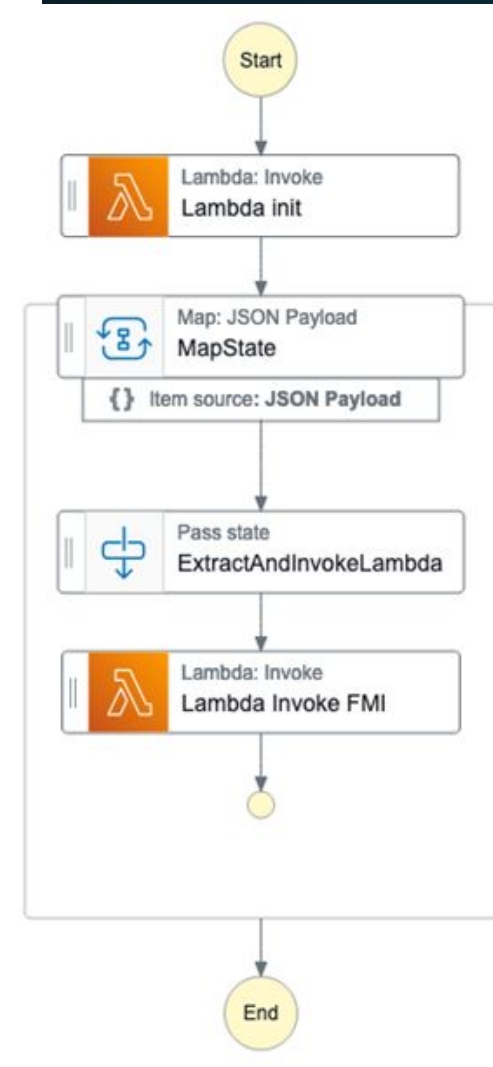


# Project Step 1 Assignment: AWS Lambda Step Function Submission

Michael Amadi and Christian Ollen

# Designed state machine in AWS Step Functions

- AWS Step Function designed to orchestrate distributed inference. It initializes the environment, maps over partitioned payloads, invokes Lambda workers with synchronized rank/world size, and finalizes results.



StepFunctions-MyStateMachine-e5ydt2afc-role-z0ymrut89 [Info](#) [Delete](#)

**Summary** [Edit](#)

**Creation date**  
August 08, 2024, 11:53 (UTC-04:00)

**Last activity**  
55 minutes ago

**ARN**  
arn:aws:iam::211125778552:role/service-role/StepFunctions-MyStateMachine-e5ydt2afc-role-z0ymrut89

**Maximum session duration**  
1 hour

[Permissions](#) [Trust relationships](#) [Tags](#) [Last Accessed](#) [Revoke sessions](#)

**Permissions policies (6)** [Info](#) [Simulate](#) [Remove](#) [Add permissions](#)

You can attach up to 10 managed policies.

**Filter by Type** [All types](#) [<](#) [1](#) [>](#) [Settings](#)

<input type="checkbox"/>	Policy name <a href="#">?</a>	Type	Attached entities
<input type="checkbox"/>	<a href="#">AmazonS3FullAccess</a>	AWS managed	14
<input type="checkbox"/>	<a href="#">AWSLambda_FullAccess</a>	AWS managed	5
<input type="checkbox"/>	<a href="#">AWSStepFunctionsFullAccess</a>	AWS managed	2
<input type="checkbox"/>	<a href="#">cloudwatchaccess</a>	Customer inline	0
<input type="checkbox"/>	<a href="#">LambdaInvokeScopedAccessPolicy-2cc79138-73d6-498b-a822-...</a>	Customer managed	1
<input type="checkbox"/>	<a href="#">XRayAccessPolicy-fcb083fb-726c-4c08-bf1c-46c7492a31cc</a>	Customer managed	1

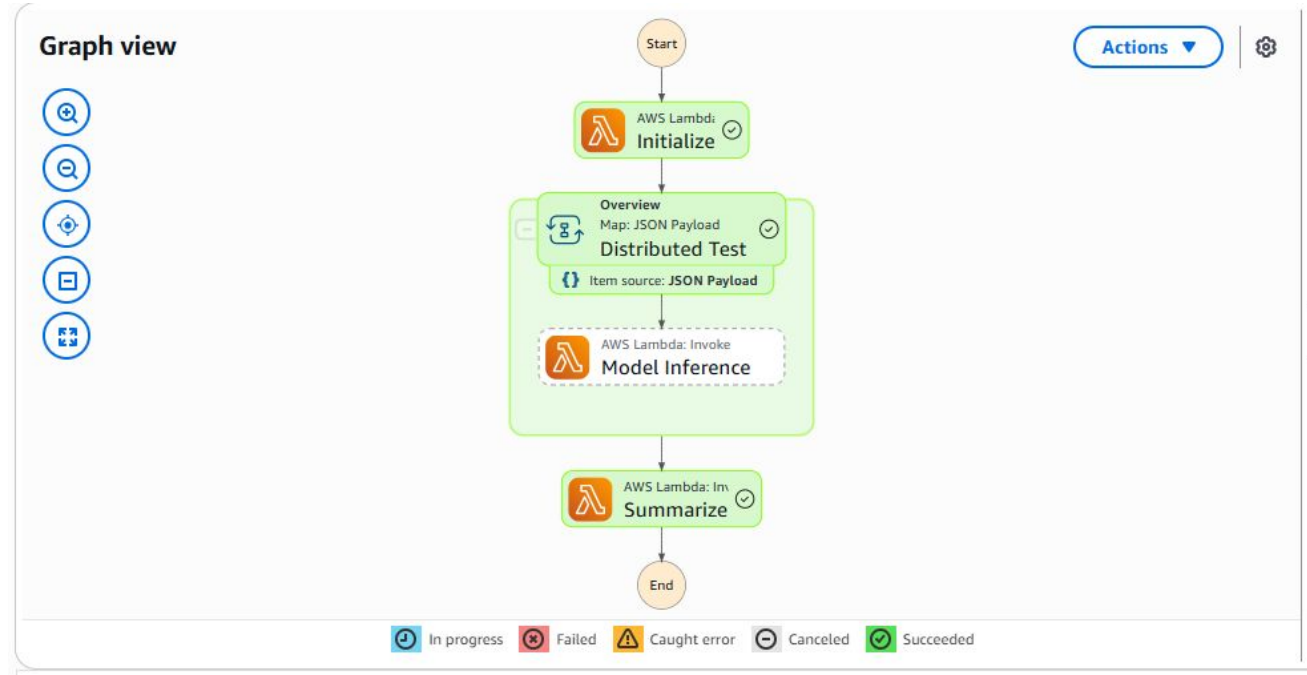
- Configured IAM role with AmazonS3FullAccess for testing purposes. Limited least-privilege policies can be applied in production to scope only to relevant buckets and paths.

# Configured IAM roles for access to S3 buckets

```
{  
  "bucket": "cosmicai-data",  
  "file_limit": "11",  
  "batch_size": 512,  
  "object_type": "folder",  
  "S3_object_name": "Anomaly Detection",  
  "script": "/tmp/Anomaly Detection/Inference/inference.py",  
  "result_path": "result-partition-100MB/1GB/1",  
  "data_bucket": "cosmicai-data",  
  "data_prefix": "100MB"  
}
```

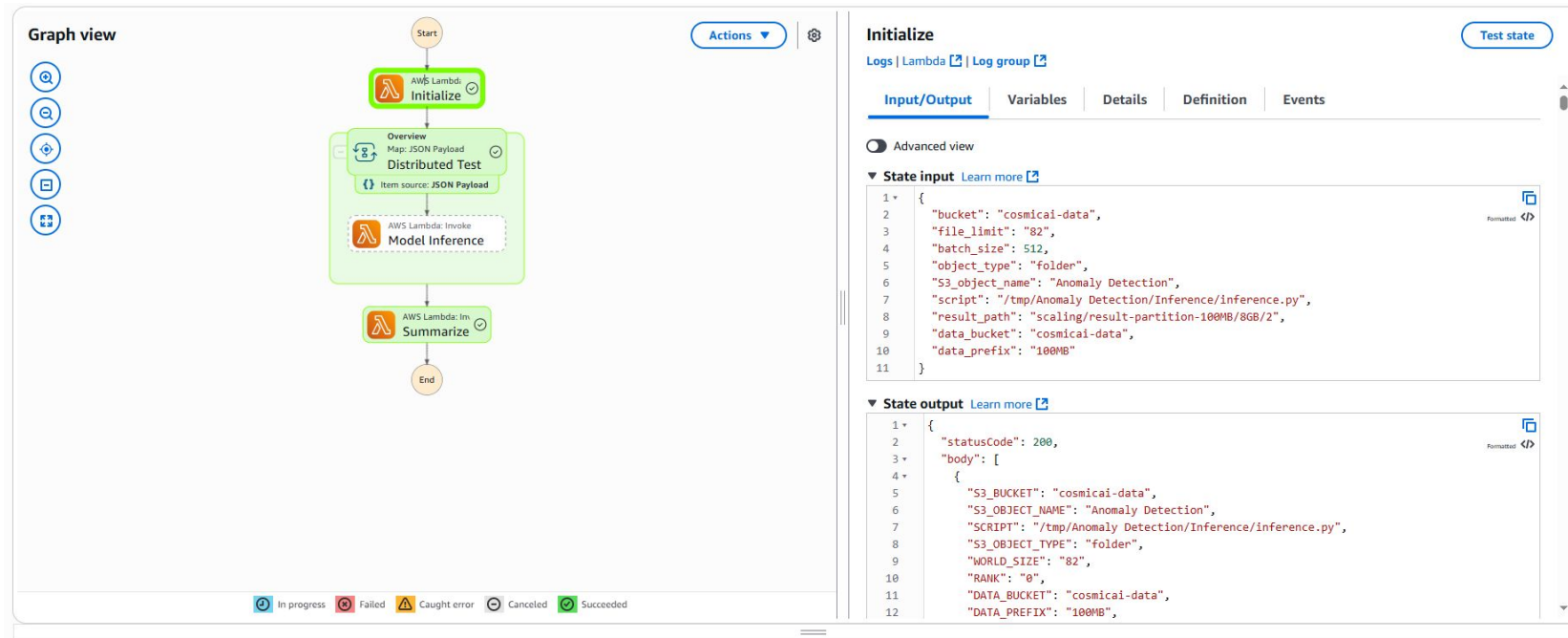
- Each item in the payload represents a parallel task, specifying its data partition (DATA\_PATH), RANK, and RESULT\_PATH. WORLD\_SIZE is shared across tasks to coordinate distributed inference.

## Used JSON payloads to pass parameters



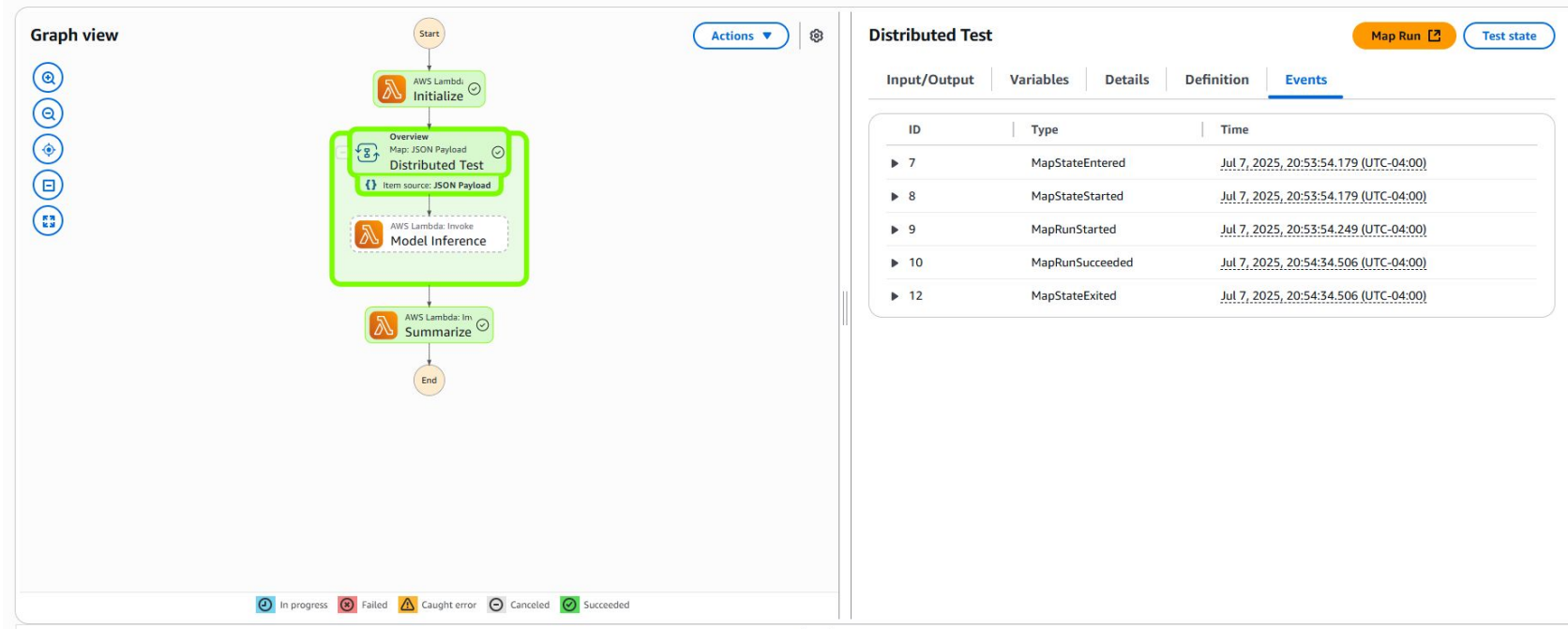
- The workflow demonstrates successful initialization, parallel model inference using a Map state with dynamic JSON payloads, and final summarization. The green checks confirm that the scalable pipeline completed without errors across all configurations.

Executed a scalable workflow capable of processing across configurations



- This step reads the input configuration (e.g., world size, batch size, data path) and transforms it into a JSON payload for downstream processing. The output contains individual task parameters, enabling parallel execution across partitions.

# Lambda Init: Prepare runtime environment and load configurations (e.g. world size, batch size)



- The Map state iterates over the JSON payload and launches concurrent Lambda executions for each partition. This enables scalable and parallel processing based on configuration parameters like world size and batch size.

# Map State: Assign tasks to multiple Lambda functions, iterating over a JSON payload



## Extract and Invoke: Ensure that each Lambda function receives necessary allocated data and carries out inference

- Each Lambda function receives a unique payload entry in the body array with parameters including RANK, DATA\_PATH, and BATCH\_SIZE. This enables the function to extract the correct data shard, run inference using the specified script, and save results to the appropriate S3 path. This demonstrates that data is extracted and inference is invoked per rank in a distributed manner.

```
{
  "statusCode": 200,
  "body": [
    {
      "S3_BUCKET": "cosmicai-data",
      "S3_OBJECT_NAME": "Anomaly Detection",
      "SCRIPT": "/tmp/Anomaly Detection/Inference/inference.py",
      "S3_OBJECT_TYPE": "folder",
      "WORLD_SIZE": "82",
      "RANK": "0",
      "DATA_BUCKET": "cosmicai-data",
      "DATA_PREFIX": "100MB",
      "DATA_PATH": "100MB/1.pt",
      "RESULT_PATH": "scaling/result-partition-100MB/8GB/2",
      "BATCH_SIZE": 512
    },
    {
      "S3_BUCKET": "cosmicai-data",
      "S3_OBJECT_NAME": "Anomaly Detection",
      "SCRIPT": "/tmp/Anomaly Detection/Inference/inference.py",
      "S3_OBJECT_TYPE": "folder",
      "WORLD_SIZE": "82",
      "RANK": "1",
      "DATA_BUCKET": "cosmicai-data",
      "DATA_PREFIX": "100MB",
      "DATA_PATH": "100MB/10.pt",
      "RESULT_PATH": "scaling/result-partition-100MB/8GB/2",
      "BATCH_SIZE": 512
    },
  ]
}
```

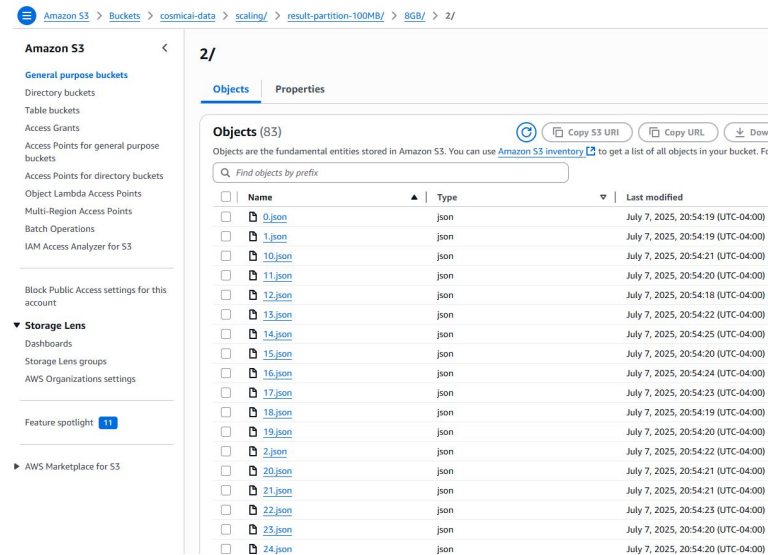


## Lambda Invoke FMI: Invoke FMI interface to synchronize distributed tasks

- Each Lambda worker is connected to the Federated Machine Interface (FMI) using `fmi_communicator`. Workers are synchronized using a `comm.barrier()` call, and distributed performance metrics (e.g., total CPU time) are reduced across ranks using `comm.reduce`. This confirms that all inference tasks are coordinated and measured collectively in a distributed environment.

```
164     comm = fmi_communicator(args.world_size, rank)
165
166     comm.hint(fmi.hints.fast)
167
168     comm.barrier()
169     reduce_res = comm.reduce(total_time, 0, fmi.func(fmi.op.sum), fmi.types(fmi.datatypes.double))
---
```

# End State: Ensure all results are logged and stored in S3



The screenshot shows the Amazon S3 console interface. The breadcrumb navigation at the top indicates the path: Amazon S3 > Buckets > cosmicai-data > scaling/ > result-partition-100MB/ > 8GB/ > 2/. The left sidebar shows the 'Storage Lens' section expanded. The main content area displays a table of objects under the 'Objects (83)' tab. The table has columns for Name, Type, and Last modified. The objects are listed sequentially from 0.json to 24.json, all of type 'json' and created on July 7, 2025.

Name	Type	Last modified
0.json	json	July 7, 2025, 20:54:19 (UTC-04:00)
1.json	json	July 7, 2025, 20:54:19 (UTC-04:00)
10.json	json	July 7, 2025, 20:54:21 (UTC-04:00)
11.json	json	July 7, 2025, 20:54:20 (UTC-04:00)
12.json	json	July 7, 2025, 20:54:18 (UTC-04:00)
13.json	json	July 7, 2025, 20:54:22 (UTC-04:00)
14.json	json	July 7, 2025, 20:54:25 (UTC-04:00)
15.json	json	July 7, 2025, 20:54:20 (UTC-04:00)
16.json	json	July 7, 2025, 20:54:24 (UTC-04:00)
17.json	json	July 7, 2025, 20:54:23 (UTC-04:00)
18.json	json	July 7, 2025, 20:54:19 (UTC-04:00)
19.json	json	July 7, 2025, 20:54:20 (UTC-04:00)
2.json	json	July 7, 2025, 20:54:22 (UTC-04:00)
20.json	json	July 7, 2025, 20:54:21 (UTC-04:00)
21.json	json	July 7, 2025, 20:54:21 (UTC-04:00)
22.json	json	July 7, 2025, 20:54:23 (UTC-04:00)
23.json	json	July 7, 2025, 20:54:20 (UTC-04:00)
24.json	json	July 7, 2025, 20:54:20 (UTC-04:00)

```
{
  "statusCode": 200,
  "body": [
    {
      "S3_BUCKET": "cosmicai-data",
      "S3_OBJECT_NAME": "Anomaly Detection",
      "SCRIPT": "/tmp/Anomaly Detection/Inference/inference.py",
      "S3_OBJECT_TYPE": "folder",
      "WORLD_SIZE": "82",
      "RANK": "0",
      "DATA_BUCKET": "cosmicai-data",
      "DATA_PREFIX": "100MB",
      "DATA_PATH": "100MB/1.pt",
      "RESULT_PATH": "scaling/result-partition-100MB/8GB/2",
      "BATCH_SIZE": 512
    }
  ]
}
```

- Upon completion, all inference results are serialized as JSON and stored in Amazon S3, organized by partition and configuration (e.g., world size, data prefix). This ensures reproducibility, traceability, and scalable post-processing access.

PARTITION	WORLD SIZE	REQUESTS	DURATION (\$)	MEMORY	COST (\$)
25MB	517	517	6.55	2.8GB	0.16
50MB	259	259	11.8	4.0GB	0.20
75MB	173	173	17.6	5.9GB	0.30
100MB	130	130	25.0	7.0GB	0.38

- Performance was evaluated by varying world size and input partition. Larger world sizes improved throughput but increased memory consumption. Trade-offs between cost and performance were observed.

Performance measurement: Create a Table to show the performance in "memory, duration and cost" of Step Function. Measure the execution time by varying the world size.

## Brief explanation (1-2 paragraphs) of your implementation approach and any challenges encountered

- **Implementation Summary:** To implement the distributed inference pipeline, we designed an AWS Step Function that orchestrates multiple Lambda functions for scalable model execution. The workflow begins with the Initialize Lambda, which reads configuration settings such as `batch_size`, `world_size`, and the input data mapping from an S3-hosted JSON payload. The Step Function then uses a Map state to invoke parallel Lambda workers, each assigned a unique rank to process a separate chunk of input data using the `inference_FMI.py` script. Each worker performs model inference, records performance metrics (e.g., memory usage, CPU time, throughput), and stores these results in S3. Finally, the Summarize Lambda can be used to consolidate and log outcomes.
- **Challenges Encountered:** One of the primary challenges was ensuring correct environment configuration across parallel Lambda executions. This included managing S3 permissions, setting up correct payload mappings, and integrating FMI for distributed coordination. Initial runs failed due to missing input parameters (e.g., `bucket`), highlighting the importance of passing a complete and correctly structured payload to the Step Function. Once resolved, the system successfully executed across multiple world sizes, allowing performance comparisons by varying batch sizes and rank count.