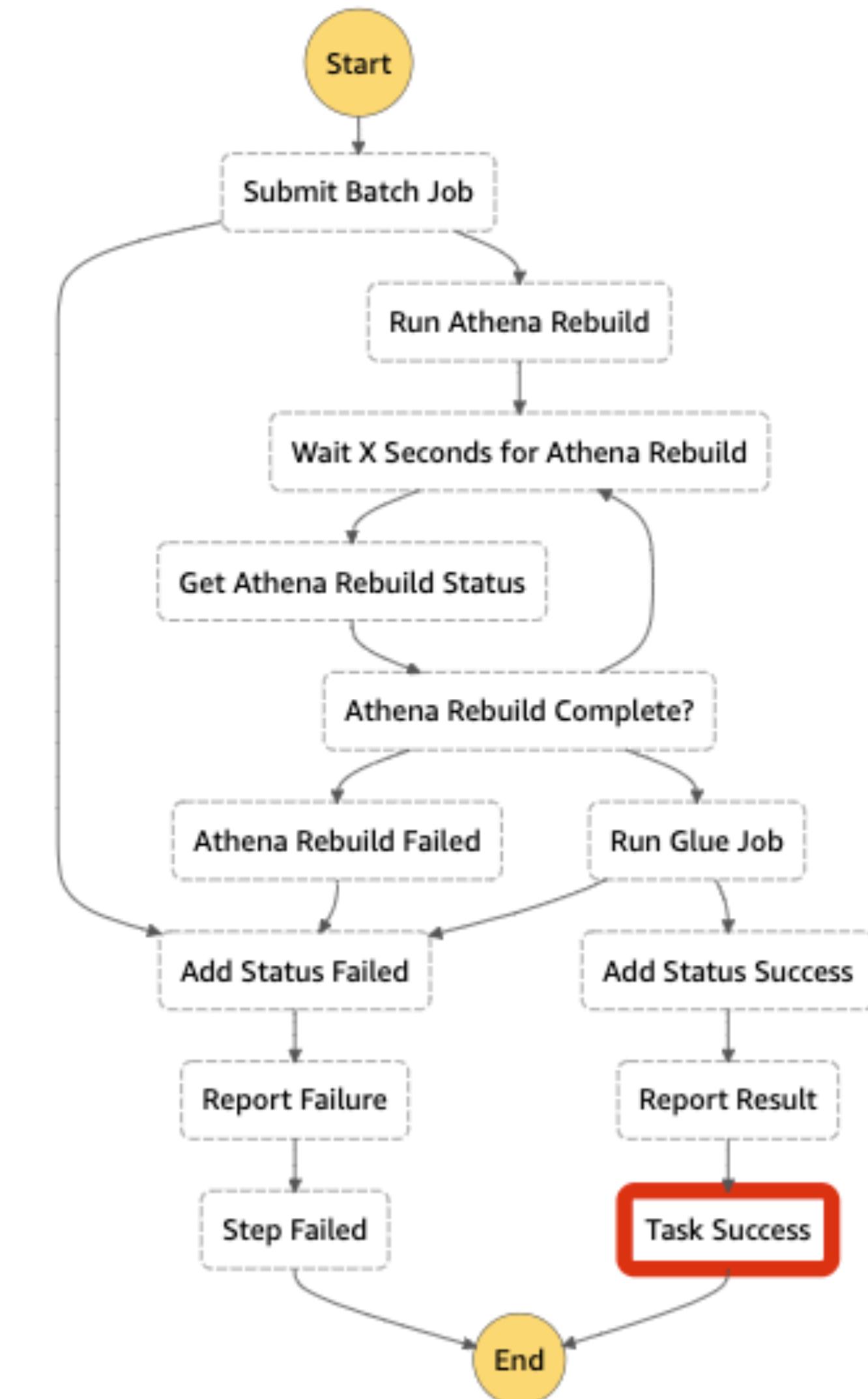
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



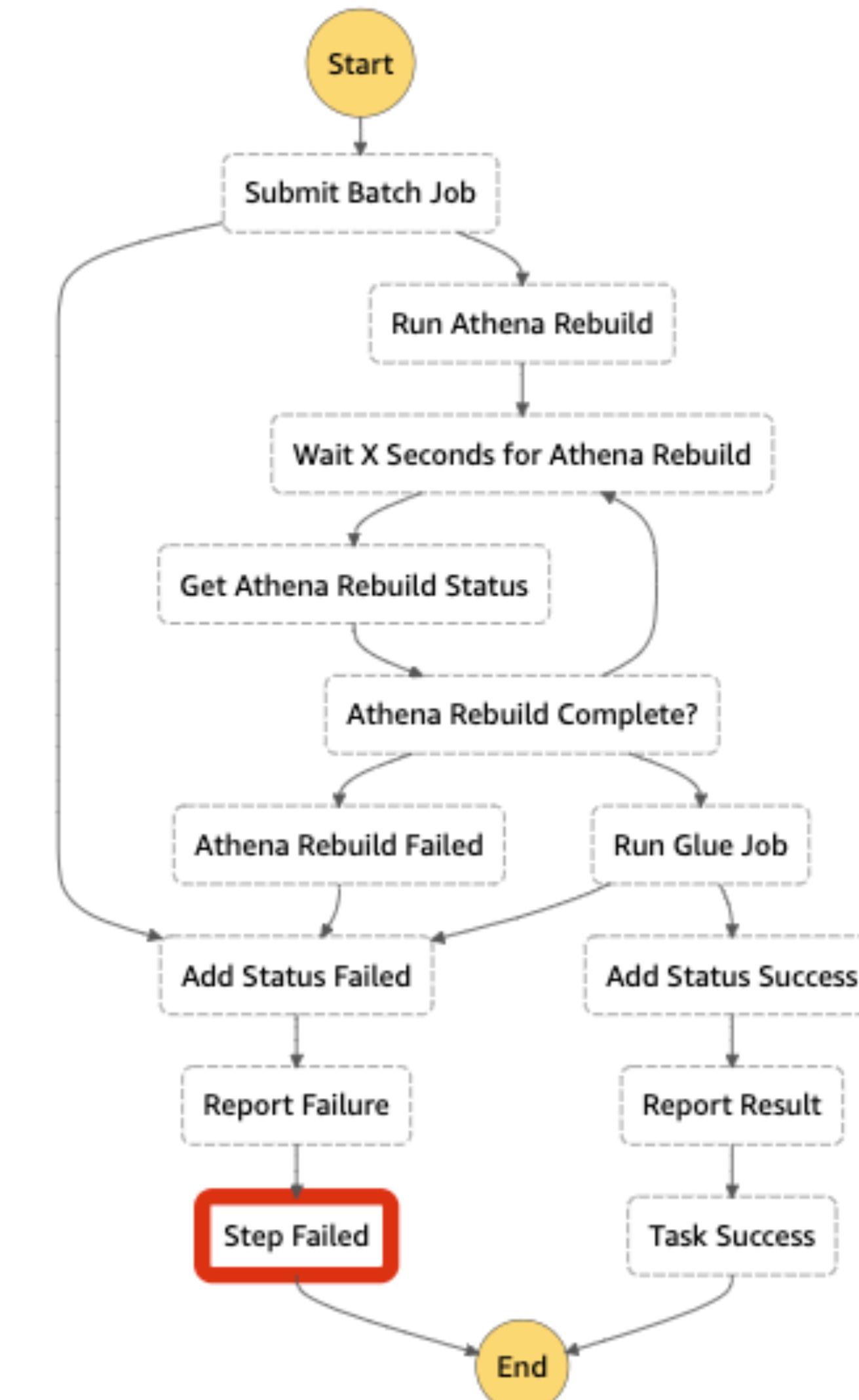
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



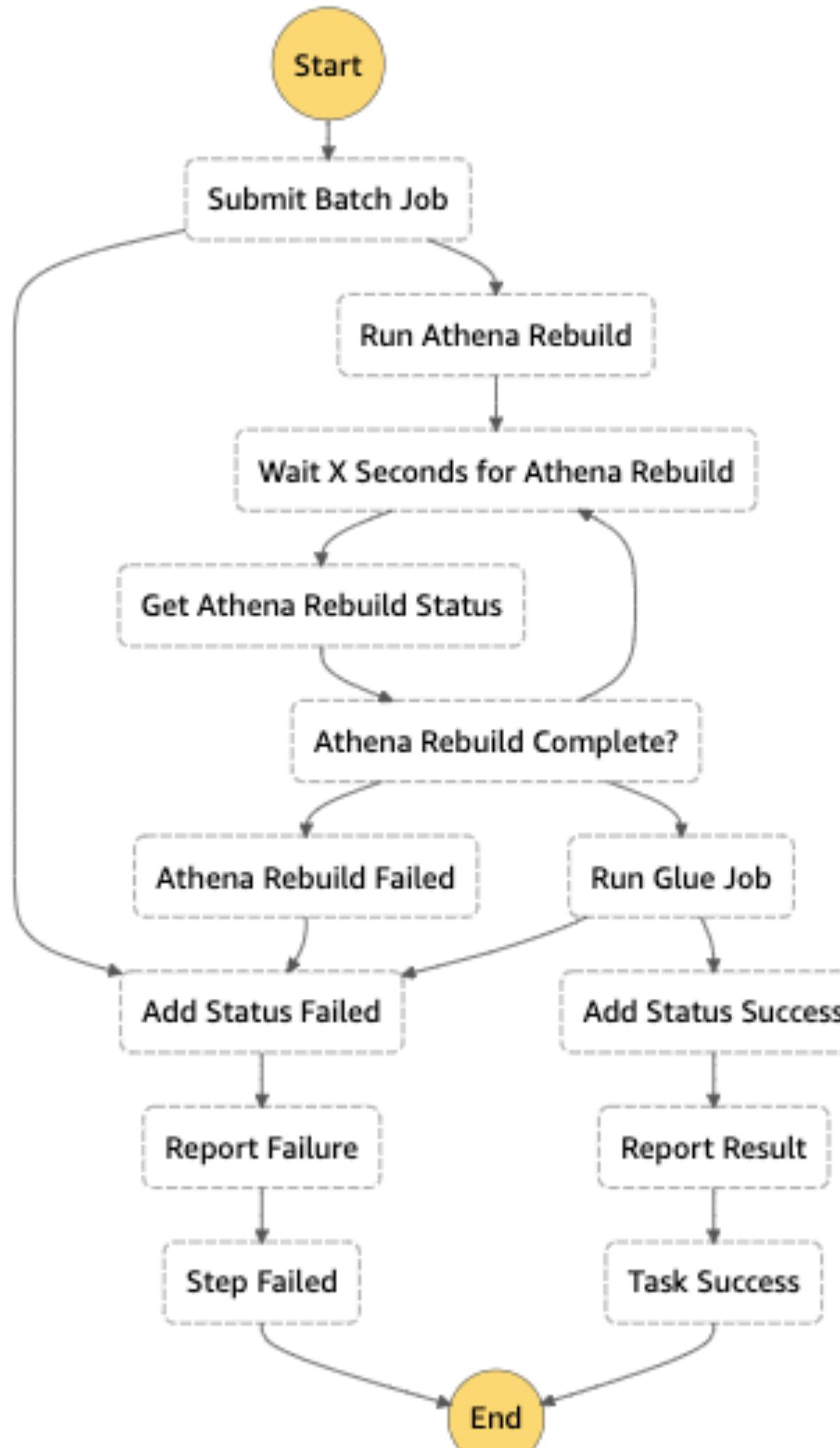
## State machine definition

- Tasks - A Task state represents a single unit of work performed by a state machine.
  - Invoke an AWS Lambda function.
  - Run an AWS Batch job and then perform different actions based on the results.
  - Insert or get an item from Amazon DynamoDB.
  - Run an Amazon Elastic Container Service (Amazon ECS) task and wait for it to complete.
  - Publish a topic in Amazon Simple Notification Service (Amazon SNS).
  - Send a message in Amazon Simple Queue Service (Amazon SQS).
  - Manage a job for AWS Glue or Amazon SageMaker.
- Pass - Passes input to output without doing any work.
- Wait - Waits a x seconds before continuing
- Succeed - Terminates successfully
- Fail - Terminates state machine unsuccessfully
- Choice - Applies branching logic
- Parallel - Performs tasks in parallel



# Step Function Transitions

## State machine definition

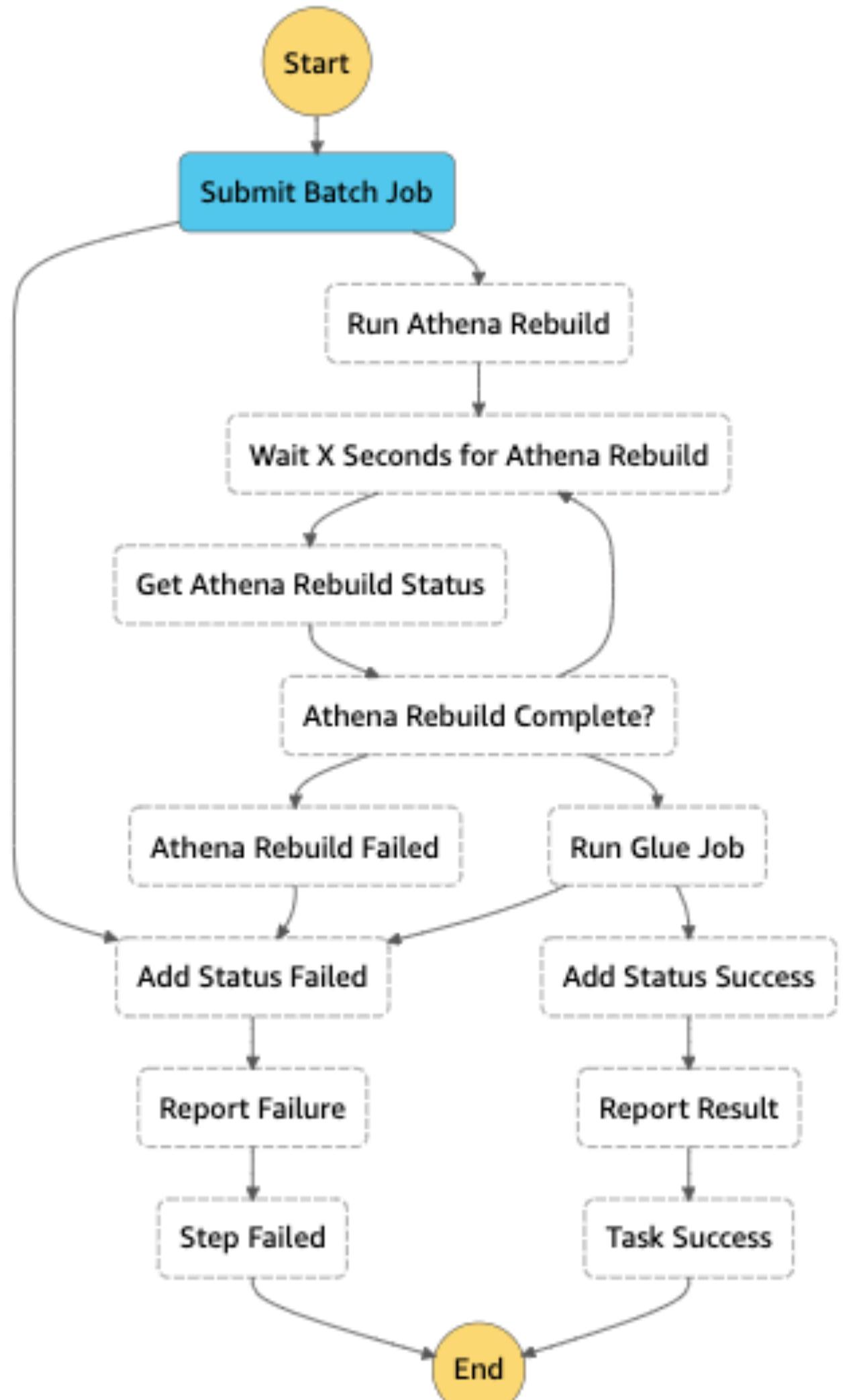


Extract/Transform

Load

Report

## State machine definition

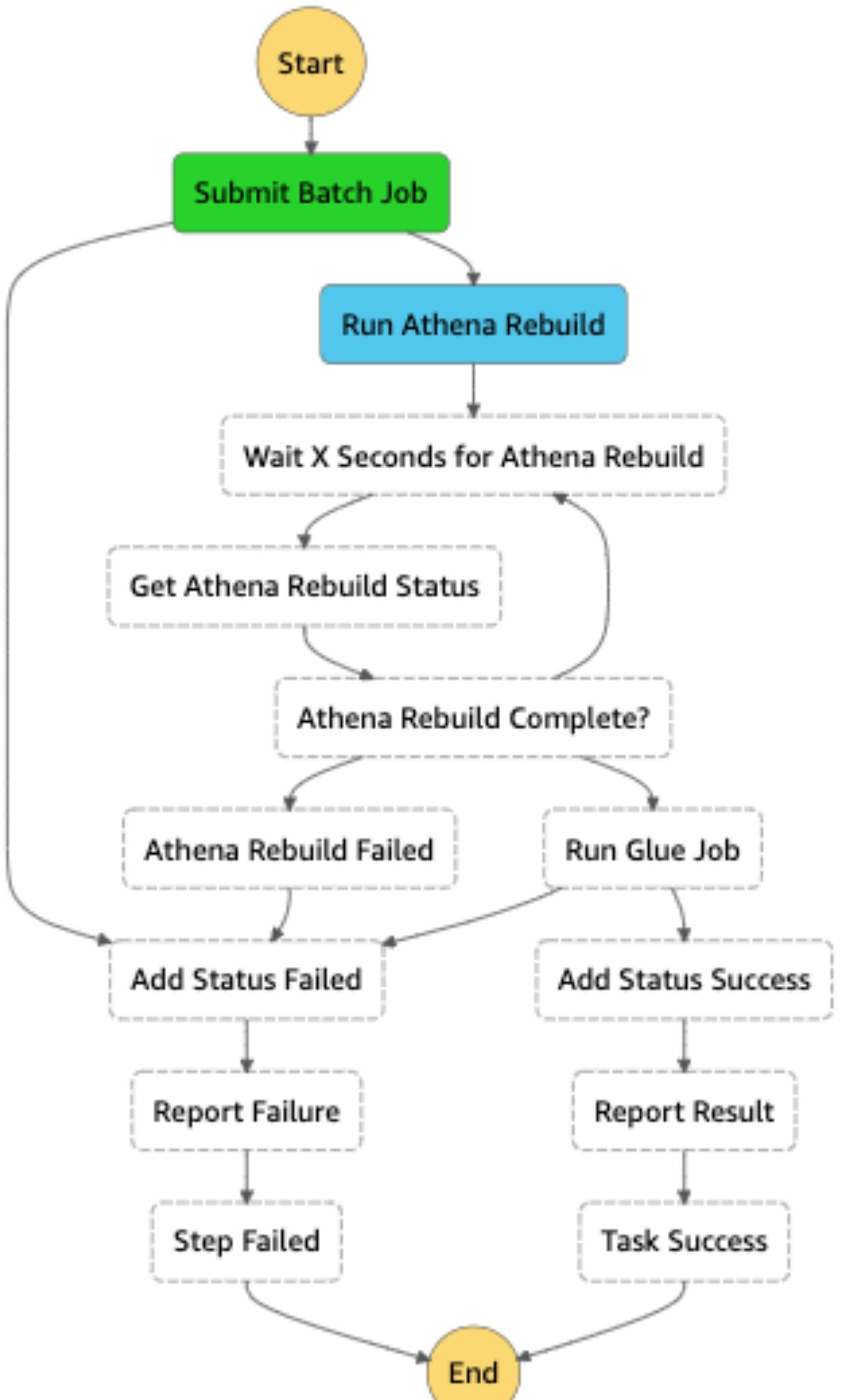


Extract/Transform

Load

Report

## State machine definition

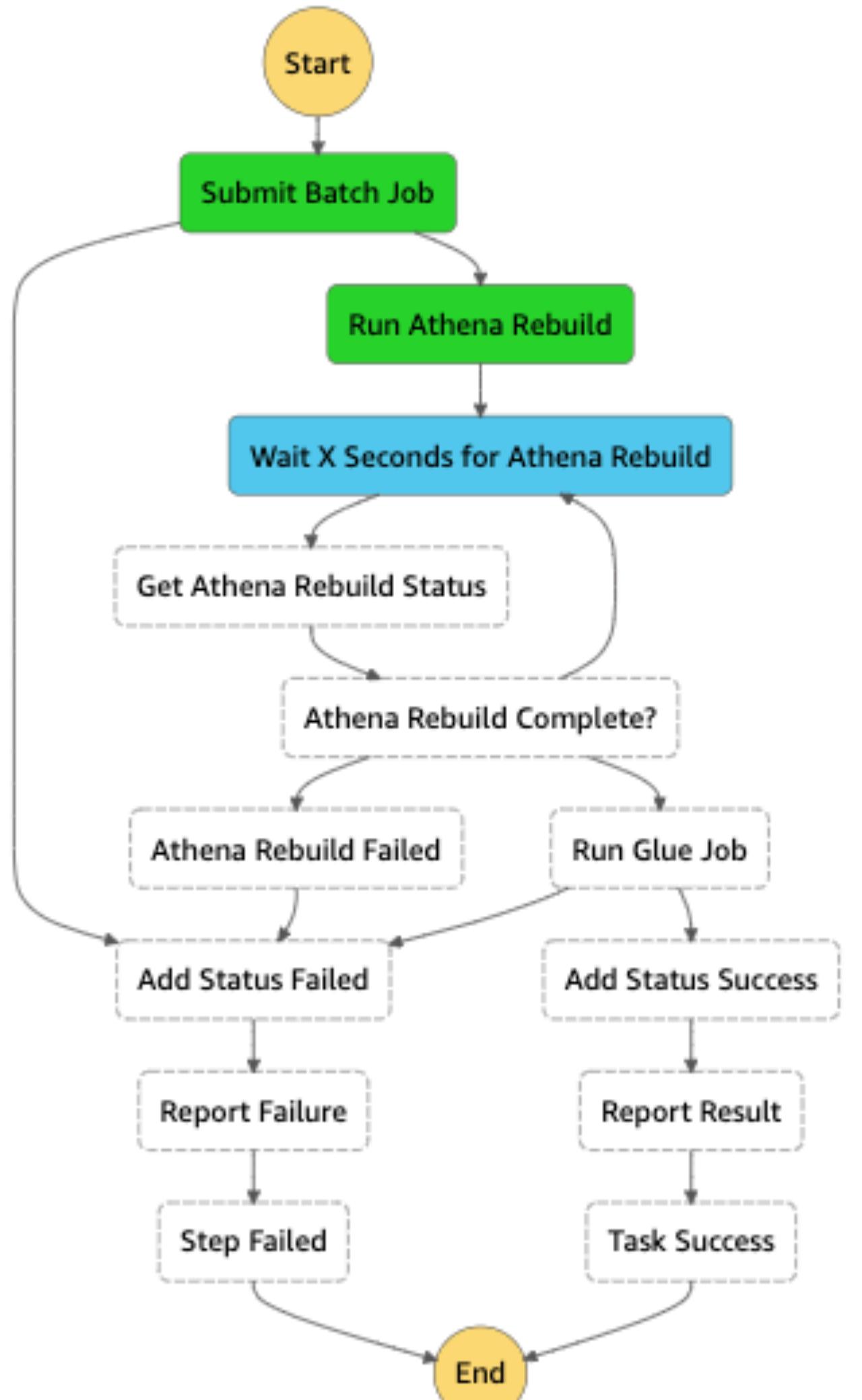


Extract/Transform

Load

Report

## State machine definition

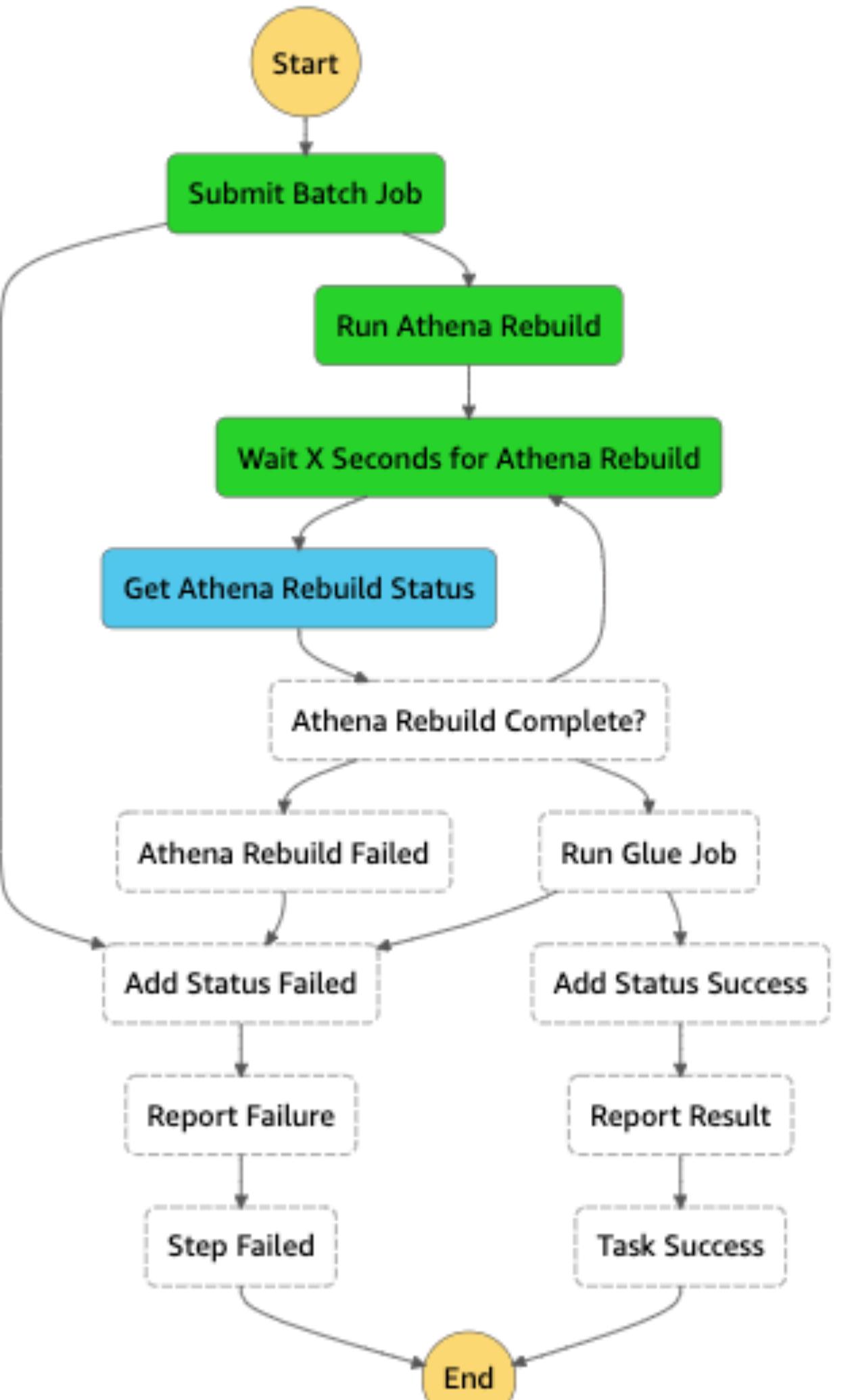


Extract/Transform

Load

Report

## State machine definition

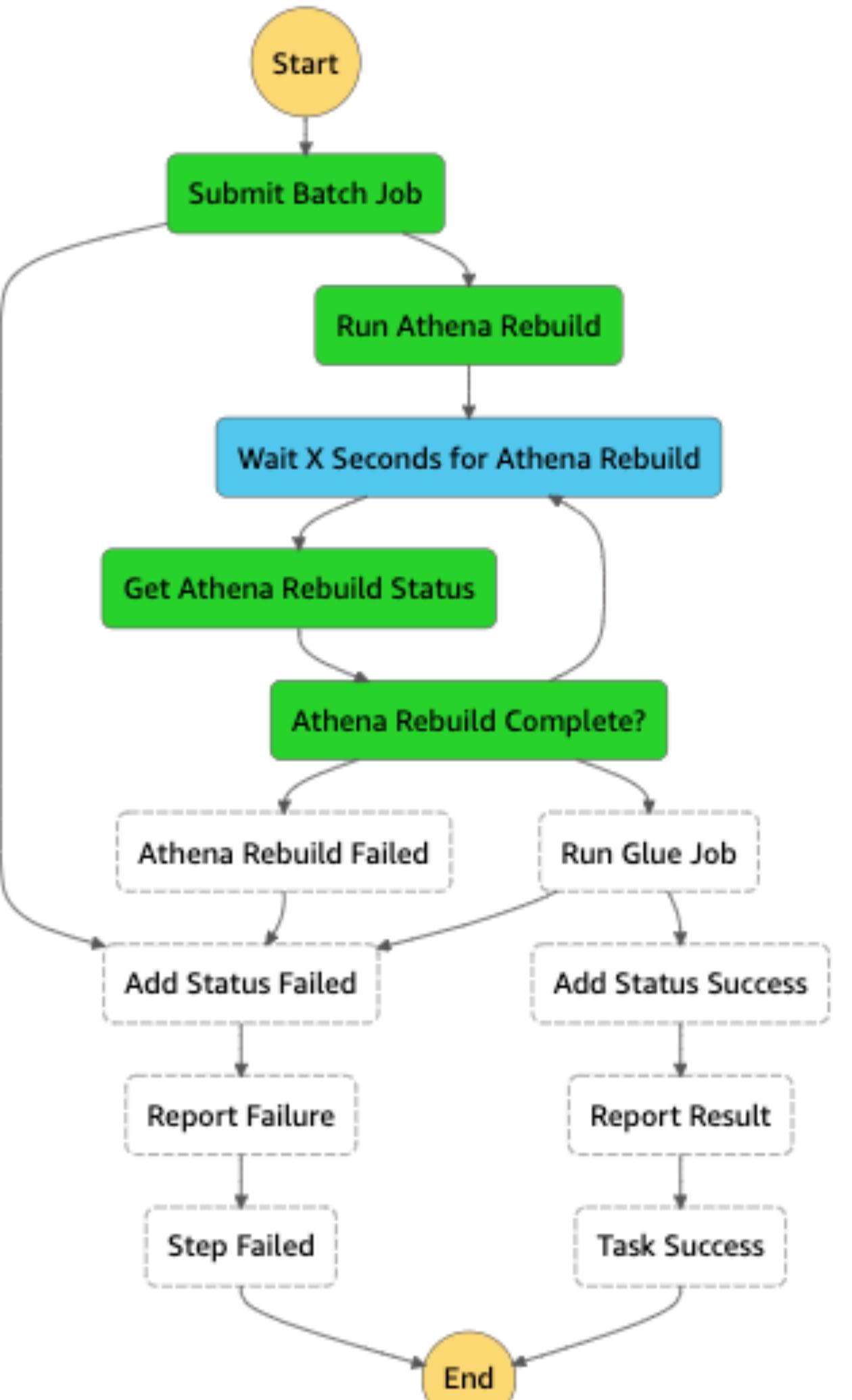


Extract/Transform

Load

Report

## State machine definition

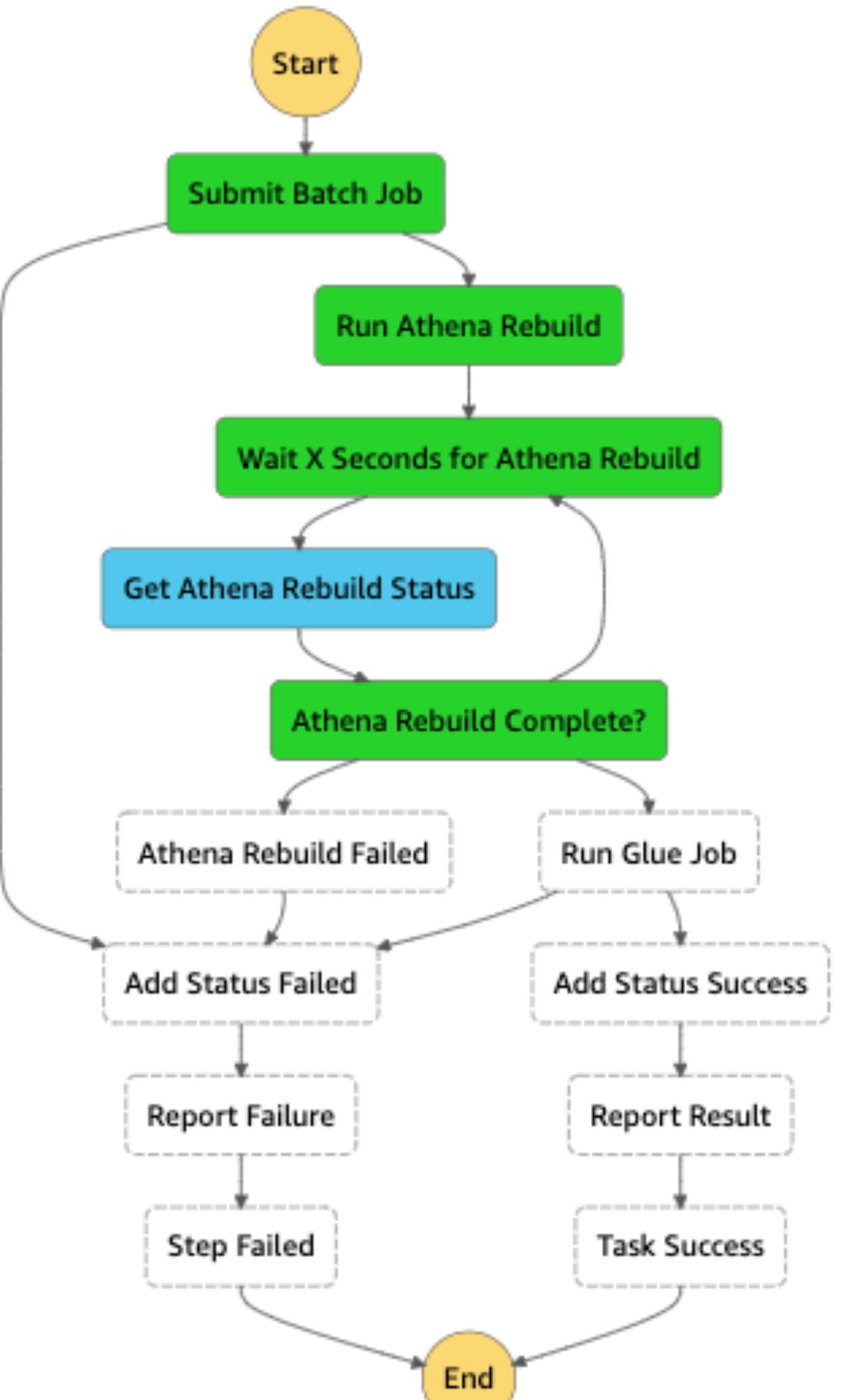


Extract/Transform

Load

Report

## State machine definition

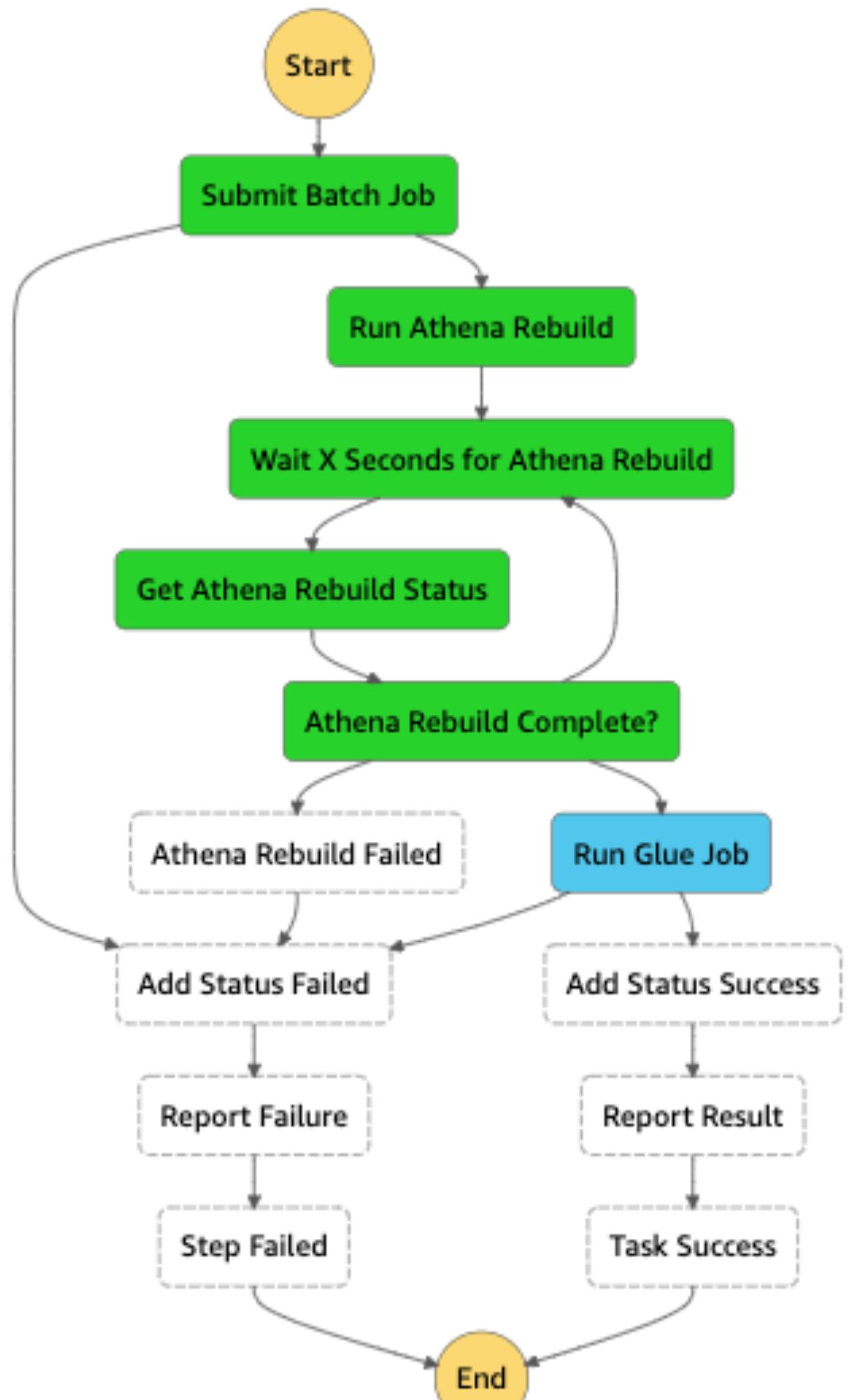


Extract/Transform

Load

Report

## State machine definition

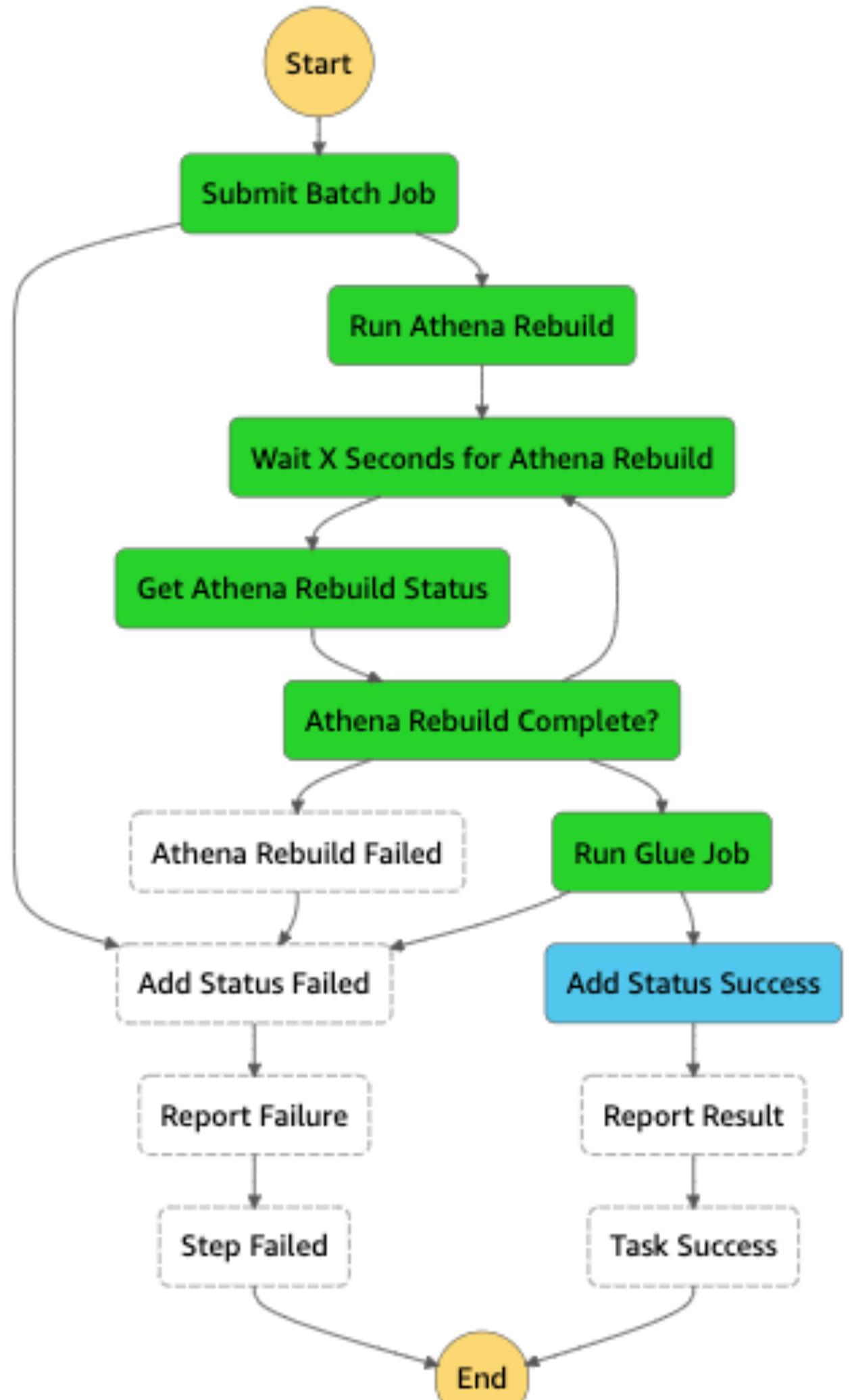


Extract/Transform

Load

Report

## State machine definition

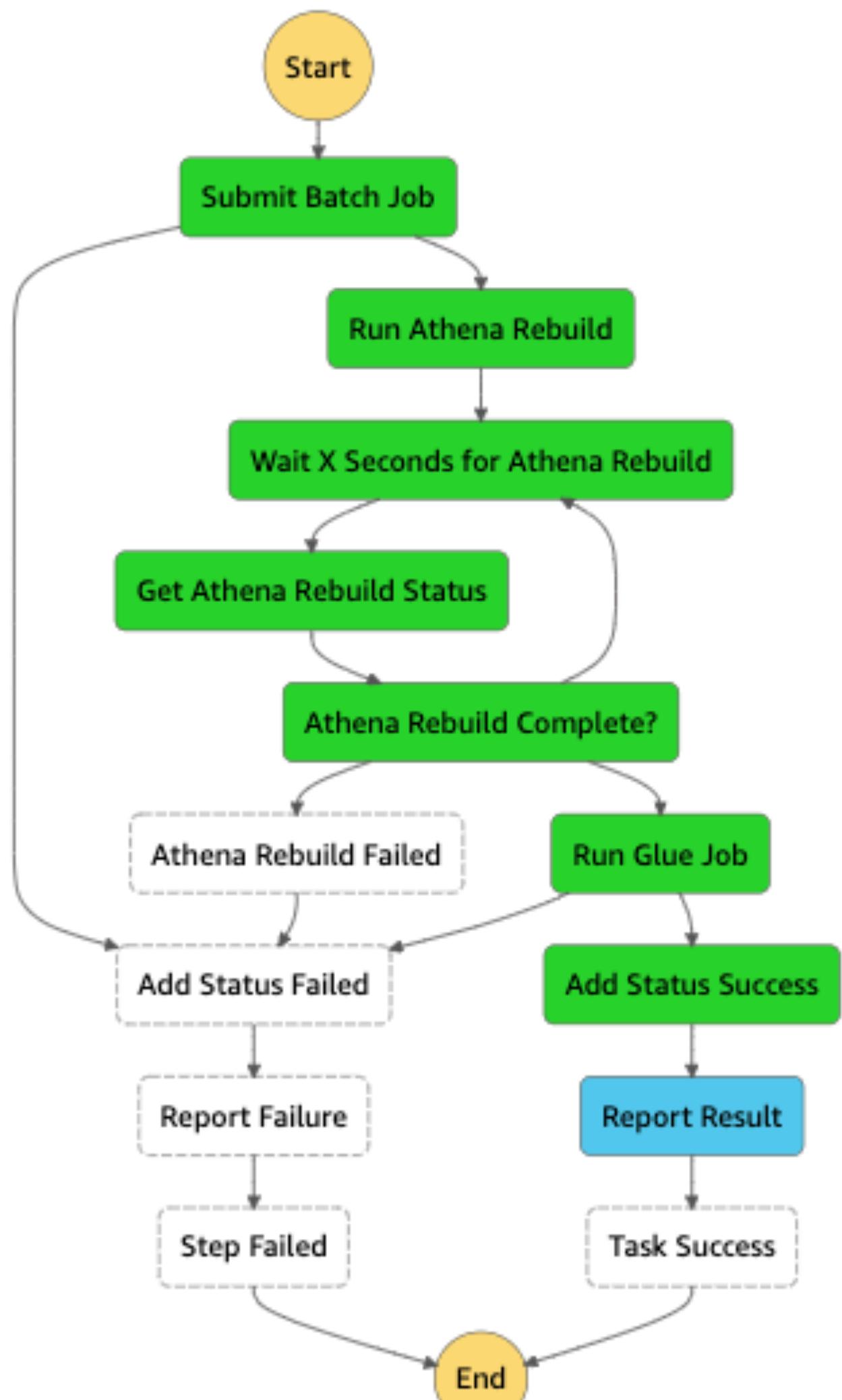


Extract/Transform

Load

Report

## State machine definition

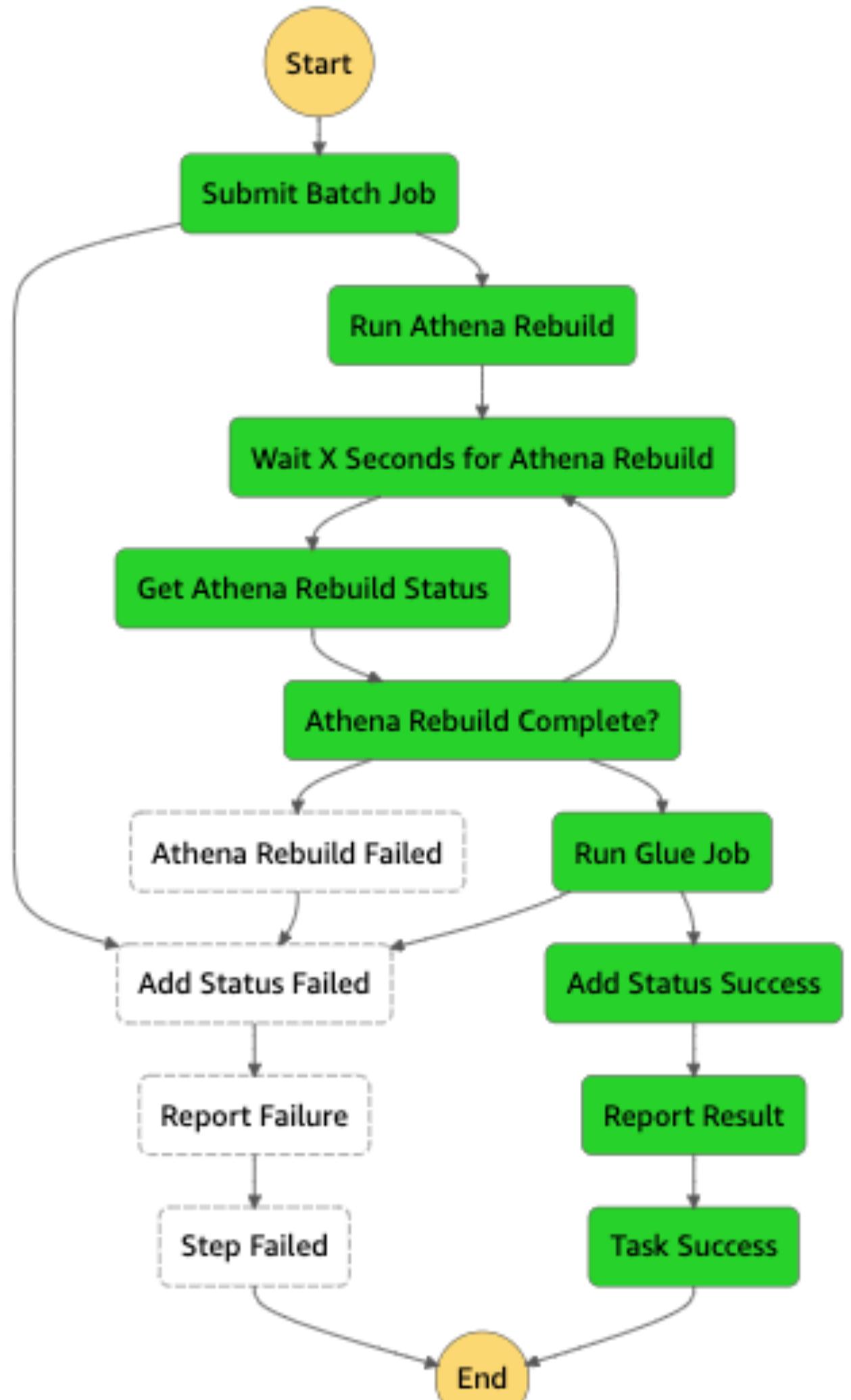


Extract/Transform

Load

Report

## State machine definition

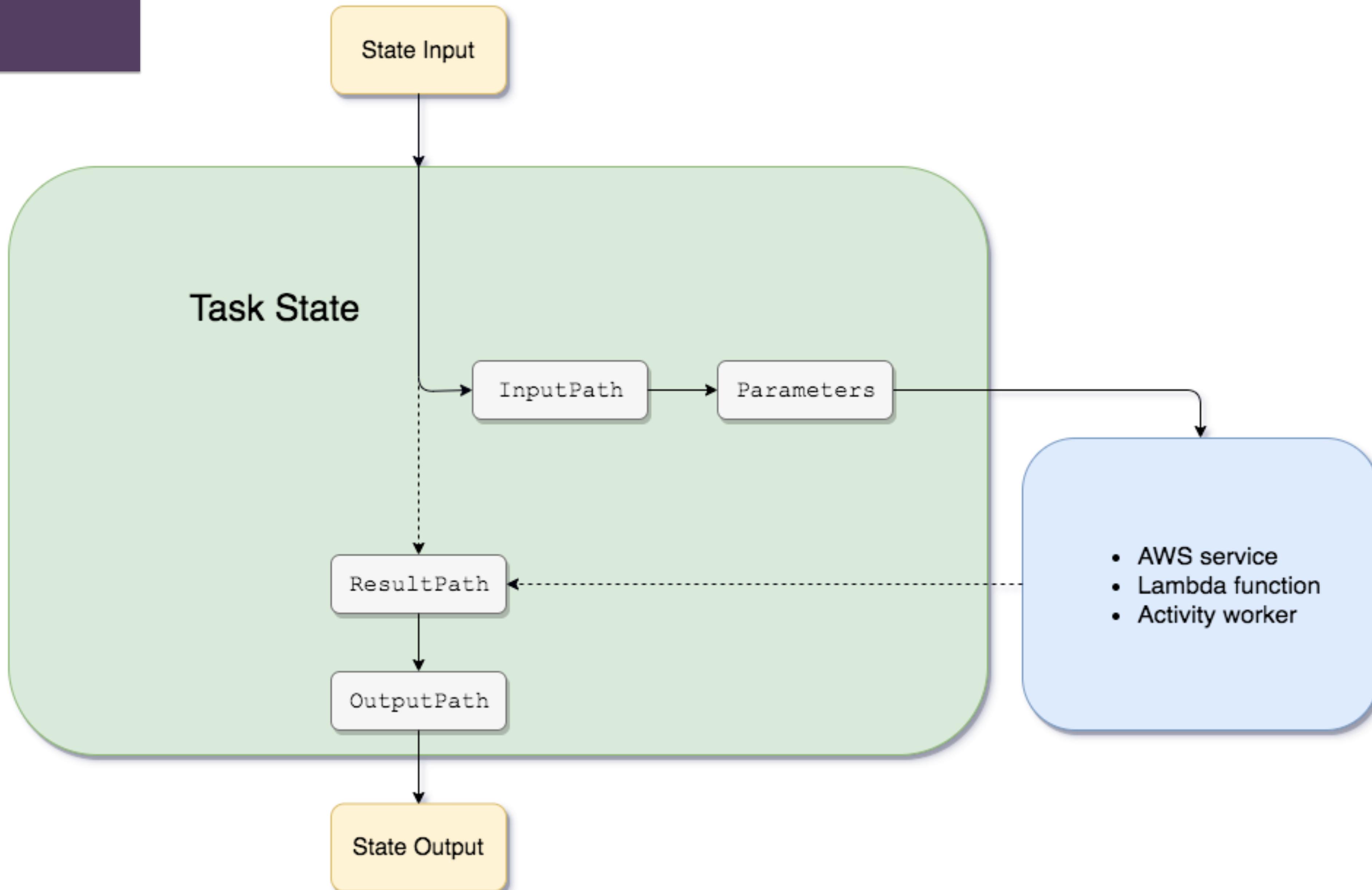


Extract/Transform

Load

Report

# State Input



# Step Function Definition

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",  
        "JobQueue.$": "$.batchJobQueue",  
        "JobDefinition.$": "$.batchJobDef",  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging"  
                }  
            ],  
            "Command.$": "$.batchJobCmd"  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

Task Definition

```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": "{$\"stateMachineLink\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        \"$\"stateMachineExecutionID\":  
        \"0190715234500\", \"$\"stepfunctionName\":  
        \"\", \"$\"countyName\": \"Example\",  
        \"$\"state\": \"ca\", \"$\"dataType\": \"parcel\", \"$\"start-time\":  
        \"20190715234500\"}",  
    "batchJobCmd": []  
}
```

Job Input

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",   
        "JobQueue.$": "$.batchJobQueue",  
        "JobDefinition.$": "$.batchJobDef",  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging"  
                }  
            ],  
            "Command.$": "$.batchJobCmd"  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

## Task Definition

```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": {"\\"stateMachineLink\\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        "\\"stateMachineExecutionID\\":  
            \"0190715234500\", \\"stepfunctionName\\":  
                \"\", \\"countyName\\\": \"Example\",  
                \\"state\\\": \"ca\", \\"dataType\\\": \"parcel\", \\"start-time\\\":  
                    \"20190715234500\""},  
    "batchJobCmd": []  
}
```

## Job Input

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",  
        "JobQueue.$": "$.batchJobQueue", ←  
        "JobDefinition.$": "$.batchJobDef",  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging"  
                }  
            ],  
            "Command.$": "$.batchJobCmd"  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

## Task Definition

```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": "{$\"stateMachineLink\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        \"$stateMachineExecutionID\":  
        \"0190715234500\", \"$stepfunctionName\":  
        \"\", \"$countyName\": \"Example\",  
        \"$state\": \"ca\", \"$dataType\": \"parcel\", \"$start-time\":  
        \"20190715234500\"}",  
    "batchJobCmd": []  
}
```

## Job Input

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",  
        "JobQueue.$": "$.batchJobQueue",  
        "JobDefinition.$": "$.batchJobDef", ←  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging"  
                }  
            ],  
            "Command.$": "$.batchJobCmd"  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

## Task Definition

```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": "{$\"stateMachineLink\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        \"$\"stateMachineExecutionID\":  
        \"0190715234500\", \"$\"stepfunctionName\":  
        \"\", \"$\"countyName\": \"Example\",  
        \"$\"state\": \"ca\", \"$\"dataType\": \"parcel\", \"$\"start-time\":  
        \"20190715234500\"}",  
    "batchJobCmd": []  
}
```

## Job Input

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",  
        "JobQueue.$": "$.batchJobQueue",  
        "JobDefinition.$": "$.batchJobDef",  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging" ←  
                }  
            ],  
            "Command.$": "$.batchJobCmd"  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

## Task Definition

```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": "{\"stateMachineLink\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        \"stateMachineExecutionID\":  
            \"0190715234500\", \"stepfunctionName\":  
            \"\", \"countyName\": \"Example\",  
            \"state\": \"ca\", \"dataType\": \"parcel\", \"start-time\":  
            \"20190715234500\"}",  
    "batchJobCmd": []  
}
```

## Job Input

```
"Submit Batch Job": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::batch:submitJob.sync",  
    "Parameters": {  
        "JobName.$": "$.jobName",  
        "JobQueue.$": "$.batchJobQueue",  
        "JobDefinition.$": "$.batchJobDef",  
        "ContainerOverrides": {  
            "Environment": [  
                {  
                    "Name": "logging",  
                    "Value.$": "$.logging"  
                }  
            ],  
            "Command.$": "$.batchJobCmd" ←  
        }  
    },  
    "ResultPath": "$.result",  
    "Next": "Run Athena Rebuild",  
    "Catch": [  
        {  
            "ErrorEquals": [ "States.ALL" ],  
            "ResultPath": "$.failstep",  
            "Next": "Add Status Failed"  
        }  
    ]  
},
```

## Task Definition

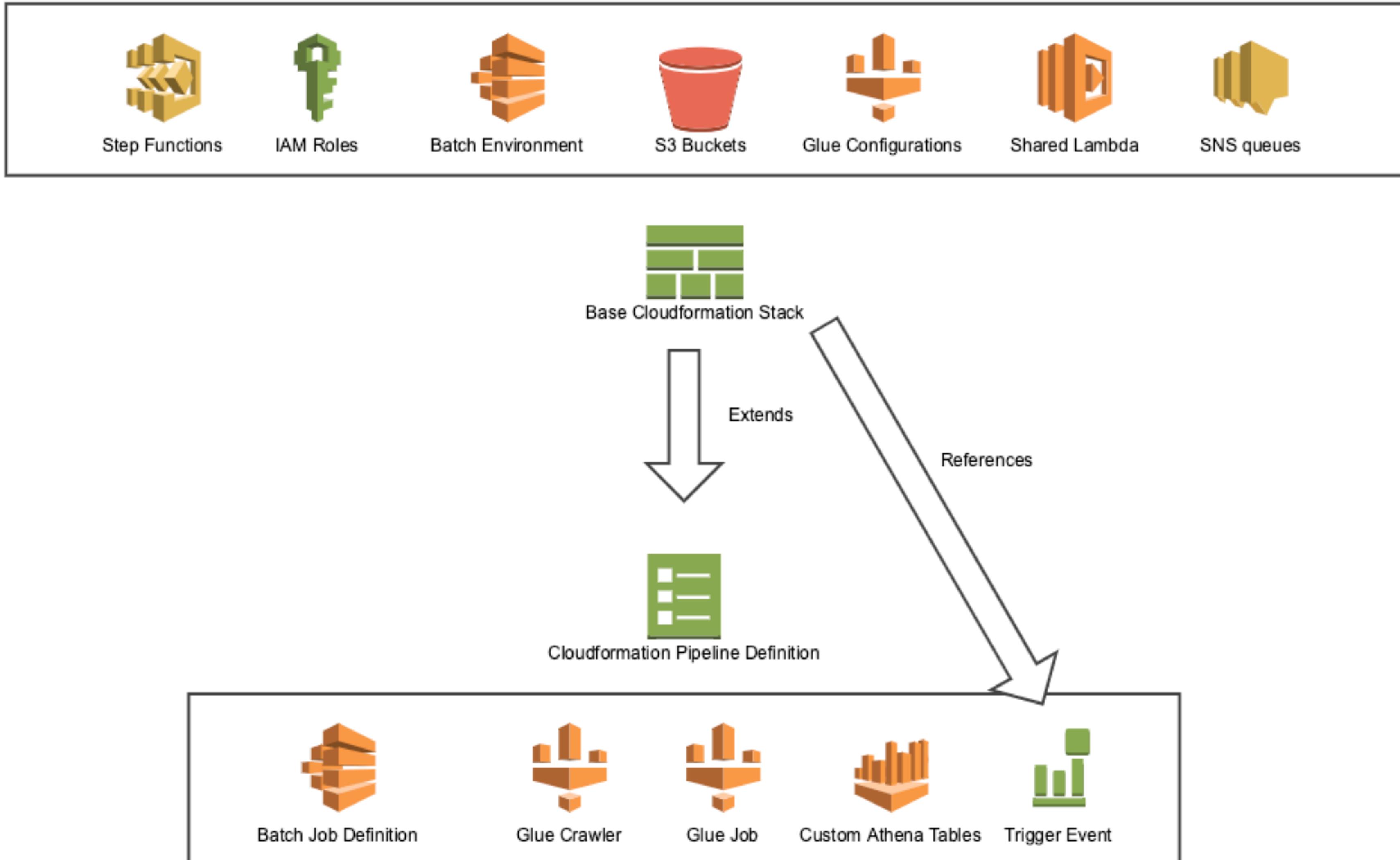
```
{  
    "athenaTable": "parcel_county_example",  
    "athenaDatabase": "structured",  
    "batchJobQueue": "arn:aws:batch:us-east-2:xxxxx:job-queue  
        /ETLJobQueue-xxxxxxxx",  
    "batchJobDef": "arn:aws:batch:us-east-2:xxxxx:job-definition  
        /Parcel_CA_EXAMPLE_Job:21",  
    "glueJobName": "ParcelCaExample_GlueJob",  
    "metricTopic": "arn:aws:sns:us-east-2:xxxxx:DataDogMetrics",  
    "wait_time": "30",  
    "dataType": "parcel",  
    "state": "ca",  
    "county": "example",  
    "jobName": "ParcelCaExample20190715234500",  
    "logging": "{$\"stateMachineLink\": \"https://us-east-2.console  
        .aws.amazon.com/states/home?region=us-east-2#/executions  
        /details/ParcelCaExample20190715234500\",  
        \"$\"stateMachineExecutionID\":  
        \"0190715234500\", \"$\"stepfunctionName\":  
        \"\", \"$\"countyName\": \"Example\",  
        \"$\"state\": \"ca\", \"$\"dataType\": \"parcel\", \"$\"start-time\":  
        \"20190715234500\"}",  
    "batchJobCmd": []  
}
```

## Job Input



**Creating a custom step function for each pipeline, ended up with a lot of duplicated code and as we added more validation we had to go and update it everywhere.**

# Basic Pipeline Definition



# Triggering a Data Pipeline



# Where are the Logs?

- Execution event history summary in the Step Function console
- CloudWatch Logs - **LogGroup/LogStream**
  - Lambda
    - **/aws/lambda/functionName1/LambdaExecutionID1**
    - **/aws/lambda/functionName2/LambdaExecutionID1**
  - Batch
    - **/aws/batch/job/JobName\_Batch\_Job/BatchExecutionID**
  - Glue
    - **/aws-glue/jobs/output/GlueExecutionID**
    - **/aws-glue/jobs/error/GlueExecutionID**
    - **/aws-glue/crawlers/JobName\_Crawler**



**Tried redirecting all the logs for each pipeline to its own cloud watch stream.**

**This is possible but made a mess, all the shortcuts AWS gives you with errors to look at logs no longer worked.**

# JSON Logging

Everything is logged as JSON. python, pyspark, even bash.

For python we use JSON logging with:  
[python-json-logger](#)

For shell scripts eg entrypoint.sh we use a forked version of bash-banyan that lets us set additional fields:

<https://github.com/JessFlan/bash-bunyan>

## Additional Logging Fields

- stateMachineLink
- stateMachineExecutionID
- stepfunctionName
- countyName
- city
- state
- dataType
- start-time

# Data Dog

CloudWatch makes it hard to read JSON logs without expanding every line.

 **DATADOG**

Events

Dashboards

Infrastructure

Monitors

Metrics

Integrations

APM

Notebooks

Logs

Synthetics

**Log Explorer** Save As

Stepfunctionname:ParcelFakeCountyYow x

Filters (8) Hide Controls | 10 results found

DATE ↓	STATEMACHINEEXECUTIONID	STEPFUNCTIONNAME	ST...	COUNTYN...	LOG GROUP	MESSAGE
Apr 27 22:56:24.121	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Pipeline Failed
Apr 27 22:56:12.252	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Download Error
Apr 27 22:55:50.147	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Downloading 50%
Apr 27 22:55:43.689	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Downloading 40%
Apr 27 22:55:38.040	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Downloading 30%
Apr 27 22:55:31.405	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Downloading 20%
Apr 27 22:55:24.777	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Downloading 10%
Apr 27 22:55:11.300	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	New Data Located
Apr 27 22:54:58.333	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Initialising Parcel Scrape
Apr 27 22:52:12.080	ParcelFakeCountyYow20190426222721	ParcelFakeCountyYow	CA	FakeCounty	/aws/batch/job	Example

10

10

2

0

8

# Data Dog Checks

Custom checks are an easy way to report the status of a pipeline or service.

At the end of a job run our custom lambda submits a check with a status of:

- Ok (Job ran produced data)
- Warning (Job ran no new data)
- Critical (Job Failed)

# Dashboard Example

★ Yow Example Dashboard ▾ Edit Widgets +

\$var \* ⚙

The screenshot shows a DataDog dashboard titled "Yow Example Dashboard". On the left is a dark sidebar with the DataDog logo and navigation links: Events, Dashboards, Infrastructure, Monitors, Metrics, Integrations, APM, Notebooks, Logs, and Synthetics. The main area contains several widgets:

- A logo for "DECKARD TECHNOLOGIES" featuring a purple dog head icon and the company name.
- A "aws.billing.estimated\_charges" metric chart showing a 1-month trend with values ranging from +0.082 to +48.09.
- A section titled "Fake County" containing five status indicators: "ParcelDataFake" (WARN), "PropertyDataFake" (OK), "CommunityDataFake" (OK), "CensusDataFake" (OK), and "AerialDataFake" (OK).
- A section titled "Another County" containing five status indicators: "ParcelDataAnother" (OK), "PropertyDataAnother" (OK), "DailyExampleData" (2 2), "City1AnotherData" (WARN), and "City2PropertyData" (WARN).



**Checks are considered 'Inactive' if  
Data Dog doesn't receive an update  
for 24 hours**

# Datadog Events

Events allow us to track pipeline results for months rather than hours.

# Datadog Events

At the end of a job run our custom lambda uses the Datadog events API to:

- Submit an event of type **error** (Job Failed)
  - tagged with “unresolved”

# Datadog Events

At the end of a job run our custom lambda uses the Datadog events API to:

- Submit an event of type **error** (Job Failed)
  - **tagged with “unresolved”**
- Submit an event of type **success** if the job ran produced data
  - **Tag all previous error events as resolved**

# Datadog Events

At the end of a job run our custom lambda uses the Datadog events API to:

- Submit an event of type **error** (Job Failed)
  - **tagged with “unresolved”**
- Submit an event of type **success** if the job ran produced data
  - **Tag all previous error events as resolved**
- Submit an event of type **warning** if the job ran no new data

# Datadog Events

At the end of a job run our custom lambda uses the Datadog events API to:

- Submit an event of type **error** (Job Failed)
  - **tagged with “unresolved”**
- Submit an event of type **success** if the job ran produced data
  - **Tag all previous error events as resolved**
- Submit an event of type **warning** if the job ran no new data

These are all tagged with the same fields as our JSON logs.

# Dashboard Example

☆ Presentation ▾

Add Template Variables ?

## County 1

County 1

OK  
8

STATUS NAME

OK	parcel has new data within 32 days
OK	permit(permitlist): Run within 8 days
OK	permit(permitlist): New data within 16 days
OK	permit(permit): Job succeeded

## County 2

County 2

Alert  
3

OK  
8

STATUS NAME

ALERT	permit: New data within 16 days
ALERT	permit: Run within 8 days
ALERT	permit(permit): New data within 16 days
OK	permit(permit): Job succeeded

## County 3

County 3

Alert  
3

OK  
5

STATUS NAME

ALERT	permit(permitlist): Run within 8 days
ALERT	permit(permitlist): New data within 16 days
ALERT	permit(permit): New data within 16 days

## Assessor Website

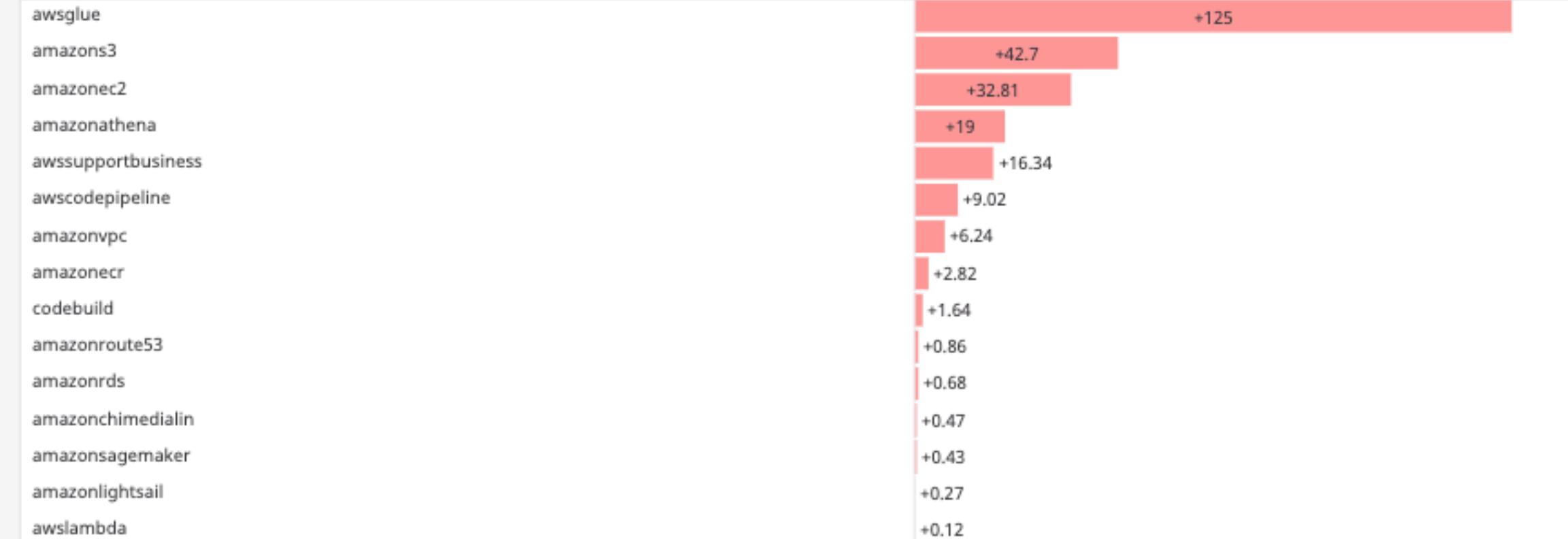
Web Replicant

OK  
2

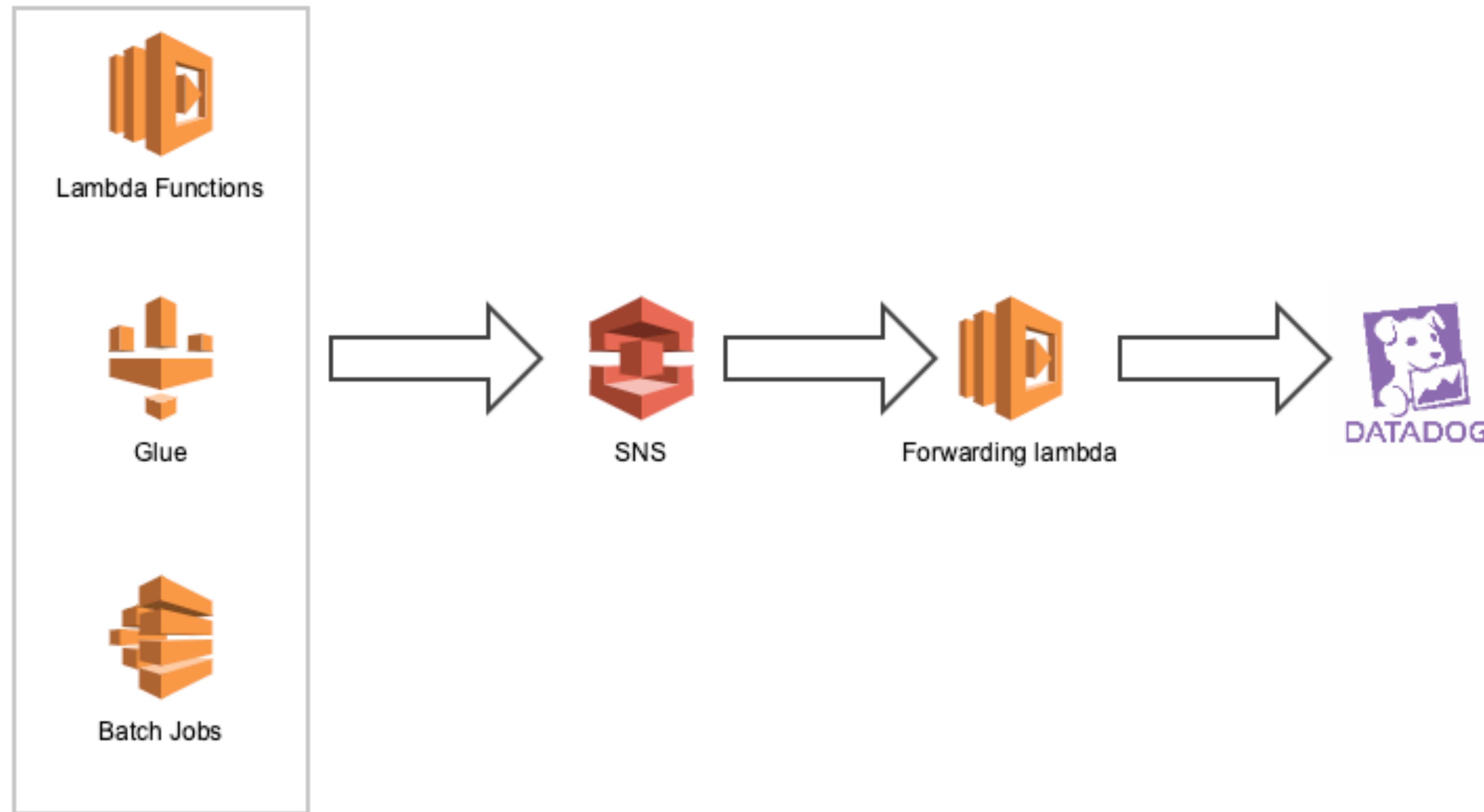
STATUS NAME

OK	[Synthetics] Test on [Synthetics] Test on [Synthetics] Test on [Synthetics]
OK	[Synthetics] Test on [Synthetics] Test on [Synthetics] Test on [Synthetics]

aws.billing.estimated\_charges



# Monitoring Pipeline Metrics



# Pipeline Dashboard

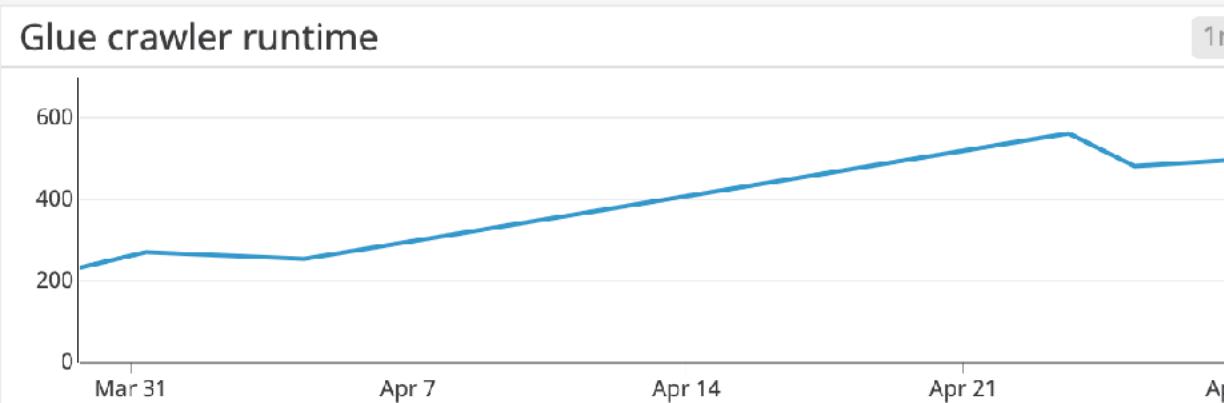
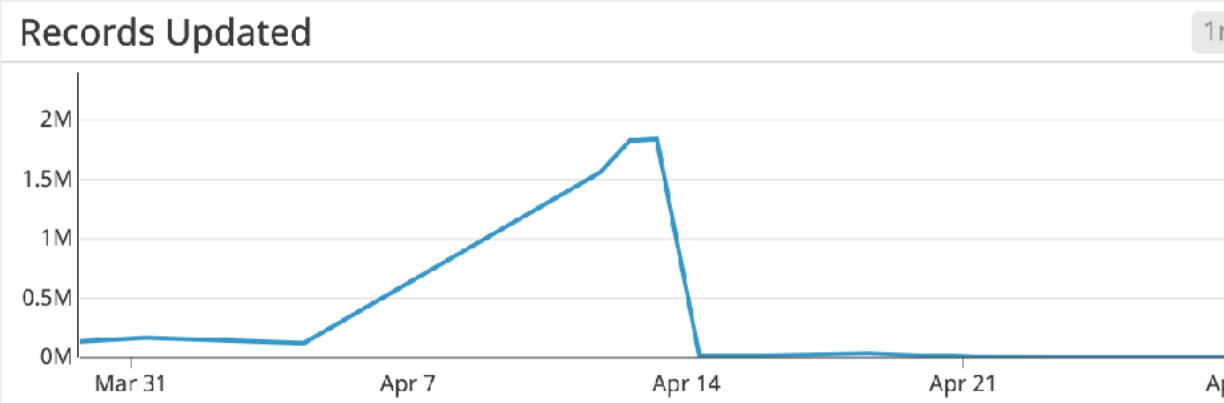
DATADOG

- Events
- Dashboards
- Infrastructure
- Monitors
- Metrics
- Integrations
- APM
- Notebooks
- Logs
- Synthetics

★ Yow Pipeline Dashboard ▾ Edit Widgets +

Add Template Variables ?

## Yow Example Pipeline Dashboard



### 21 logs that match "@countyName:FakeCounty"

DATE ↓	STATEMACHINEEXECUTIONID
Apr 27 22:56:24.121	ParcelFakeCountyYow20190426222721
	Pipeline Failed
Apr 27 22:56:12.252	ParcelFakeCountyYow20190426222721
	Download Error
Apr 27 22:55:50.147	ParcelFakeCountyYow20190426222721
	Downloading 50%
Apr 27 22:55:43.689	ParcelFakeCountyYow20190426222721
	Downloading 40%
Apr 27 22:55:38.040	ParcelFakeCountyYow20190426222721
	Downloading 30%
Apr 27 22:55:31.405	ParcelFakeCountyYow20190426222721
	Downloading 20%
Apr 27 22:55:24.777	ParcelFakeCountyYow20190426222721
	Downloading 10%
Apr 27 22:55:11.300	ParcelFakeCountyYow20190426222721
	New Data Located
Apr 27 22:54:58.333	ParcelFakeCountyYow20190426222721
	Initialising Parcel Scrape

Yow Pipeline

1 1

Yow Sample Check

Alert

1

STATUS NAME

ALERT YOW Test alert

Events that match "countyname:anothercounty"

	<b>New Jira Issue</b>	Sun Apr 28 2019 15:19:18 GMT+1000 (Australian Eastern Standard Time)
	[Triggered on {countyname:anothercounty,environment:dev,host:yow.pipeline,start-time:4,statemachineexecutionid:yowrun1,statemachinelink:https://us-east-2.console.aws.amazon.com/states/home_region_us-east-2/executions/details/jessteststack20190416023829,stepfunctionname:jessteststack}] YOW	Sun Apr 28 2019 15:19:16 GMT+1000 (Australian Eastern Standard Time)
	[Recovered on {countyname:anothercounty,environment:dev,host:yow.pipeline,start-time:20190416023829,statemachineexecutionid:yowrun1,statemachinelink:https://us-east-2.console.aws.amazon.com/states/home_region_us-east-2/executions/details/jessteststack20190416023829,stepfunctionname:jessteststack}] YOW	Sun Apr 28 2019 15:17:19 GMT+1000 (Australian Eastern Standard Time)
	<b>New Jira Issue</b>	Sun Apr 28 2019 15:15:10 GMT+1000 (Australian Eastern Standard Time)
	[Triggered on {countyname:anothercounty,environment:dev,host:yow.pipeline,start-time:20190416023829,statemachineexecutionid:yowrun1,statemachinelink:https://us-east-2.console.aws.amazon.com/states/home_region_us-east-2/executions/details/jessteststack20190416023829,stepfunctionname:jessteststack}] YOW	Sun Apr 28 2019 15:15:08 GMT+1000 (Australian Eastern Standard Time)
	[Recovered on {countyname:anothercounty,environment:dev,host:yow.pipeline,start-time:20190416023829,statemachineexecutionid:yowrun1,statemachinelink:https://us-east-2.console.aws.amazon.com/states/home_region_us-east-2/executions/details/jessteststack20190416023829,stepfunctionname:jessteststack}] YOW	Sun Apr 28 2019 15:10:12 GMT+1000 (Australian Eastern Standard Time)

# Alerting



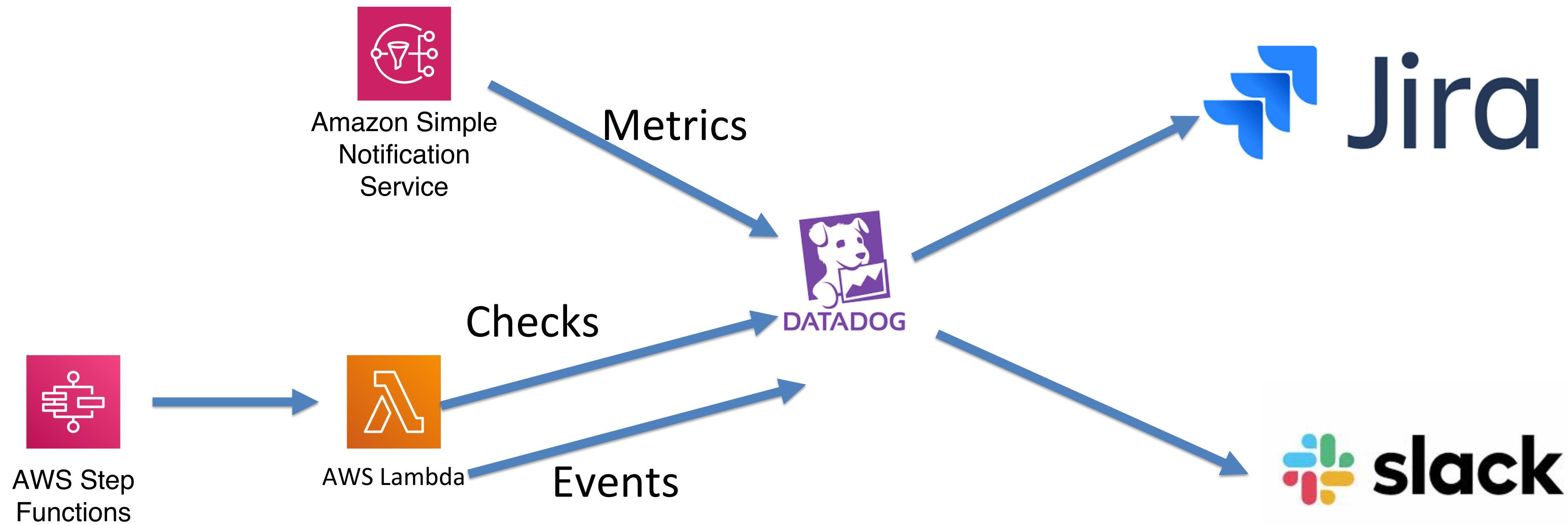
We then tried using SNS task within AWS step functions to generate alerts and send metrics. However, you were only able to provide a JSON path we needed to construct a string and format the output.

# Alerting



**Then we wrote a lambda to format notifications and pushed them to SNS which went to Slack. This was much too noisy; some pipelines run frequently and kept failing. Other failures got lost in the noise everyone thought someone else was dealing with it.**

# Alerting Solution



# Example Alert

Triggered Monitors Manage Monitors Manage Downtime [New Monitor +](#)

Cloning: YOW Test alert  
[Edit Monitor / Custom Check](#)

**1 Pick a Custom Check**  
yow.ok ▾

**2 Pick monitor scope**  
countyname:anothercounty x excluding Select tags to exclude

**3 Set alert conditions**

Check Alert  Cluster Alert An alert is triggered when consecutive run statuses cross your threshold [?](#)

Trigger a separate alert for each (select group) reporting your check. [?](#)

Trigger the alert after selected consecutive failures:

Status: Warning 

Status: Critical 

Do not notify for Unknown status

Resolve the alert after selected consecutive successes:

Status: OK 

Notify if data is missing for more than 10080 minutes. [?](#)

[Never] automatically resolve this event from a no data state. [?](#)

For new hosts, wait 300 seconds before evaluating this monitor [?](#)

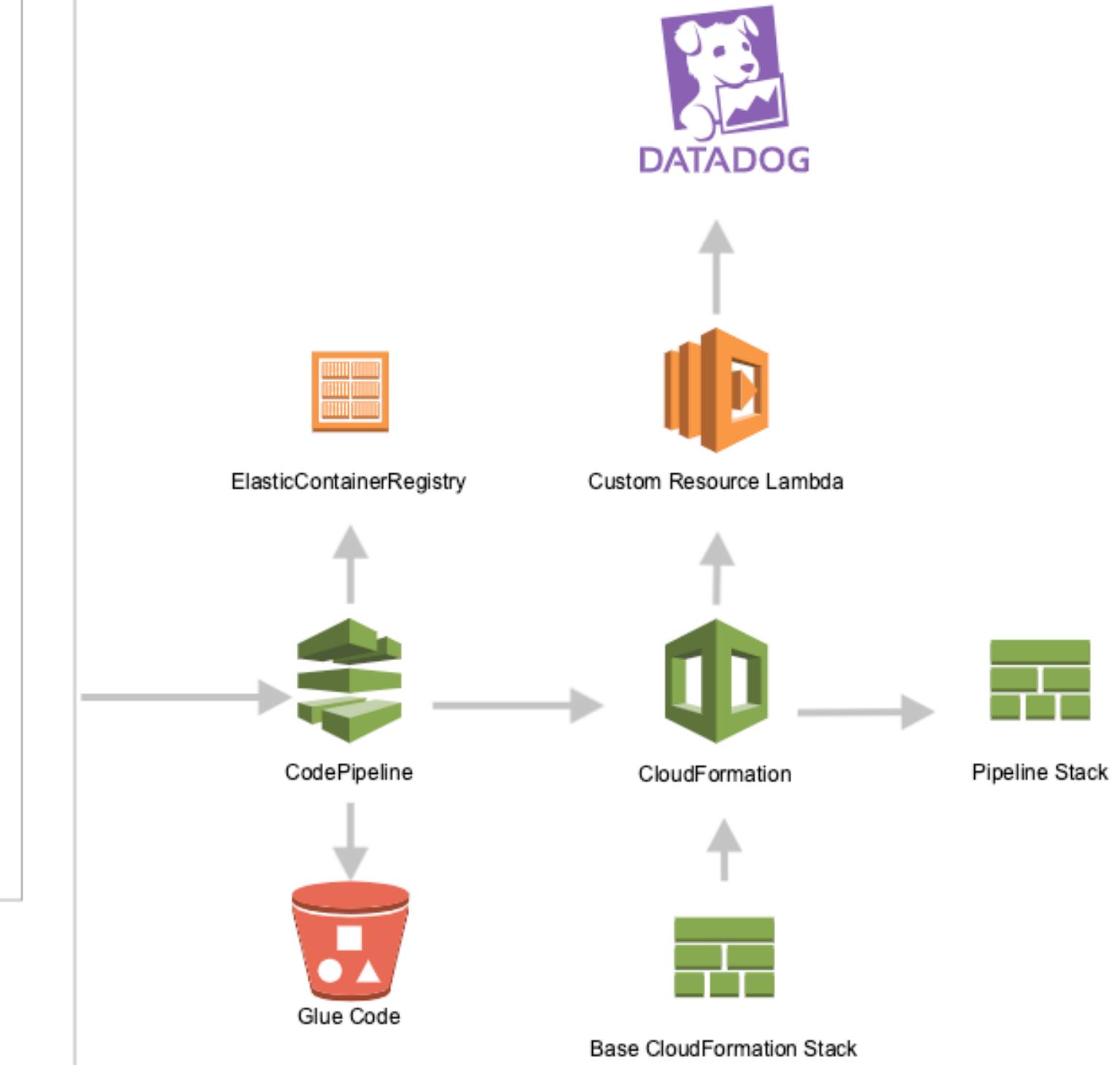
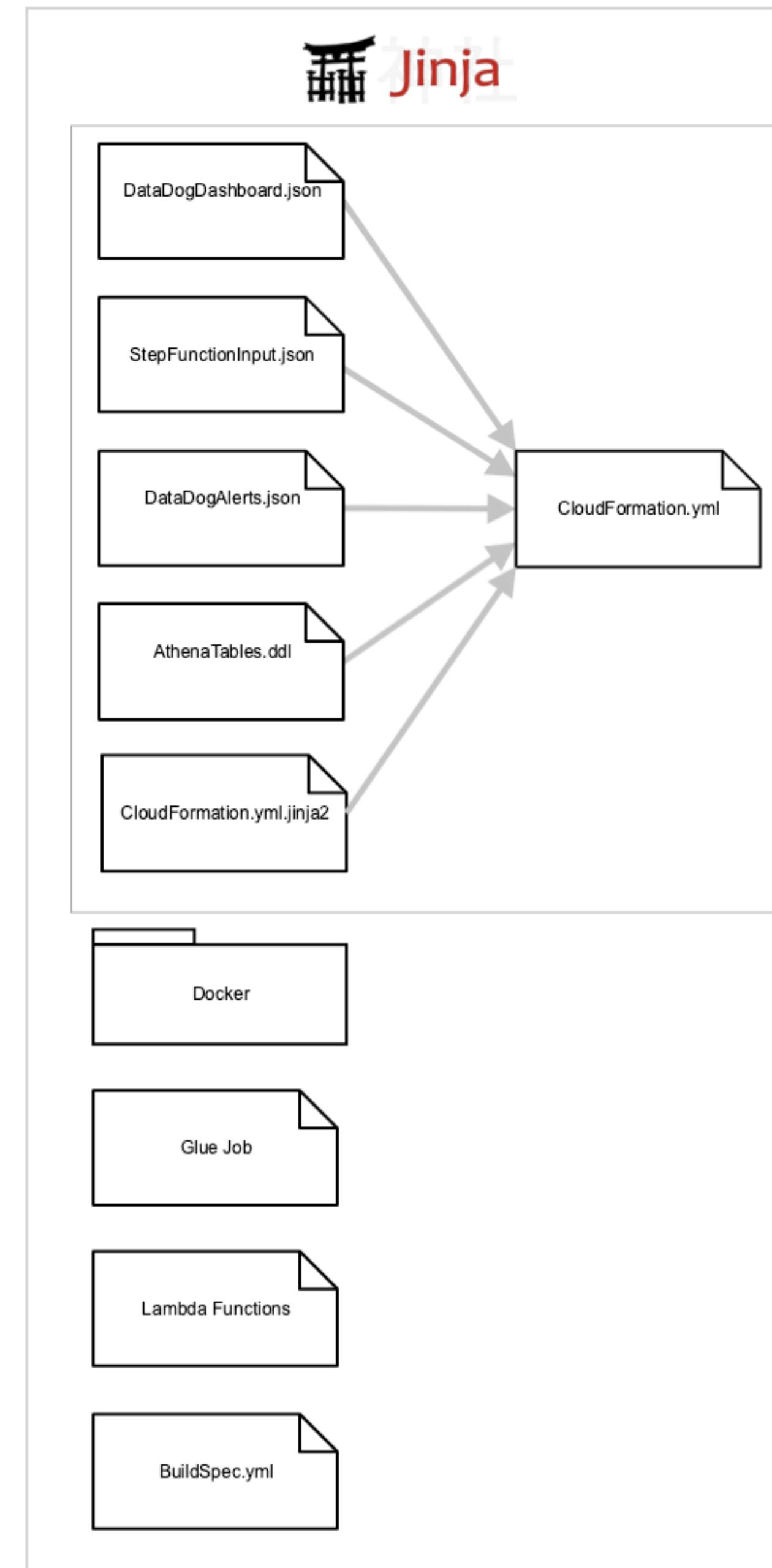
**4 Say what's happening**

Preview  Edit  Include triggering tags in notification title [Use message template variables ?](#) Markdown supported

YOW Test alert

```
{{{#is_alert}}}
@jira-yow-task
{{/is_alert}}
{{{#is_recovery}}}
@jira-update Status is now in recovered state
{{/is_recovery}} @jess@deckardtech.com
```

# Deployment



# Architecture Goals



**Create Pipelines that are easy to debug.  
When they fail we need to see where the  
failure occurred and any related logging.**



**Monitoring so we can see if our pipelines  
are degrading or producing less results  
over time.**



**We need a whole system view so we  
can see that our pipelines are  
running and which succeed or fail.**



**A way of making sure that if our  
pipelines fail that someone is tasked  
with fixing the problem.**

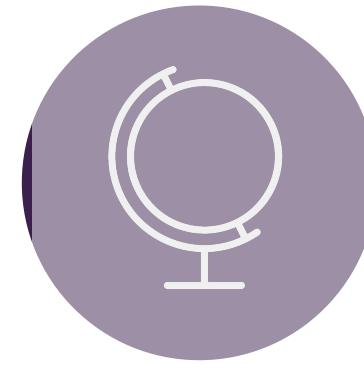
# Architecture Goals



**Create Pipelines that are easy to debug.**  
**When they fail we need to see where the failure occurred and any related logging.**  
***Step Function + JSON logging to DataDog***



Monitoring so we can see if our pipelines are degrading or producing less results over time.



We need a whole of system view so we can see that our pipelines are running and which succeed or fail.



A way of making sure that if our pipelines fail that someone is tasked with fixing the problem.

# Architecture Goals



**Create Pipelines that are easy to debug.**  
When they fail we need to see where the failure occurred and any related logging.

*Step Function + JSON logging to DataDog*



**Monitoring so we can see if our pipelines are degrading or producing less results over time.**



**We need a whole system view so we can see that our pipelines are running and which succeed or fail.**

*DataDog Dashboards*



**A way of making sure that if our pipelines fail that someone is tasked with fixing the problem.**

# Architecture Goals



Create Pipelines that are easy to debug.  
When they fail we need to see where the failure occurred and any related logging.

*Step Function + JSON logging to DataDog*



We need a whole system view so we can see that our pipelines are running and which succeed or fail.

*DataDog Dashboards*



**Monitoring so we can see if our pipelines are degrading or producing less results over time.**

***DataDog Metrics***



A way of making sure that if our pipelines fail that someone is tasked with fixing the problem.

# Architecture Goals



**Create Pipelines that are easy to debug.**  
When they fail we need to see where the failure occurred and any related logging.

*Step Function + JSON logging to DataDog*



We need a whole system view so we can see that our pipelines are running and which succeed or fail.

*DataDog Dashboards*



**Monitoring so we can see if our pipelines are degrading or producing less results over time.**

*DataDog Metrics*



**A way of making sure that if our pipelines fail that someone is tasked with fixing the problem.**

*DataDog Alerts + Jira*

# Recap

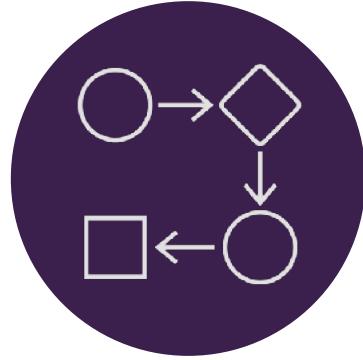


**Log everything in JSON - Bash banyan  
for logging entry points etc.**

# Recap



**Log everything in JSON - Bash banyan  
for logging entry points etc.**

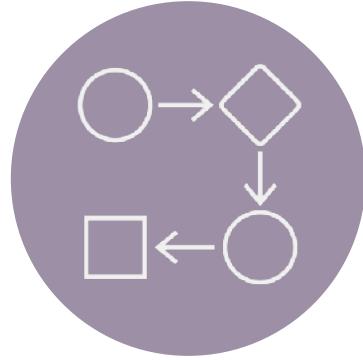


**Where possible share step functions.**

# Recap



**Log everything in JSON - Bash banyan  
for logging entry points etc.**



**Where possible share step functions.**

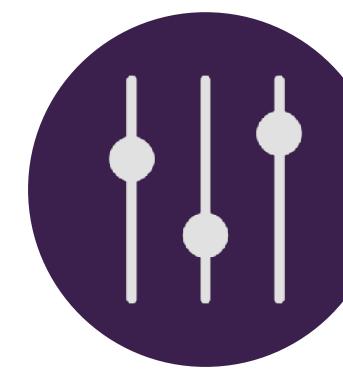


**Use the same tags for checks, events,  
alerts and logging.**

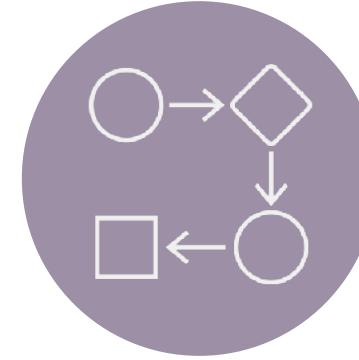
# Recap



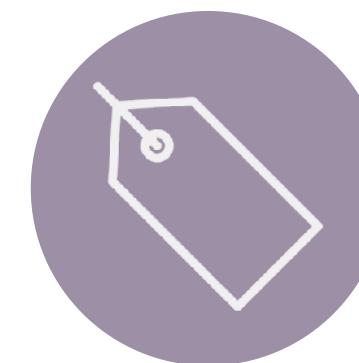
Log everything in JSON - Bash banyan  
for logging entry points etc.



**Use custom resources in cloud  
formation so that tear down is sorted  
and everything is revision controlled.**



Where possible share step functions.

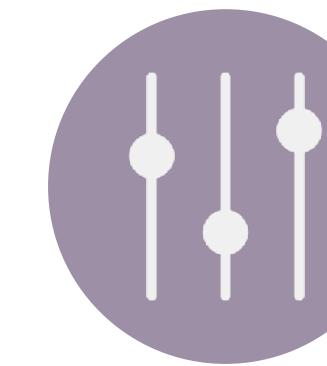


Use the same tags for checks, events,  
alerts and logging.

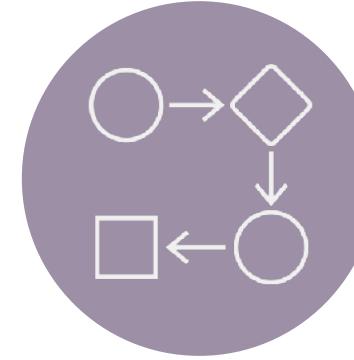
# Recap



**Log everything in JSON - Bash banyan for logging entry points etc.**



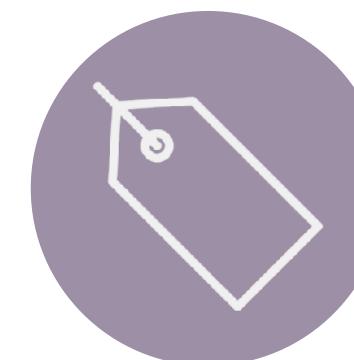
**Use custom resources in cloud formation so that tear down is sorted and everything is revision controlled.**



**Where possible share step functions.**



**Don't move where logging goes just push a copy.**

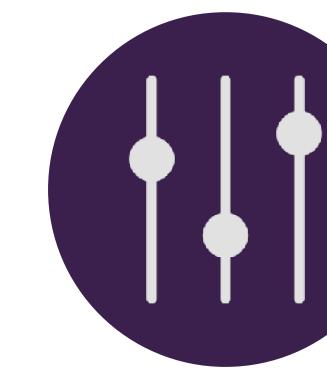


**Use the same tags for checks, events, alerts and logging.**

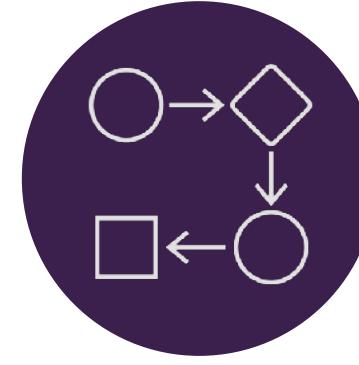
# Recap



**Log everything in JSON - Bash banyan for logging entry points etc.**



**Use custom resources in cloud formation so that tear down is sorted and everything is revision controlled.**



**Where possible share step functions.**



**Don't move where logging goes just push a copy.**



**Use the same tags for checks, events, alerts and logging.**

# Questions?



**Jessica Flanagan**  
Chief of Data Architecture  
Deckard Technologies