LambdaFlow

Drag. Drop. Deploy — Instant Serverless Automation

LambdaFlow is a visual serverless workflow builder powered by AWS Lambda, enabling instant automation of complex tasks. Each step runs as an isolated Lambda function, orchestrating AWS services like S3, EventBridge, and DynamoDB—all through an intuitive, no-code interface.

Problem Statement

1. Barrier to Entry for Non-Experts

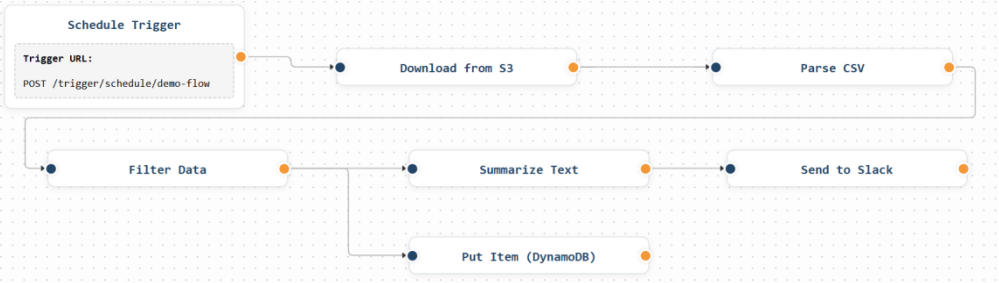
Problem: Only experienced developers can build automation pipelines.

Solution:

LambdaFlow

Prebuilt templates, default node configs, and code-only Custom Node let even semi-technical users build powerful flows.

Example: A business analyst sets up a daily report workflow that runs on a schedule, pulls sales data from S3, filters top-performing products, stores the data & send the summary in Slack— without writing any code.



Problem Statement

2. Complexity in Orchestration

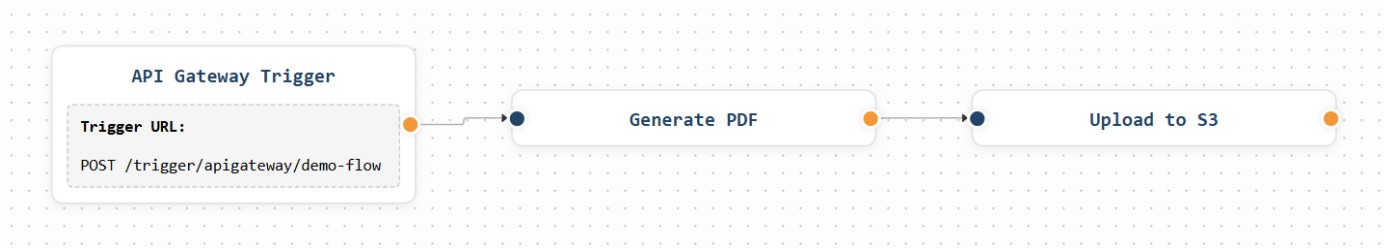
Problem: Most of the automation tools require verbose JSON definitions, strict state management, and deep cloud knowledge.

Solution:

LambdaFlow

Drag-and-drop builder using React Flow to visually define workflow steps.

Example: Connect a API Gateway Trigger to a PDF generator and S3 upload in 60 seconds, no JSON/YAML required.



Problem Statement

3. Lack of Visual Transparency

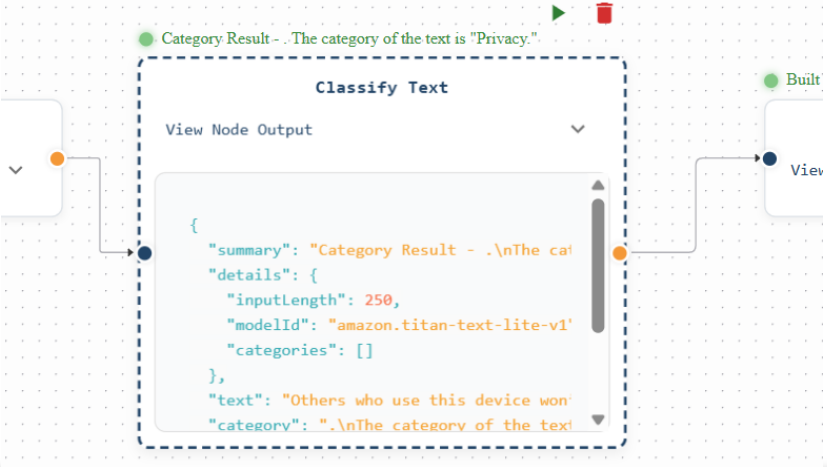
Problem: Complex flows in most tools are hard to debug visually

Solution:

LambdaFlow

Every node shows live status, output, and logs in a side panel with tree + JSON views.

Example: View the output of a "Classify Text" node directly, trace data flow step-by-step across the graph.



Problem Statement

4. Tight Coupling & Modularity

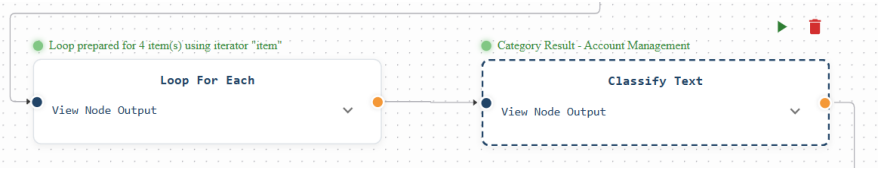
Problem: Logic is hard-wired between services; changes break upstream / downstream flow.

Solution:

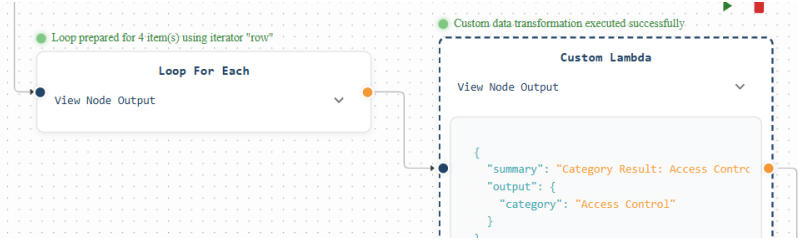
LambdaFlow

Workflows are modular DAGs — each node runs in its own Lambda, making logic isolated, reusable, and easy to swap without affecting others.

Example: Replace a “Classify Text” node with a "Custom Code" node to test a new model—no impact on other steps.



Before



After

Problem Statement

5. Fragmented Tooling Across AWS Services

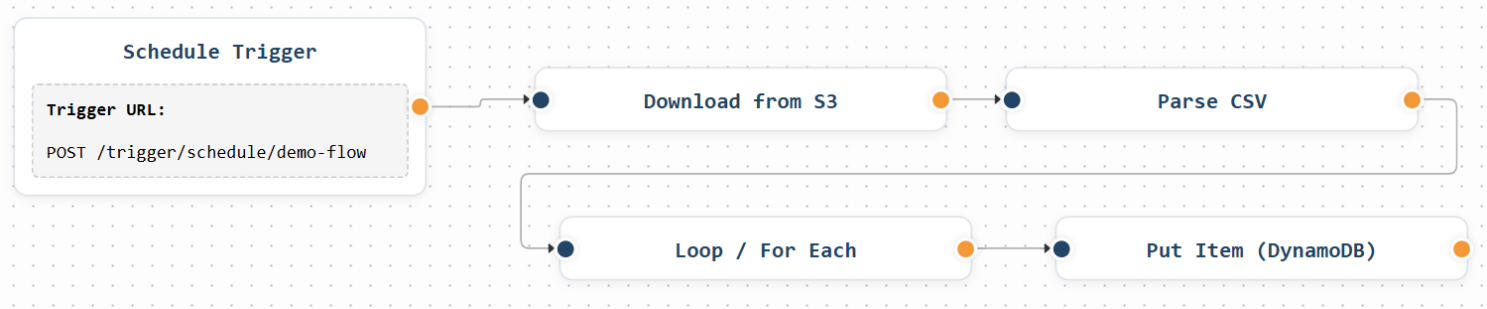
Problem: Developers jump between S3, DynamoDB, EventBridge etc. to link services.

Solution:

LambdaFlow

Unified node library for S3 triggers, SNS publishing, DynamoDB reads, EventBridge scheduling, etc.

Example: Set up a scheduled workflow to fetch a CSV from S3, parse it, and upload each row to DynamoDB—using just 5 nodes, with almost no manual configurations.



Problem Statement

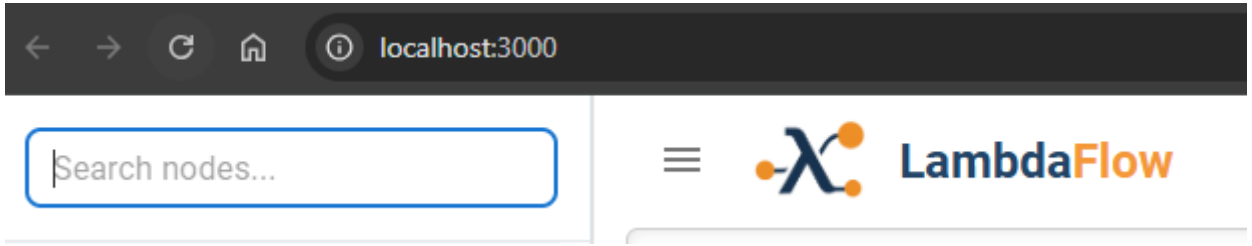
6. No Unified Local + Cloud Execution

Problem: Code written for local testing doesn't scale to Lambda without modification.

Solution:

LambdaFlow

Nodes can run offline (in local mode) or be deployed to AWS Lambda with zero code change.



Problem Statement

7. Slow Development Cycle for Automation

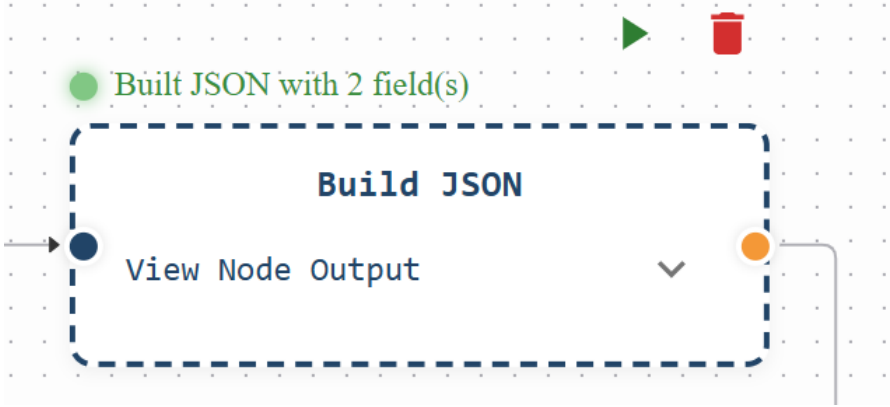
Problem: Manually wiring Lambda functions, testing permissions, and iterating is slow.

Solution:

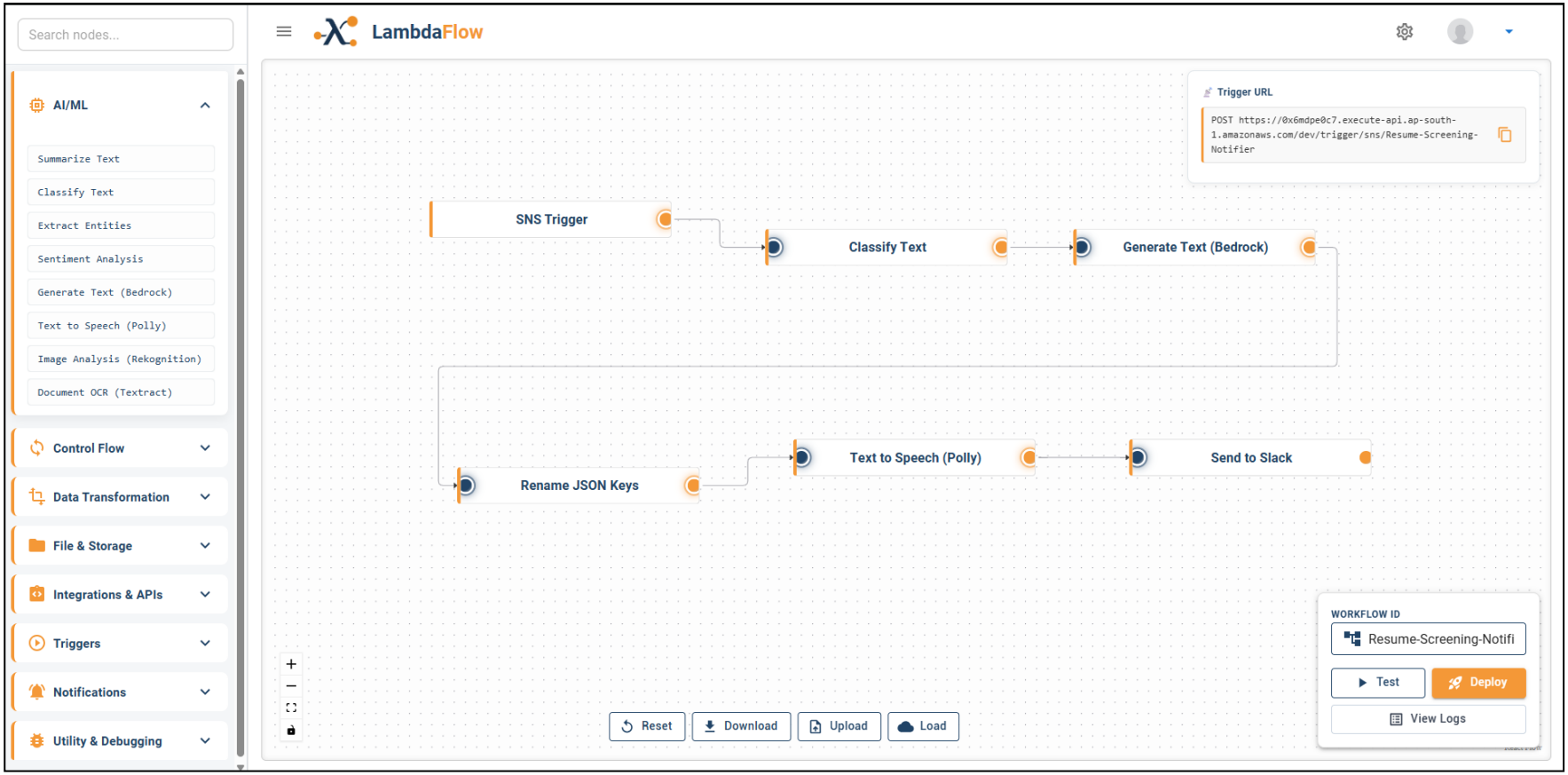
LambdaFlow

Each node runs in its own AWS Lambda, decoupling logic and allowing per-node deployment and testing.

Example: Update a single data transformation node, redeploy just that Lambda, and test it independently—no monolithic redeployment needed.

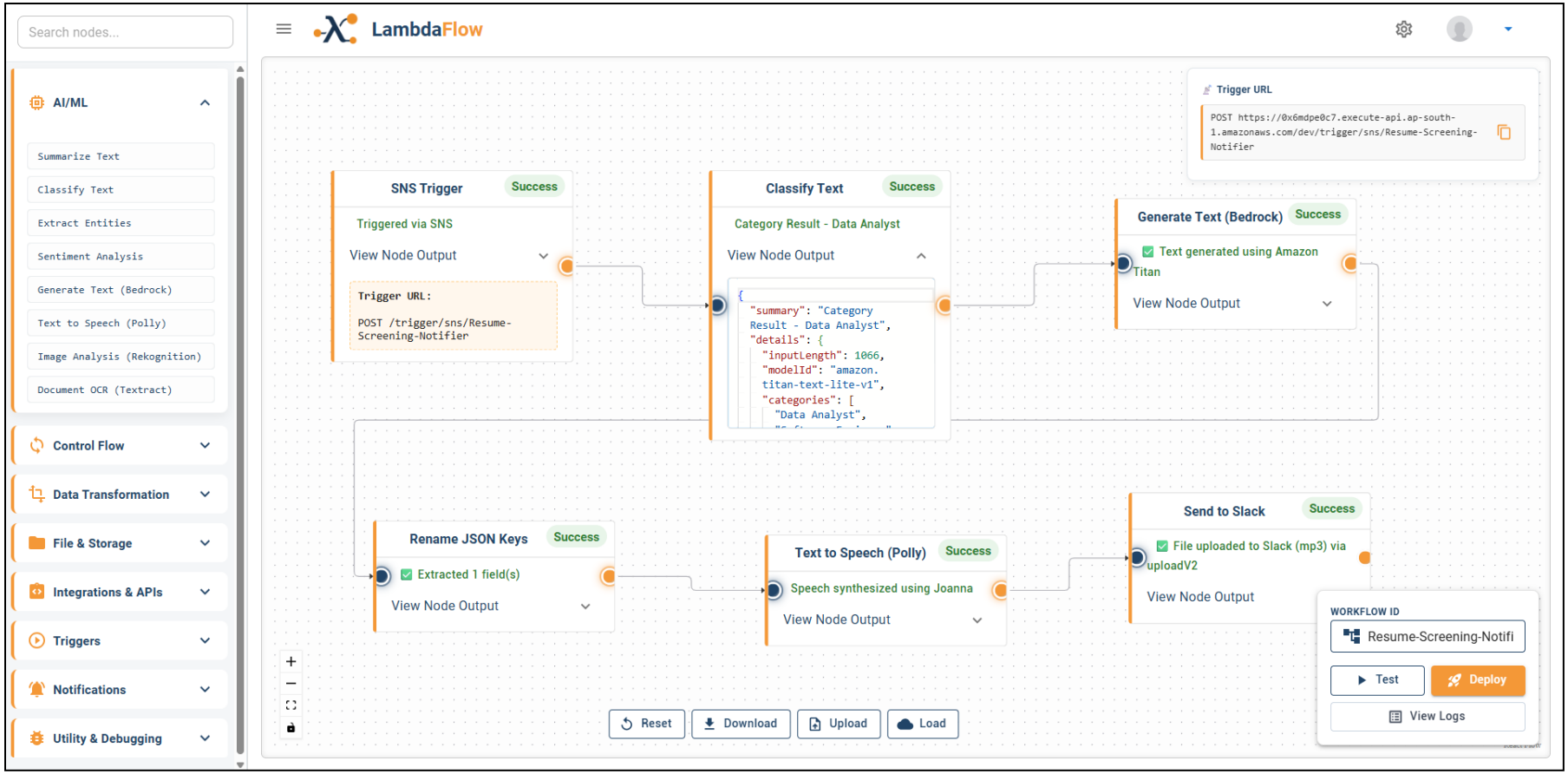


LambdaFlow - Builder



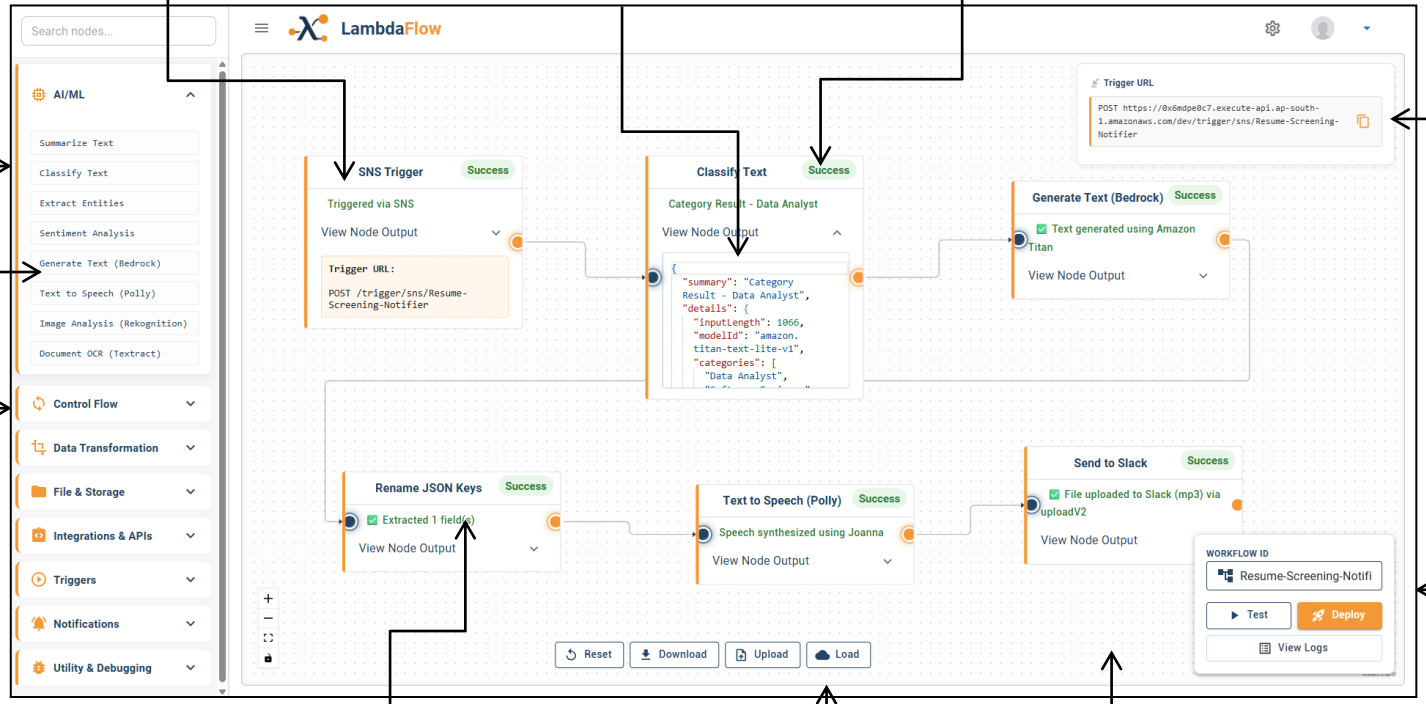
New Workflow - Before Execution

LambdaFlow - Builder



Nodes with Execution Result

LambdaFlow - Builder



Trigger Node

Node Output

Execution Status

Node Panel

Trigger URL

Nodes

Node Category

Workflow ID

Execution Summary

Download Workflows

Builder Canvas

LambdaFlow - Builder

Search nodes...

LambdaFlow

AI/ML

- Summarize Text
- Classify Text
- Extract Entities
- Sentiment Analysis
- Generate Text (Bedrock)
- Text to Speech (Polly)
- Image Analysis (Rekognition)
- Document OCR (Textract)

Control Flow

Data Transformation

File & Storage

Integrations & APIs

Triggers

Notifications

Utility & Debugging

SNS Trigger — Success

Triggered via SNS

View Node Output

Trigger URL:

POST /trigger/sns/Resume-Screening-Notifier

Rename JSON Keys — Success

Extracted 1 field(s)

View Node Output

Reset

Dow

Execution Logs

List | Raw | Table

Search logs...

NODEID	LABEL	SUCCESS	SUMMARY	DETAILS	OUTPUT
0468968f-9696-437c-	Classify Text	true	Category Result - Data	{"inputLength":1066,"m...	{"summary":"Category
75c8132e-2737-4f99-	Generate Text (Bedrock)	true	Text generated using	{}	{"summary":" Text
652dff86-59b7-4872-	Rename JSON Keys	true	Extracted 1 field(s)	{"mapping":	{"summary":"
b68c0827-d1ca-4677-	Text to Speech (Polly)	true	Speech synthesized	{}	{"summary":"Speech
f2071278-1f78-4b18-	Send to Slack	true	File uploaded to	{"ok":true,"files":	{"fileName":"speech_1...
0e7de9a3-3ebe-4019-	SNS Trigger	true	Triggered via SNS	{}	{}

Rows per page: 100 1–6 of 6

Execution Logs – Table View

LambdaFlow - Builder

Execution Logs

List | Raw | Table

Search logs...


```
[
  {
    "nodeId": "0468968f-9696-437c-b60c-999eec152ab8",
    "label": "Classify Text",
    "success": true,
    "summary": "Category Result - Data Analyst",
    "details": {
      "inputLength": 1066,
      "modelId": "amazon.titan-text-lite-v1",
      "categories": [
        "Data Analyst",
        "Software Engineer",
        "Project Manager"
      ]
    },
    "output": {
      "summary": "Category Result - Data Analyst",
      "details": {
        "inputLength": 1066,
        "modelId": "amazon.titan-text-lite-v1",
        "categories": [
          "Data Analyst",
          "Software Engineer",
          "Project Manager"
        ]
      },
      "text": " Jordan Patel Email: jordan.patel@example.com Phone: +1-555-123-7890 LinkedIn: linkedin.com/in/jordanpatel  Professional
      Summary: Detail-oriented Data Analyst with over 3 years of experience interpreting and analyzing data to drive business
      solutions. Strong skills in SQL, Python, Tableau, and Excel. Proven track record of building dashboards, automating reports, and
      delivering insights.  Skills: - SQL, Python, R - Tableau, Power BI, Excel - A/B Testing, Forecasting, Clustering, Regression
      Work Experience: Data Analyst | Retail Analytics Co. | Aug 2021 – Present - Built sales performance dashboards and monitored KPIs
      for 50+ stores. - Automated weekly data pipelines, saving 8+ hours/week. - Presented findings to cross-functional stakeholders.
      Business Data Intern | TechSolutions | Jan 2020 – Jul 2021 - Cleaned CRM datasets using Python scripts and regex. - Developed a
      customer churn model with 82% accuracy.  Education: B.Sc. in Statistics, University of Illinois, 2020  Certifications: - Google
      Data Analytics Certificate - Tableau Desktop Specialist",
      "category": "Data Analyst",
      "aiResult": "Category Result - Data Analyst"
    }
  },
  {
    "nodeId": "75c8132e-2737-4f99-a2cc-e8526182f525",
    "label": "Generate Text (Bedrock)",
    "success": true,
    "summary": "✅ Text generated using Amazon Titan",
    "details": {},
    "output": {
      "summary": "✅ Text generated using Amazon Titan",
```


Execution Logs – Raw View

Execution Logs

List | Raw | Table

Search logs...

Generate Text (Bedrock) (75c8132e-2737-4f99-a2cc-e8526182f525)


```
{
  "summary": "✅ Text generated using Amazon Titan",
  "text": "{\n    \"rows\": [\n        {\n            \"score\": \"85\",\n            \"summary\": \"Detail-oriented Data Analyst with over 3 years of experience interpreting and analyzing data to drive business solutions. Strong skills in SQL, Python, Tableau, and Excel. Proven track record of building dashboards, automating reports, and delivering insights.\"\n        }\n    ]\n}",
  "json": {
    "score": "85",
    "summary": "Detail-oriented Data Analyst with over 3 years of experience interpreting and analyzing data to drive business solutions. Strong skills in SQL, Python, Tableau, and Excel. Proven track record of building dashboards, automating reports, and delivering insights."
  },
  "raw": {
    "rows": [
      {
        "score": "85",
        "summary": "Detail-oriented Data Analyst with over 3 years of experience interpreting and analyzing data to drive business solutions. Strong skills in SQL, Python, Tableau, and Excel. Proven track record of building dashboards, automating reports, and delivering insights."
      }
    ]
  },
  "aiResult": "Text Generated : {\n    \"rows\": [\n        {\n            \"score\": \"85\",\n            \"summary\": \"Detail-oriented Data Analyst with over 3 years of experience interpreting and analyzing data to drive business solutions. Strong skills in SQL, Python, Tableau, and Excel. Proven track record of building dashboards, automating reports, and delivering insights.\"\n        }\n    ]\n}"
}
```


Rename JSON Keys (652dff86-59b7-4872-8f83-6d79991605f6)


```
{
  "summary": "✅ Extracted 1 field(s)",
  "details": {
    "mapping": {
      "text": "json.summary"
    },
```

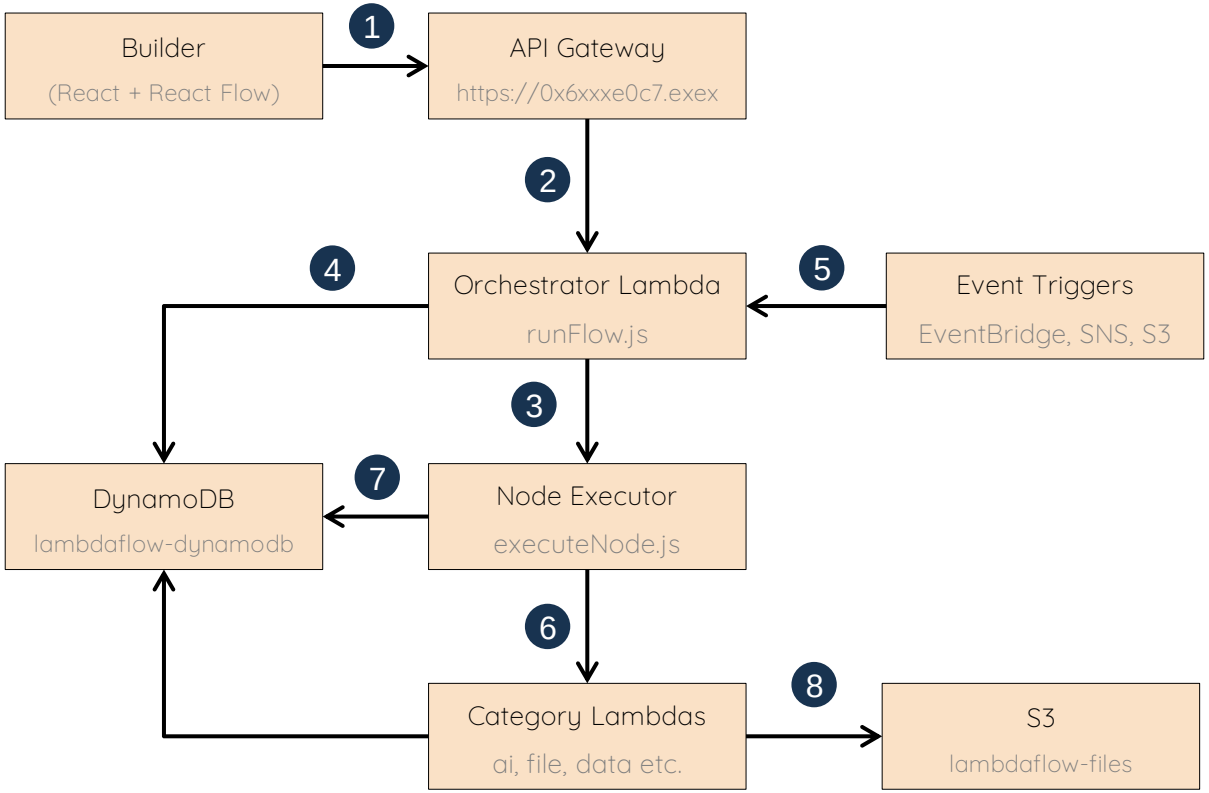

Execution Logs – List View

Architecture Overview

1. Architectural Goals

Goal	Implementation Mechanism
Developer productivity	React UI + Node.js dev backend (Express)
Cloud-native, serverless infra	AWS Lambda, API Gateway, DynamoDB, EventBridge
Visual programming interface	React Flow with drag-drop, live node editing
Fully modular & scalable	Each node = Individual Lambda function
Flexible triggering	Manual + Event-based (S3, SNS, CRON, etc.)
Easy migration	Local dev → serverless deploy via Serverless Framework
CI/CD ready	Compatible with GitHub Actions / Amplify Pipelines

Architecture Overview



1. User triggers workflows by interacting with UI
2. Receives trigger, initiates flow execution
3. Delegates execution of the current node
4. Reads/writes workflow definitions and runtime metadata
5. Receives events that trigger orchestrator flows
6. Resolves node type → invokes category Lambda function
7. May store node output or status
8. For reading/writing files

Architecture Overview

2. Environment Modes

Mode	Description
Local Development	A self-contained sandbox that runs locally. The React editor connects to a local Node.js + Express server that uses the same orchestrator as production. With IS_OFFLINE=true, each node runs via import() instead of calling Lambda, enabling instant flow execution, console logs, and breakpoints. /trigger/* routes simulate API Gateway, SNS, S3, and more. Workflow definitions are saved to DynamoDB—ready for cloud use without changes.
Cloud Development	A fully serverless, production-grade runtime. Each node category (AI, Data, Control, etc.) is deployed as a Lambda for scalable, isolated execution. The orchestrator Lambda fetches workflow definitions from DynamoDB, executes steps in order (with branching), and triggers via real AWS services like API Gateway, EventBridge, SNS, S3, and DynamoDB. The UI, hosted on AWS Amplify, updates instantly—no redeploy needed when flows change.

Architecture Overview

2. Environment Modes

Technical Execution Details

	Local Mode (offline)	Cloud Mode (AWS Lambda)
Executor switch	src/core/executor/executeNode.js checks isOffline (ENV var) and imports the node module directly.	Same file builds an InvokeCommand and calls the category Lambda whose name comes from nodeMap.
Orchestrator	Same runFlow engine as cloud, but runs inside the local Express server (src/server).	runFlow is packaged as src/lambdas/orchestrator/… and exposed through API Gateway/Schedule/SNS/S3 & friends.
Trigger endpoints	/trigger/* routes in Express mimic API Gateway, SNS, S3, etc. – great for Postman tests.	Each trigger is its own Lambda with the identical handler code, auto-wired by serverless.yml.
State / Workflow definitions	Always one source of truth – DynamoDB table LambdaFlowWorkflows. Local edits are saved instantly via /api/save-workflow.	Deployed Lambdas read the same table, so *modifying a flow does not require redeploy.*
Custom node sandbox	customdatatransformation becomes a new Function() and runs inline for super-fast feedback.	Exact same node runs inside the dataCategory Lambda; nothing to re-package.
What does require deploy?	Anything that relies on AWS-side events (cron, S3 notifications, SES→SNS) or IAM changes.	Only when you add new node types or modify Lambda memory/timeout. Flow edits are hot-swapped.

Architecture Overview

2. Environment Modes

Feature Comparison

Feature	Local Mode	Cloud Mode
Execution Engine	Node.js + Express (Local)	AWS Lambda (Fully Serverless)
Trigger Simulation	Simulated via /trigger/* routes (API, SNS, S3, etc.)	Real AWS triggers (API Gateway, EventBridge, S3, SNS, etc.)
Node Execution	import() used to execute local node logic instantly	Each node is a deployed Lambda (scalable & isolated)
Speed	Instant (no cold starts, no deploys)	Production-grade performance, slightly more latency
Debugging Support	Console logs + breakpoints for step-by-step debugging	CloudWatch logs (per-node), output visible in UI
Workflow Storage	Saves definitions to DynamoDB	Fetches definitions from the same DynamoDB table
Redeploy on Edit?	No redeploy needed — edit and run immediately	No redeploy needed — trigger Lambda fetches latest flow
Best For	Fast iteration, testing logic, debugging complex flows	Live triggers, production execution, cloud-scale workflows
UI Hosting	Localhost	AWS Amplify (auto-updates with latest flow changes)

Architecture Overview

2. Environment Modes

Local Mode (Developer Laptop)

A self-contained sandbox that runs exactly the same workflow engine you'll deploy later, but inside Node + Express.

Key points

- Instant iteration – React front-end runs on localhost, talking to the Express API for sub-second prototyping, console logging and step-by-step debugging.
- Same JS, two execution paths – executeNode.js decides whether to import() a node module or InvokeCommand a Lambda, based on the environment flag IS_OFFLINE=true. Flip the flag, restart, and the engine redirects every node call to AWS without touching your flow definition.
- Trigger emulation – Express routes under /trigger/* mimic API Gateway, SNS, S3, EventBridge, etc., so you can post raw payloads from Postman or Jest before paying for real cloud events.
- Shared source of truth – Flows are still saved in DynamoDB, meaning the IDs and metadata you create locally are immediately honoured by the cloud runtime after deployment.

Architecture Overview

2. Environment Modes

Cloud Mode (Production)

A fully serverless architecture that swaps the local sandbox for AWS-native building blocks:

- Per-node Lambdas – Each functional "category" (AI, Data, Control, …) is packaged as its own Lambda, giving horizontal scale and isolation while keeping cold-starts low.
- Central orchestrator – A lightweight Lambda invokes nodes in topological order, exactly as the Express engine does locally. No re-authoring required.
- Always-on triggers – API Gateway, EventBridge, SNS, S3 and DynamoDB streams fan in to the same /trigger/* handlers, allowing flows to wake on any AWS event.
- Persistent state & definitions – The table LambdaFlowWorkflows in DynamoDB stores every node, edge and revision. Editing a flow in the UI updates the table immediately; deployed Lambdas read it on each invocation, so changes go live without redeploys.
- Frontend on Amplify – The React editor is served via AWS Amplify, giving automatic CI/CD from your main branch and global edge caching. Designers see their modifications reflected in running Lambdas within seconds.

Architecture Overview

3. Component Architecture

Frontend (React + React Flow)

Feature	Details
Framework	React
Diagram Editor	React Flow with custom node types
Node Config Editor	Dynamic MUI-based modal inputs generated from nodeSpec
Workflow Engine Integration	Sends API requests to trigger workflows
Execution Results UI	Drawer showing status/success/error per node with expandable JSON
Deployment	Hosted on AWS Amplify, built with npm run build
Auth Support (optional)	AWS Amplify Auth / Cognito

Architecture Overview

3. Component Architecture

Backend (Node.js + Express – Local Mode)

Feature	Details
Local Workflow API	POST /run, GET /workflows, POST /save, etc.
Node Execution	Each node imported as a module and executed in sequence
Trigger Simulation	Manual trigger endpoints for S3, API Gateway, etc.
Execution Engine	runFlow(graph) function walks DAG and handles branching
File I/O, Logs	Supports base64 file read/write and per-node execution result logs
NOT deployed to cloud	Used only in local dev for full simulation & debugging

Architecture Overview

3. Component Architecture

Cloud Infrastructure (Serverless Deployment)

Component	Description
Lambda Functions	Each node type is its own Lambda function (e.g., summarizeText, classifyText)
Trigger Functions	Trigger Lambdas for S3, SNS, EventBridge, API Gateway, DynamoDB Streams
Workflow Orchestrator	runFlow Lambda which invokes all workflow nodes in topological order
API Gateway (Optional)	Used for exposing triggerAPI, listWorkflows, etc. via REST
DynamoDB	Stores all workflows as { id, name, nodes[], edges[] }
Amplify Hosting	Hosts frontend React app (CI/CD, versioning, environment branches)

Architecture Overview

4. Workflow Execution Lifecycle

Local (Development Mode)

React UI
localhost:3000

Visual workflow editor running locally.

↓

Express API
localhost:4000

Serves as the backend to parse workflows and simulate execution.

↓

runFlow()
In-memory Executor

DAG-based executor handling node execution and branching logic.

↓

Local Node Functions
JS Modules

Individual node implementations executed as local JS functions.

↓

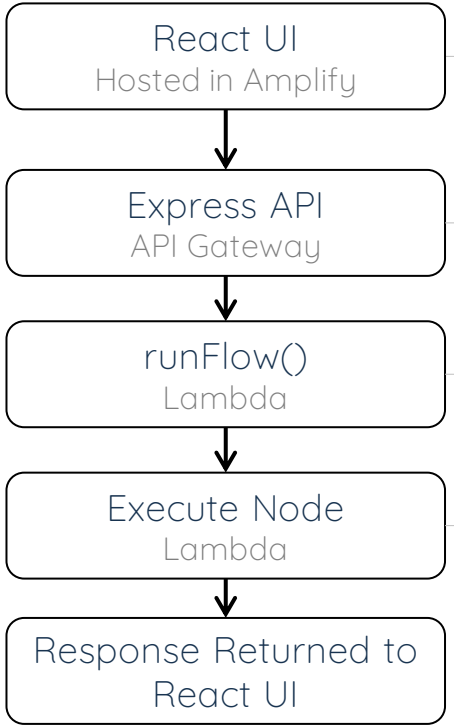
Response Returned to React UI

Optional persistence: Defaults to in-memory, but can use DynamoDB if needed.

Architecture Overview

4. Workflow Execution Lifecycle

Production (AWS Cloud)



Deployed React app for workflow creation, editing, and monitoring.

Entry point for all execution and trigger-based invocations.

Central dispatcher that interprets workflows and invokes nodes.

Each node is a dedicated Lambda function executed on-demand.

Persistence in DynamoDB : Stores workflow definitions, runtime states, and execution metadata.

Architecture Overview

5. runFlow() DAG Execution Engine

- Accepts a workflow object with:

  nodes[] : Each with id, type, parameters

  edges[] : Directed connections defining graph
- Builds topological sort of the DAG
- For each node:
  - Passes input from predecessors
  - Calls either:

    Local module (in dev)

    invokeLambda (in production)
- Collects output into execution context

```
{
  "nodeA": { status: "success", output: {...} },
  "nodeB": { status: "error", error: {...} }
}
```

Architecture Overview

6. Supported Triggers

Triggers

Trigger Type	AWS Service	Lambda Handler	Event Source Type	Use Case
API Gateway Trigger	API Gateway	triggerAPI.handler	HTTP	Run on HTTP call
Schedule Trigger	EventBridge (CRON)	triggerEventBridge.handler	Schedule	Scheduled flows
EventBridge Trigger	EventBridge (Custom)	triggerEventBridge.handler	Event Bus	Event-driven integrations
S3 Trigger	S3 Bucket Events	triggerS3.handler	Storage	Run on file upload
SNS Trigger	SNS Subscription	triggerSNS.handler	Pub/Sub	Notification-based triggers
DynamoDB Trigger	DynamoDB Streams	triggerDynamoDB.handler	Data Stream	Data change triggers

Architecture Overview

7. Nodes & Supported Capabilities

AI/ML

	AWS Service	Purpose	Output Format	Category
Summarize Text	Bedrock (Claude)	NLP summarization	Text	AI/ML
Classify Text	Bedrock	Text classification	Label	AI/ML
Extract Entities	Bedrock	Entity detection	JSON Entities	AI/ML
Sentiment Analysis	Bedrock	Sentiment scoring	Score/Label	AI/ML
Generate Text	Bedrock	Prompt-based generation	Text	AI/ML
Text To Speed	Polly	Audio generation	Audio (MP3)	AI/ML
Image Analysis	Textract	OCR from document images	JSON Text Blocks	AI/ML
Document OCR	Rekognition	Object/face detection	JSON Labels	AI/ML

Architecture Overview

7. Nodes & Supported Capabilities

Control Flow

	AWS Service	Purpose	Output Format	Category
Condition	N/A	Conditional Branching	Boolean or Path	Control Flow
Loop / For Each	N/A	Iterate over array items	Each item as input	Control Flow
Delay / Wait	N/A	Pause Execution	Pass-through input	Control Flow
Terminate	N/A	End the workflow	None	Control Flow

Architecture Overview

7. Nodes & Supported Capabilities

Data Transformation

	AWS Service	Purpose	Output Format	Category
Custom Lambda	Lambda	Custom logic execution	Varies	Data Transformation
Build JSON	N/A	Construct custom JSON	JSON Object	Data Transformation
Merge JSON	N/A	Merge JSON Inputs	JSON Entities	Data Transformation
Parse CSV	N/A	Convert CSV to JSON	Score/Label	Data Transformation
Rename JSON Keys	N/A	Rename keys in JSON	Text	Data Transformation
Flatten JSON	N/A	Flatten Nested JSON	Audio (MP3)	Data Transformation
JSON to CSV	N/A	Convert JSON to CSV	JSON Text Blocks	Data Transformation
Extract Field	N/A	Extract fields from JSON	JSON Labels	Data Transformation

Architecture Overview

7. Nodes & Supported Capabilities

Data Transformation

	AWS Service	Purpose	Output Format	Category
Filter Data	N/A	Filter data by condition	Filtered array	Data Transformation
Text Template	N/A	Dynamic templating	Rendered String	Data Transformation
String Formatter	N/A	Format text	Formatted string	Data Transformation
Generate UUID	N/A	Create unique IDs	UUID String	Data Transformation
Generate PDF	N/A	Generate PDF from HTML	PDF File (Base64)	Data Transformation

Architecture Overview

7. Nodes & Supported Capabilities

File & Storage

	AWS Service	Purpose	Output Format	Category
Read File	N/A	Read Local File	File Content	File & Storage
Write File	N/A	Write Local File	Success Flag	File & Storage
Append to File	N/A	Append to local file	Success Flag	File & Storage
Upload to S3	S3	Upload file to S3	S3 URL	File & Storage
Download from S3	S3	Download file from S3	File content	File & Storage
List S3 Files	S3	List S3 bucket files	Array of filenames	File & Storage
Put Item (Dynamo DB)	DynamoDB	Insert into DynamoDB	Success Flag	File & Storage
Get Item (Dynamo DB)	DynamoDB	Get item from DynamoDB	Retrieved item	File & Storage

Architecture Overview

7. Nodes & Supported Capabilities

Integrations & APIs

	AWS Service	Purpose	Output Format	Category
HTTP Request	N/A	Send HTTP Request	HTTP Response	Integrations & APIs
GraphQL Request	N/A	Perform GraphQL query	GraphQL response	Integrations & APIs
CRM Sync	N/A	Sync with CRM	Success response	Integrations & APIs
Send to Slack	Slack	Post message to Slack	Message ID / Status	Integrations & APIs

Architecture Overview

7. Nodes & Supported Capabilities

Notifications

	AWS Service	Purpose	Output Format	Category
Send Email (SES)	SES	Send Email	Delivery Status	Notifications
Send SMS	Twilio	Send SMS	Message Status	Notifications
Push Notification	N/A	Push web / mobile alert	Notification Status	Notifications

Architecture Overview

8. Workflow Storage Schema (DynamoDB)

Attribute	Type	Description
workflowId	String	Unique workflow ID
nodes	JSON	Node configuration array
edges	JSON	Edge (connection) array
createdAt	ISODate	Timestamp

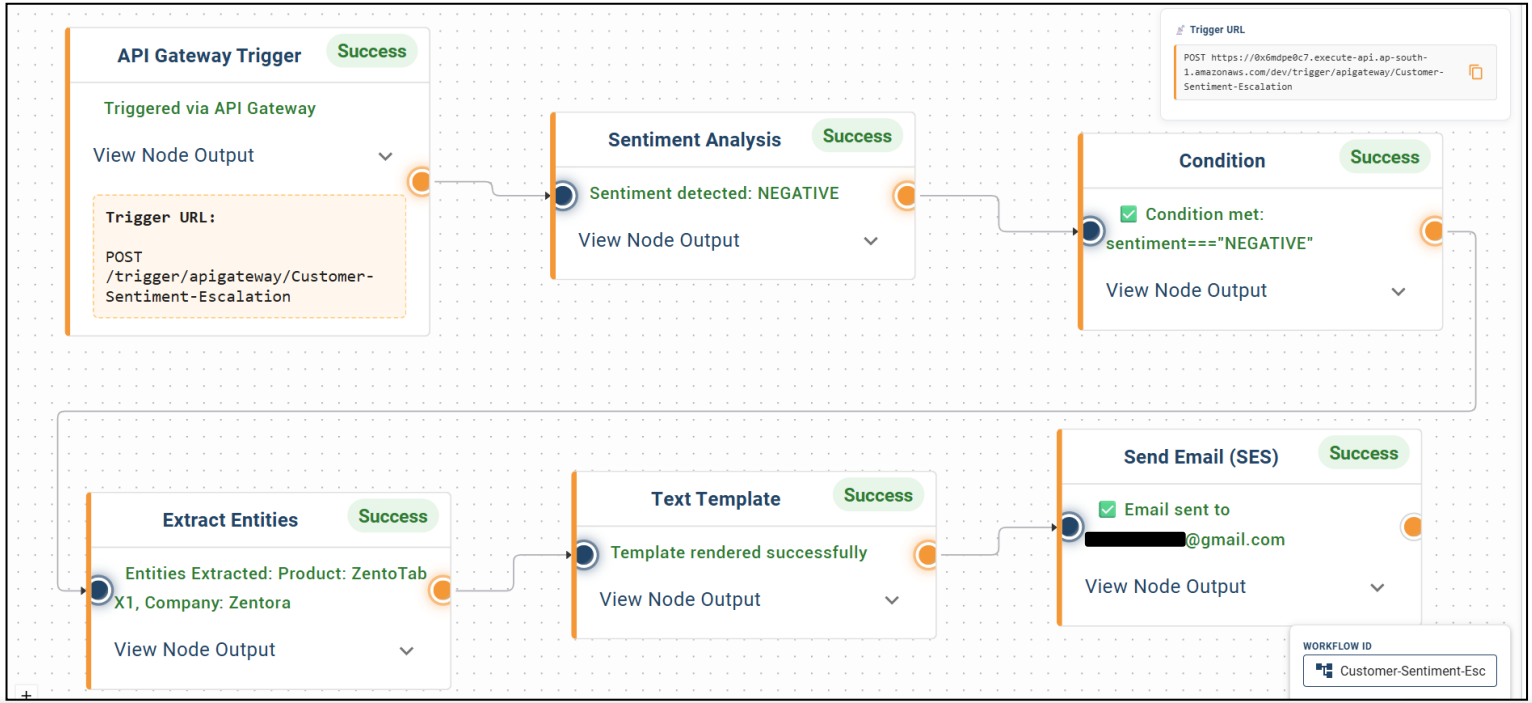
Attributes View DynamoDB JSON


```
1  {
2    "workflowId": "upload-summary",
3    "createdAt": 1750709632637,
4    "edges": [
5      {...},
17     {...},
29     {...},
41     {...}
53   ],
54   "nodes": [
55     {...},
84     {...},
124    {...},
172    {...},
202    {...}
232  ]
233 }
```

Use Cases / Example Workflows

1. Customer Sentiment Escalation Flow

Goal: Automatically analyze user feedback and escalate issues if sentiment is negative—alert the team via E-mail in real-time.



Use Cases / Example Workflows

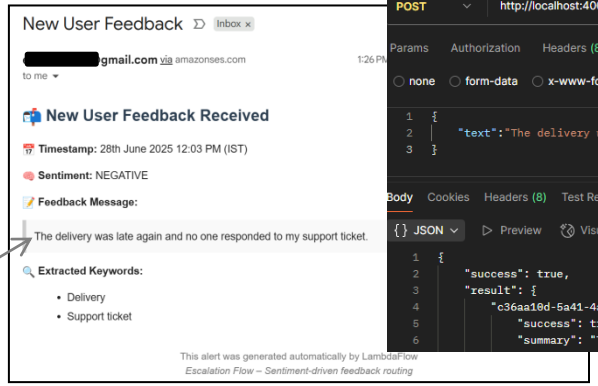
1. Customer Sentiment Escalation Flow

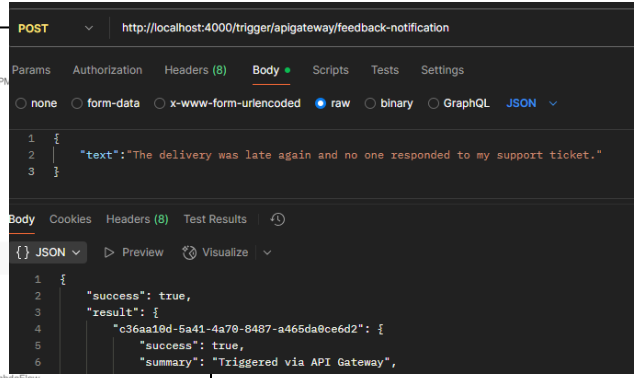
Workflow Overview

Step	Node Type	Description
1	API Gateway Trigger	Triggers the workflow when feedback is received via API Gateway
2	Sentiment Analysis	Analyzes the sentiment of the user feedback using AI
3	Condition	Checks if the sentiment is negative
4a	Extract Entities	*(Yes branch)* Extracts keywords or entities from the complaint text
5a	Text Template	*(Yes branch)* Formats a Email message including extracted issue terms
6a	Send Email (SES)	*(Yes branch)* Sends the formatted message to a Slack channel

Highlights

- Intelligent Branching (Condition-based flow)
- AI-Powered Feedback Analysis (Bedrock)
- E-mail Integration for Real-Time Alerts
- Fully Serverless & Scalable



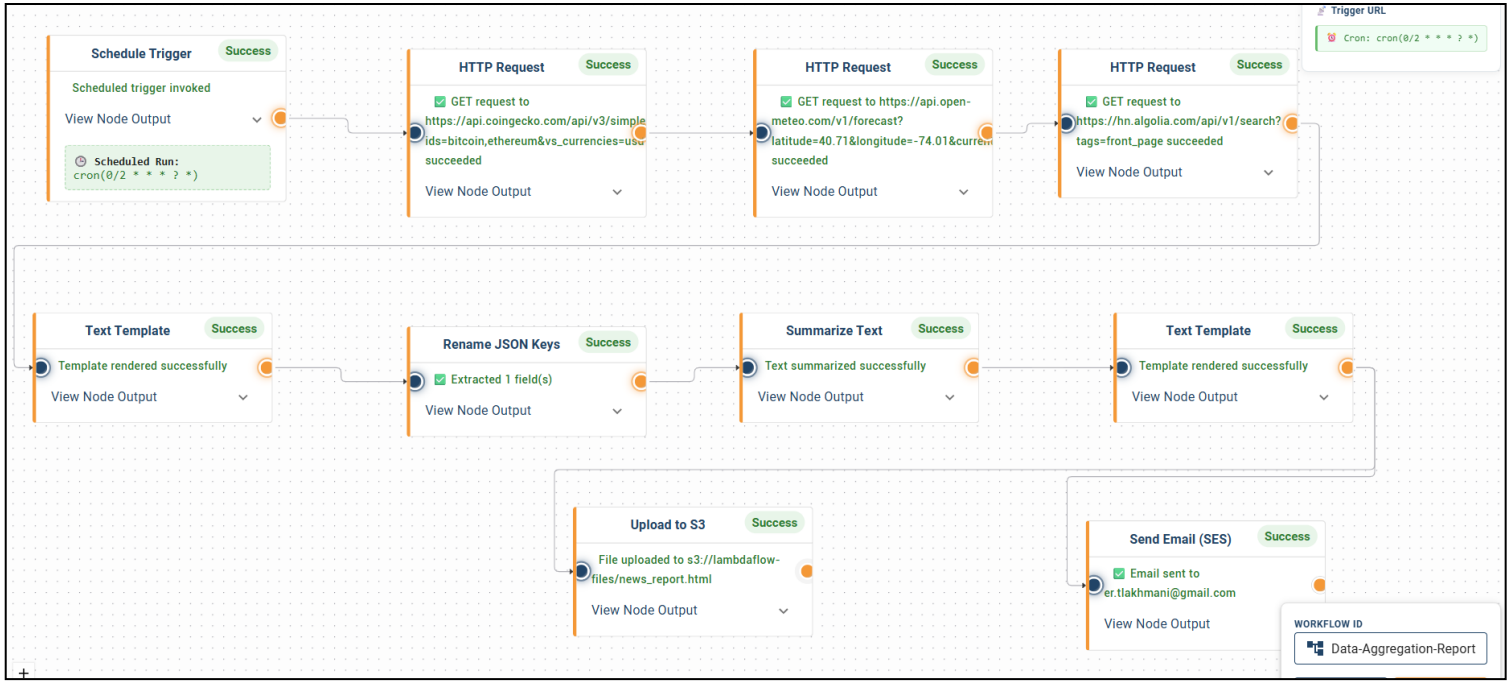


Output

Use Cases / Example Workflows

2. Dynamic Data Aggregator & Summary Report

Goal: Automate daily collection of key public data (crypto, weather, news), generate a professional report, and deliver it via Slack / Email.

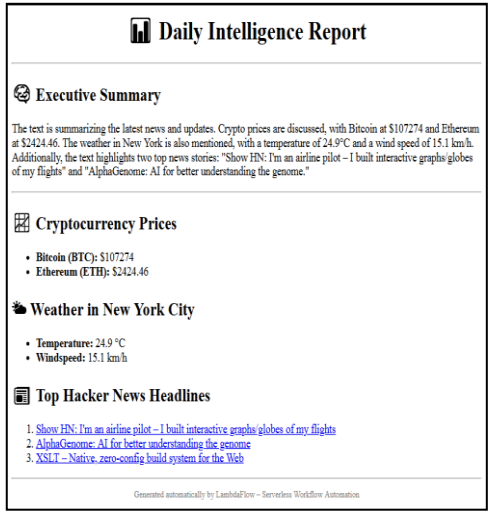


Use Cases / Example Workflows

2. Dynamic Data Aggregator & Summary Report

Workflow Overview

Step	Node Type	Description
1	Schedule Trigger	Trigger flow daily (e.g. 9 AM)
2	HTTP Request – Crypto	Fetch BTC & ETH prices from CoinGecko
3	Http Request – Weather	Fetch NYC weather from Open-Meteo
4	HTTP Request – News	Fetch headlines from Hacker News API
5	Text Template – AI Input	Prepare clean text summary of all data
6	Summarize Text – Bedrock	Use Claude (Bedrock) to generate executive summary
7	Text Template – HTML Report	Build styled HTML for report using summary + data
9	Upload S3	Upload PDF to secure S3 bucket
10	Send in Mail / Slack	Notify via Email / Slack with summary



Daily Intelligence Report

Executive Summary

The text is summarizing the latest news and updates. Crypto prices are discussed, with Bitcoin at $107274 and Ethereum at $2424.46. The weather in New York is also mentioned, with a temperature of 24.9°C and a wind speed of 15.1 km/h. Additionally, the text highlights two top news stories: "Show HN: I'm an airline pilot – I built interactive graphs/globes of my flights" and "AlphaGenome: AI for better understanding the genome."

Cryptocurrency Prices

- **Bitcoin (BTC):** $107274
- **Ethereum (ETH):** $2424.46

Weather in New York City

- **Temperature:** 24.9 °C
- **Windspeed:** 15.1 km/h

Top Hacker News Headlines

1. Show HN: I'm an airline pilot – I built interactive graphs/globes of my flights
2. AlphaGenome: AI for better understanding the genome
3. XSLT – Native, zero-config build system for the Web

Generated automatically by LambdaFlow – Serverless Workflow Automation

Output

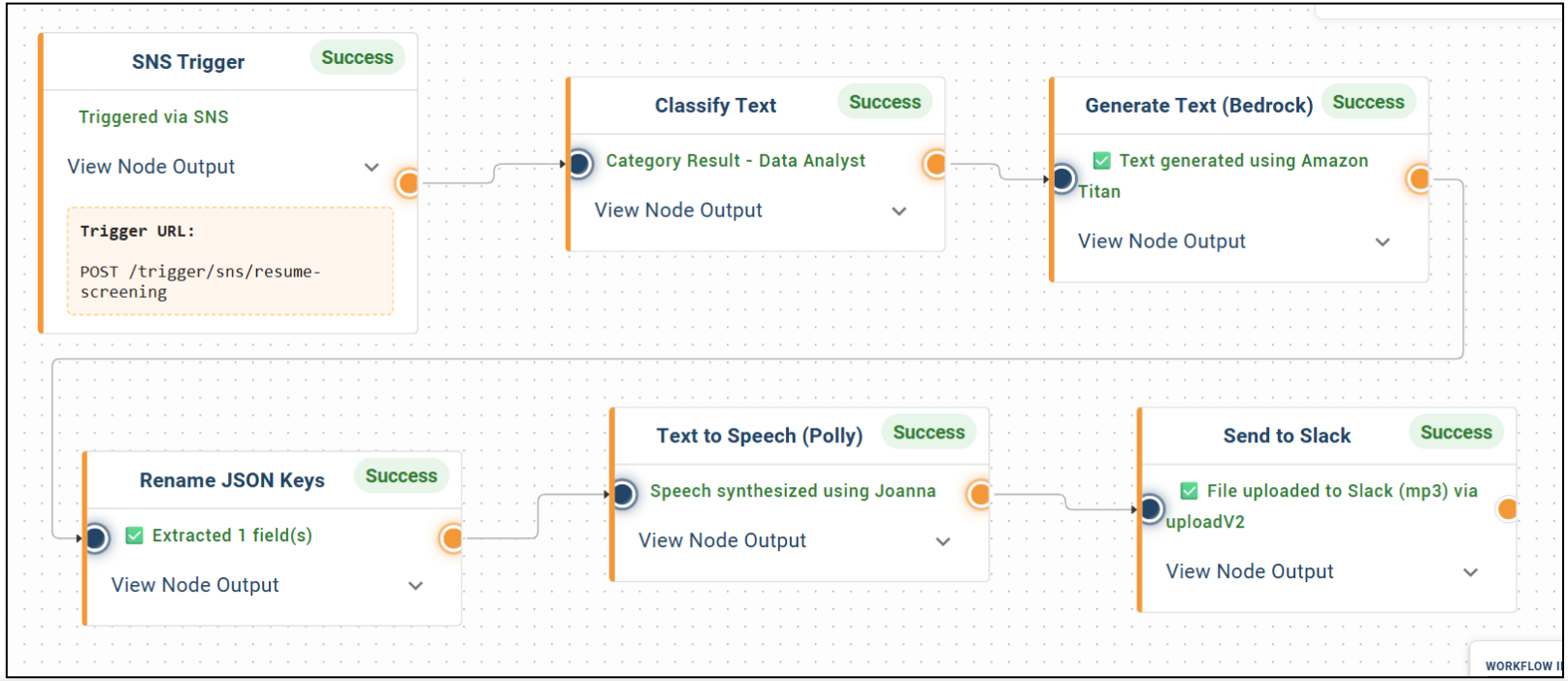
Highlights

- Uses Text Template for flexible summary input
- AI summarization via Bedrock
- Rich HTML → PDF conversion
- Email delivery with summary

Use Cases / Example Workflows

3. Resume Screening & Summary Notifying Tool

Goal: Automatically process incoming resumes to extract key details, generate a summary with a fit score, and notify the recruitment team via text and audio for faster candidate review..



Use Cases / Example Workflows

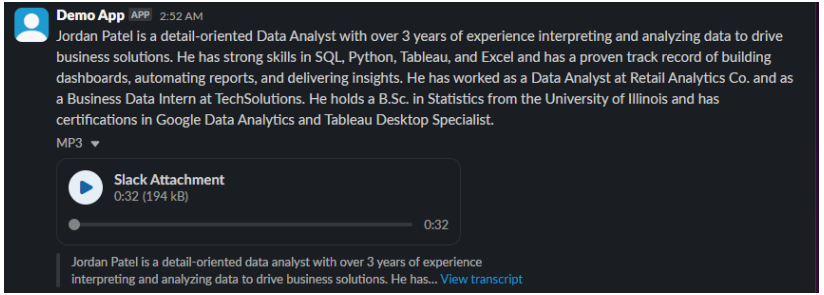
3. Resume Screening & Summary Notifier

Workflow Overview

Step	Node Type	Description
1	SNS Trigger	Triggered when a new resume email is received via SES → SNS.
2	Download from S3	Download the attached resume PDF from S3.
3	Classify Text	Uses an LLM to classify the resume into a role category (e.g., "Data Analyst", "Software Engineer", etc.).
4	Generate Text (Bedrock)	Generates a 1-paragraph AI summary tailored to the classified role, highlighting experience, skills, and fit.
5	Text to Speech (Polly)	Converts the generated summary into audio (MP3) using Amazon Polly.
6	Send to Slack	Sends a Slack message to the recruitment team with: The text summary, Role classification result A playable link to the voice summary (MP3)

Highlights

- AI-powered classification & summarization
- Voice-enhanced summaries for quick screening
- All resumes auto-processed and logged
- Slack delivery with text + audio = enhanced recruiter experience

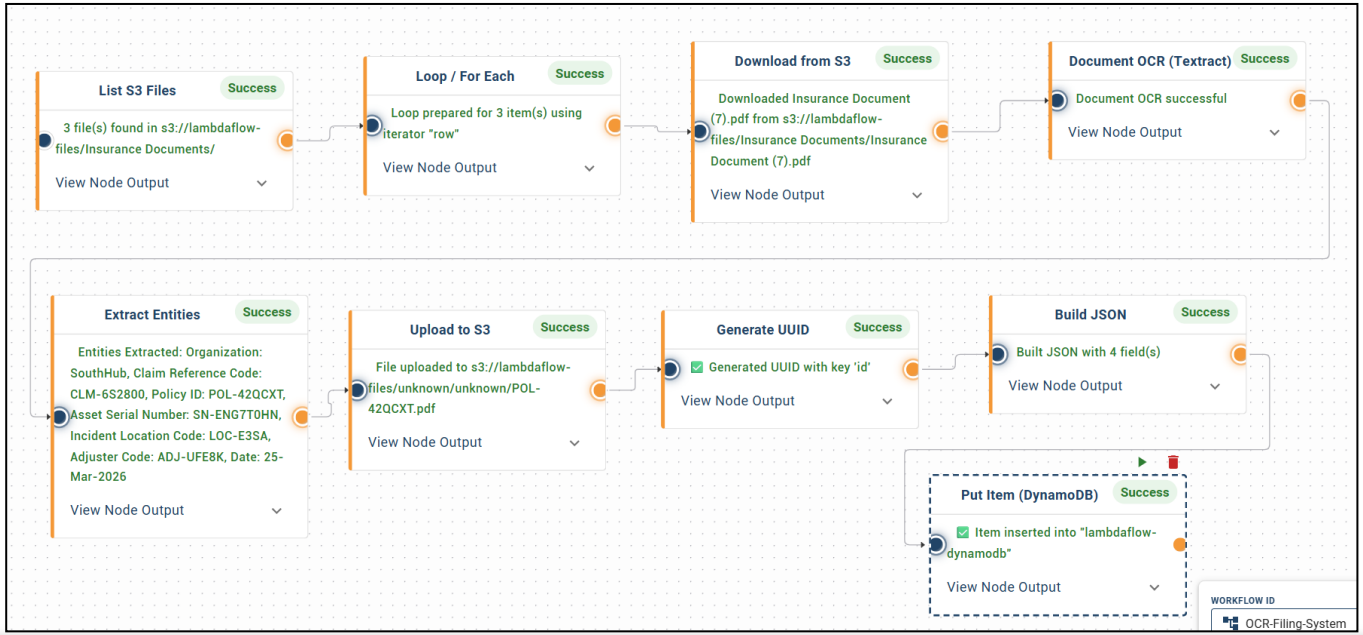


Output

Use Cases / Example Workflows

4. Auto OCR & Entity-Driven Filing System

Goal: Automatically process scanned documents by performing OCR, extracting key entities, organizing them in S3 folders based on entities , and logging structured metadata to DynamoDB for easy tracking, auditing & search - without any manual tagging.



Use Cases / Example Workflows

4. Auto OCR & Entity-Driven Filing System

Workflow Overview

Step	Node Type	Description
1	List S3 Files	Retrieve a list of newly uploaded scanned documents from the source S3 bucket
2	Loop / For Each	Loop through each document file individually
3	Document OCR	Use Amazon Textract to extract raw text from the document
4	Extract Entities	Identify key fields (e.g., Name, Company, Date) using Bedrock NER
5	Upload To S3	Save the file to its organized folder path in S3 based on extracted values
6a	Put Item (DynamoDB)	Log metadata (e.g., filename, folder path, entities) into DynamoDB

Highlights

- Textract + Bedrock integration for advanced document understanding
- Entity-based smart routing of documents into S3 folders
- DynamoDB logging enables structured, queryable, and auditable document metadata
- Fully automated pipeline — no human input required

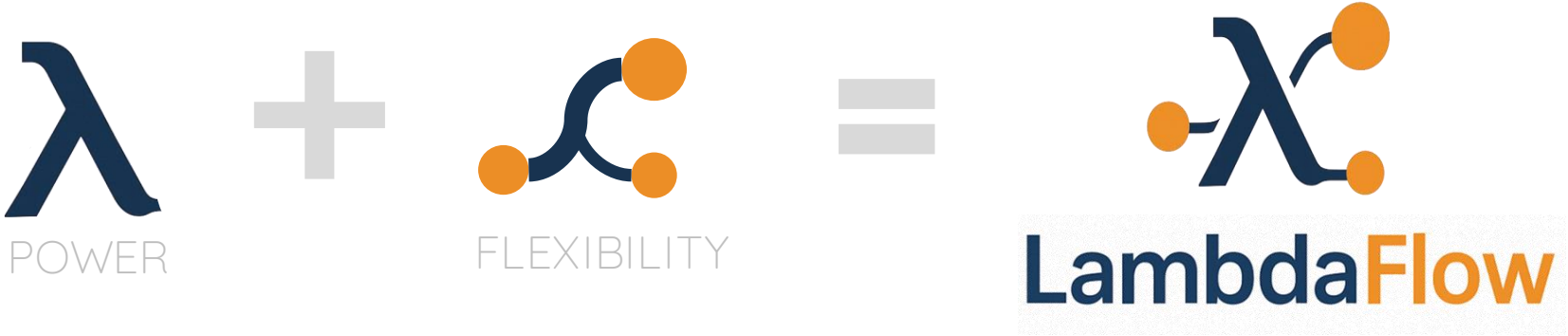
Attributes View DynamoDB JSON


```
{
 "id": "89f9b3c6-1e31-44a2-a082-ecebf96816fa",
 "bucket": "lambdaflow-files",
 "entities": {
  "Adjuster Code": "ADJ-CWMX1",
  "Asset Serial Number": "SN-WQJ4NNGP",
  "Claim Reference Code": "CLM-RJU0L8",
  "Incident Location Code": "LOC-00GS",
  "Organization": "Nimbus Insurance",
  "Policy ID": "POL-JTEEHQ",
  "Team": "CentralOps"
 },
 "key": "unknown/unknown/POL-JTEEHQ.pdf"
}
```


Name
Fire/
Insurance Documents/
Theft/
unknown/

Output

LambdaFlow isn't just a workflow tool — it's what serverless was always meant to be: powerful, flexible, and beautifully simple.



For additional information, please refer to the architectural documentation included in the Github repository