SynthDilogQA: Automated Testing for Conversational AI

Automated Testing Pipeline for Conversational AI Systems — Customer Deployment Guide

AWS Marketplace | Container Product | Amazon MWAA | ECS Fargate

Overview

SynthDilogQA automates the end-to-end testing of conversational AI systems. Rather than hand-writing test cases, teams upload their Standard Operating Procedures (SOPs) or playbooks to S3, and the pipeline automatically generates hundreds of realistic dialog scenarios that validate whether an agent answers correctly, handles edge cases, and follows business rules.

The pipeline processes your SOP files, extracts business rules and variables, generates complex multi-turn test conversations, and delivers a complete QA dataset (JSONL/CSV) back to S3, ready for testing and evaluating your conversational AI agent.

What You Provide

- An existing **Amazon MWAA** environment (Airflow 2.x or later).
- An existing **S3 bucket** for SOP uploads and pipeline outputs.
- An existing **RDS database** (PostgreSQL or MySQL) with a pre-created **Secrets Manager secret** holding the credentials.
- A **VPC** with at least two private subnets that have NAT Gateway access — the same VPC as your MWAA environment.
- A **security group** for ECS Fargate tasks with appropriate inbound and outbound rules configured for your VPC.

SOP Document Structure

The pipeline processes Standard Operating Procedure (SOP) documents to extract business logic, rules, variables, and tool structures for test scenario generation. To ensure high parsing accuracy by the pipeline's LLM engine, the uploaded SOP files must be saved as UTF-8 encoded plain text (`.txt` or `.md`) files and adhere to a standard corporate SOP structure.

A standard corporate SOP document is composed of the following sections:

1. **Document Control:** A metadata block containing identifiers, version information, effective dates, ownership details, and department association.
2. **Purpose and Scope:** A high-level description of what the procedure achieves and who/what is within the scope of execution.
3. **Roles and Responsibilities:** Defines the entities involved (e.g. customer, agent/automated bot, system APIs) and their roles.
4. **Procedure:** Step-by-step description of the sequential workflow steps, validations, data collection, and tool usage.
5. **Exception Handling:** Explicit procedures for failed validation, error scenarios, or API integration errors.
6. **Technical Specifications:** Standard OpenAPI 3.0 YAML schemas defining any backend tools or APIs that are referenced and called during the workflow.

SOP Format Template

Below is the standard corporate template for the SOP document. Replace the placeholder sections (in brackets) with your actual process flow and API definitions. Emojis and conversational labels must be excluded from structural headers to maintain formatting cleanliness.

```
SOP-[ID]: [SOP Title]

1. Document Control
- SOP ID: [SOP-XXX-###]
- Title: [Process Title]
- Version: [Version Number]
- Department: [Department Name]
- Effective Date: [YYYY-MM-DD]
- Owner: [Owner Role or Name]

2. Purpose and Scope
[Describe the purpose, objective, and scope of this procedure.]

3. Roles and Responsibilities
- [Role Name 1]: [Description of what this role is responsible for]
- [Role Name 2]: [Description of what this role is responsible for]

4. Procedure
4.1 [Sub-Process A Title]
- Step 1: [Action description, details of user information to collect, and input rules]
- Step 2: [Action description, including calling tools/APIs]
- Step 3: [Subsequent process step description]

4.2 [Sub-Process B Title]
- Step 1: [Prerequisite checks and action description]
- Step 2: [Action description, including calling tools/APIs]

5. Exception Handling
- [Scenario 1 (e.g. Validation Error)]: [Describe error path, user notifications, and fallback
action]
- [Scenario 2 (e.g. API/Tool Failure)]: [Describe error path, user notifications, and fallback
action]

6. Technical Specifications (OpenAPI Schema)
[Standard OpenAPI 3.0 YAML definitions for APIs referenced in the procedure]
```

Prerequisites

AWS Account Requirements

- Permissions to deploy CloudFormation stacks, create IAM roles, ECS services, and MWAA environments.
- An existing **S3 bucket** with **Versioning Enabled**. This bucket will hold your input documents, DAG code, and pipeline artifacts.
- An existing **RDS** instance reachable from your VPC (PostgreSQL or MySQL).
- A pre-created **Secrets Manager secret** containing your RDS credentials — see Step 3 below.
- At least **two private subnets** with NAT Gateway routes in different Availability Zones for the ECS and MWAA instances.
- **Amazon Bedrock model access** enabled for Anthropic Claude Sonnet in your AWS account and region.

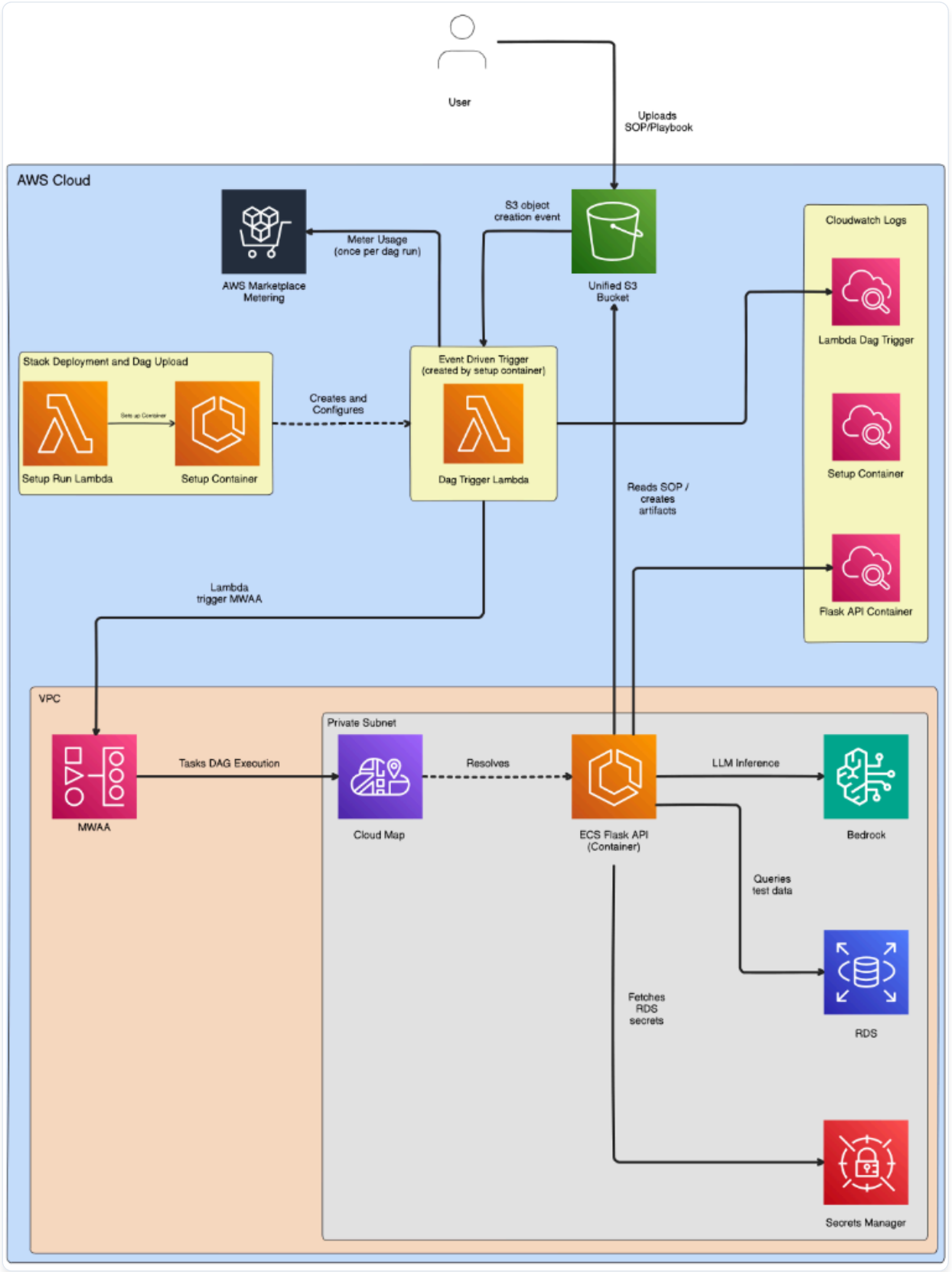
Security Group for ECS

Create (or identify) a security group for the ECS Fargate tasks. Ensure it allows inbound traffic from your MWAA worker security group and outbound HTTPS access for AWS service calls. Contact your network administrator if you need assistance configuring security group rules.

Local Tools

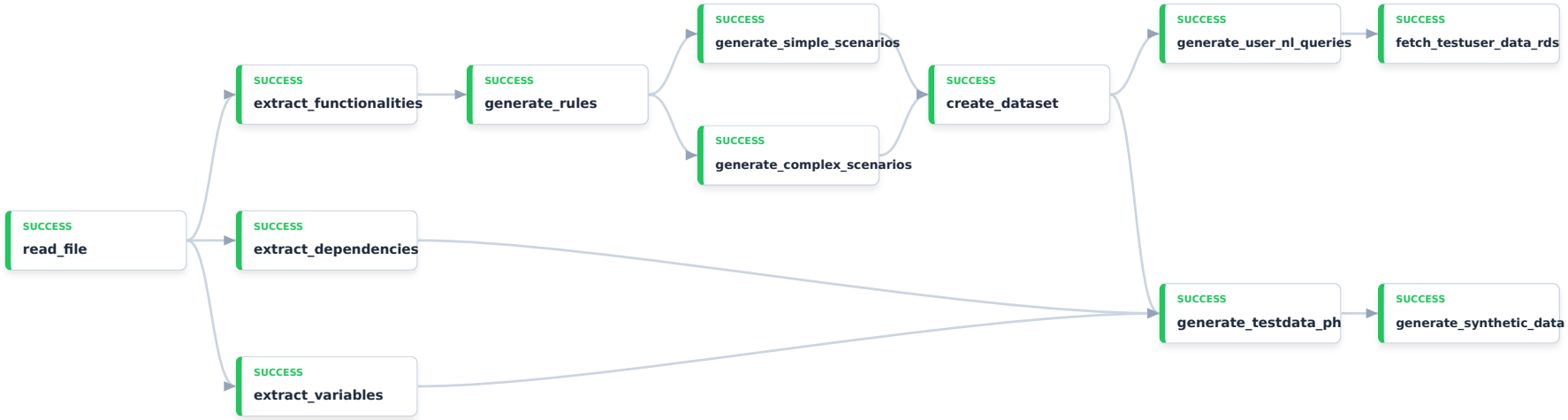
- AWS CLI v2 with credentials configured, or access to the AWS Management Console.

Architecture



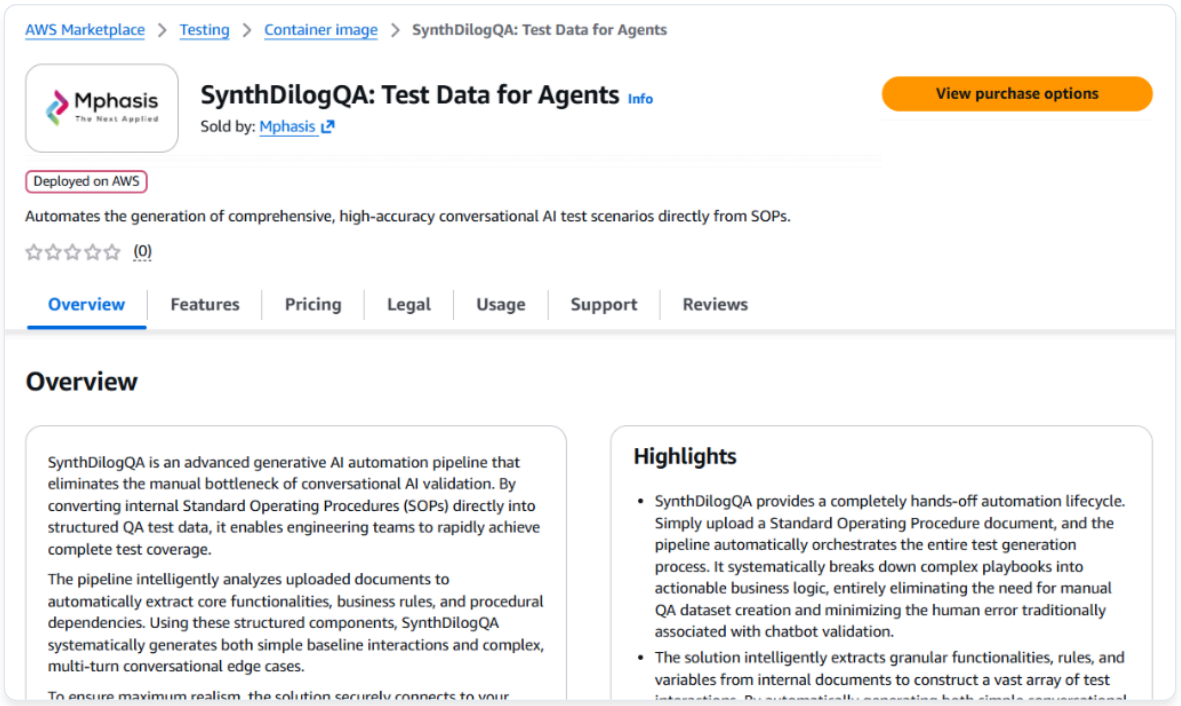
Airflow DAG Orchestration Flow

The `sop_upload_synth` DAG orchestrates the pipeline through parallel and dependent tasks, communicating with the ECS Flask API at each step. Below is the workflow layout of the DAG:

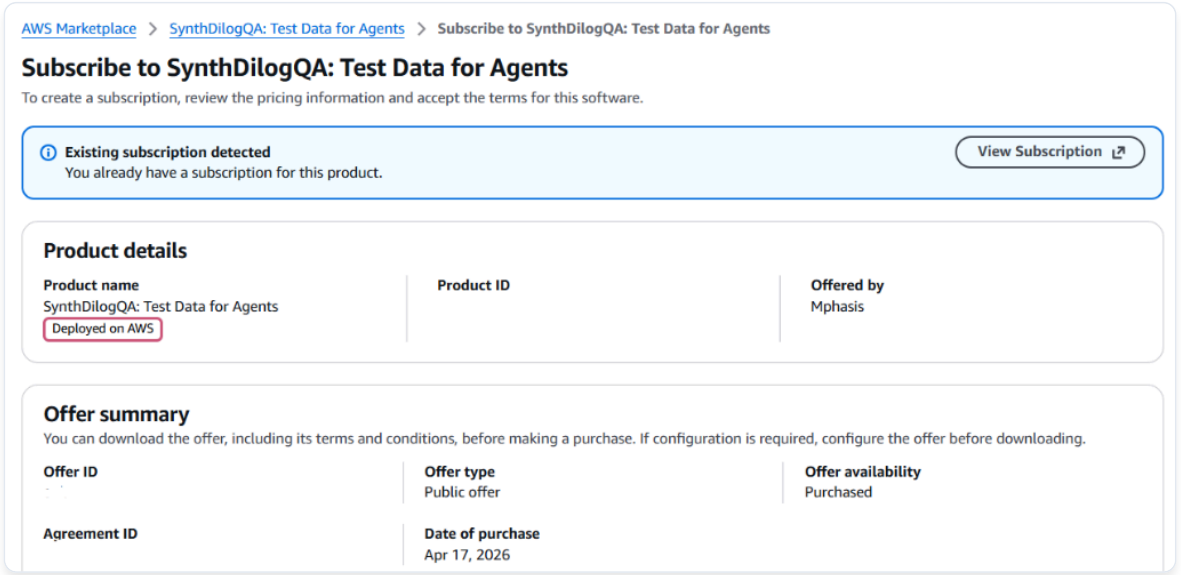

SUCCESS
read_file
SUCCESS
extract_functionalities
SUCCESS
extract_dependencies
SUCCESS
extract_variables
SUCCESS
generate_rules
SUCCESS
generate_simple_scenarios
SUCCESS
generate_complex_scenarios
SUCCESS
create_dataset
SUCCESS
generate_user_nl_queries
SUCCESS
fetch_testuser_data_rds
SUCCESS
generate_testdata_ph
SUCCESS
generate_synthetic_data

Step 1 — Subscribe on AWS Marketplace

1 **Find the product** on AWS Marketplace and click *Continue to Subscribe*.



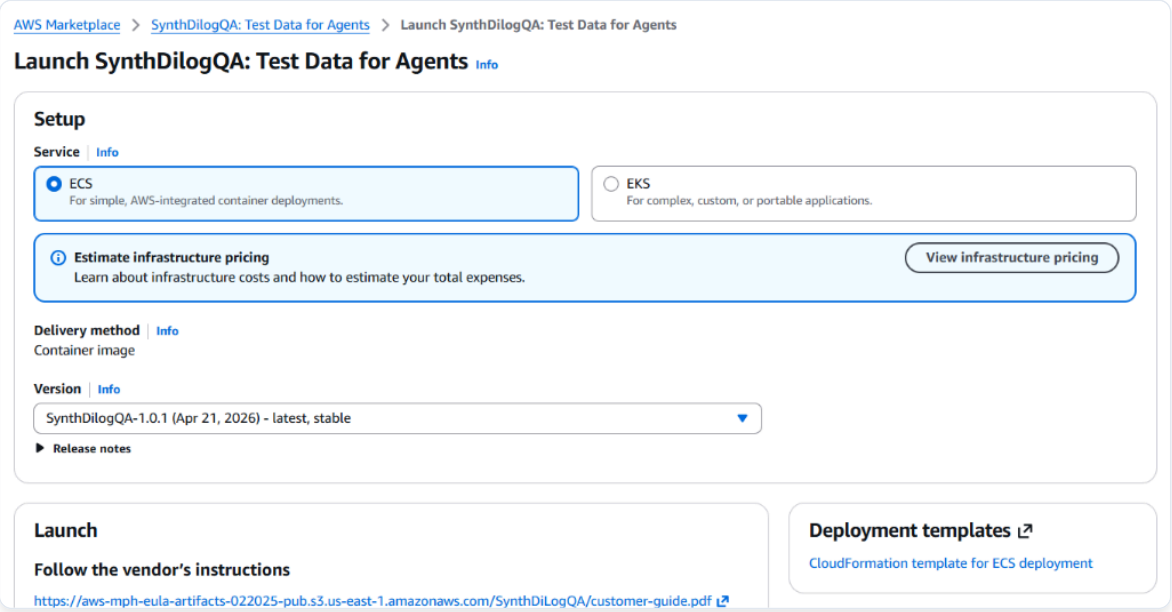
2 Accept the End User License Agreement and click *Continue to Configuration*.



3 Note the **Container Image URI** shown on the configuration page. You will enter this as the `ContainerImageUri` CloudFormation parameter in Step 4.

> 💡 **Setup Image URI:** AWS Marketplace displays only the primary `ContainerImageUri`. Construct the `SetupImageUri` by using the same ECR registry path but replacing `mphasis/synthdilogqa` with `mphasis/synthdilog-setup` (e.g., if the primary URI is `709825985650.dkr.ecr.us-east-1.amazonaws.com/mphasis/synthdilogqa:1.0.0`, your setup URI will be `709825985650.dkr.ecr.us-east-1.amazonaws.com/mphasis/synthdilog-setup:1.0.0`). Both ECR repos are accessible using the same Marketplace ECR login token.

4 Click *Continue to Launch → Launch CloudFormation* to open the CloudFormation console with the template pre-loaded, or deploy manually using the CloudFormation template.



Step 2 — Set Up Amazon MWAA

SynthDilogQA uses **Amazon Managed Workflows for Apache Airflow (MWAA)** to orchestrate the pipeline. You must have a running MWAA environment **before** deploying the CloudFormation stack. If you already have one, skip to Step 3 and note your environment name.

> ℹ️ Creating a new MWAA environment takes **20–30 minutes**. Plan accordingly before you proceed to the CloudFormation deployment.

Create an MWAA Environment (Console)

1. Open the **Amazon MWAA** console and click *Create environment*.

2. **Name** — give it a name (e.g. `synthdilogqa-mwaa`). You will enter this exact name as the `MWAAEnvName` CloudFormation parameter later.

3. **Airflow version** — select **2.9** or later.

4. **S3 bucket for DAG storage** — choose an existing S3 bucket (can be the same bucket you use for SOP documents) and set the DAG folder to `dags/`. The orchestration DAG will be uploaded to this folder automatically during deployment.

5. **Networking**:
   - Select the **same VPC** you will use for the ECS Fargate service.
   - Select at least **two private subnets** in different Availability Zones.
   - Choose or create a **security group** for MWAA workers with appropriate outbound rules to reach ECS tasks.

6. **Environment class** — `mw1.small` is sufficient for most workloads. Scale up if you process many large SOP documents concurrently.

7. **Requirements file** — create a requirements file with the required libraries and upload it to your MWAA S3 bucket. Point the *Requirements file* field to its S3 path.

8. Leave all other settings at their defaults and click *Create environment*. Wait for the status to change to **Available**.

Find Your Environment Name

Once the environment is **Available**, note the exact environment name from the MWAA console — it is case-sensitive. Enter this as the `MWAAEnvName` CloudFormation parameter.

Step 3 — Prepare Your RDS Secrets Manager Secret

The ECS container reads your RDS credentials at runtime from an existing Secrets Manager secret. You must create this secret **before** deploying the CloudFormation stack and provide its ARN as the `DBPasswordSecretArn` parameter.

> 🔒 The secret must use **exactly** these JSON key names — the application will fail to connect to the database if any key is missing or mis-spelled.

Required Secret JSON Structure

```
{
  "DB_TYPE":     "postgresql",   <!-- or "mysql" -->
  "DB_USER":     "your_username",
  "DB_PASSWORD": "your_password",
  "DB_HOST":     "your-db.cluster-xxxx.us-west-2.rds.amazonaws.com",
  "DB_PORT":     "5432",         <!-- use "3306" for MySQL -->
  "DB_NAME":     "your_database_name"
}
```

Create the Secret (AWS CLI)

```
aws secretsmanager create-secret \
  --name "synthdilogqa/rds-credentials" \
  --secret-string '{
    "DB_TYPE":     "postgresql",
    "DB_USER":     "dbadmin",
    "DB_PASSWORD": "<YOUR-PASSWORD>",
    "DB_HOST":     "my-db.cluster-xxxx.us-west-2.rds.amazonaws.com",
    "DB_PORT":     "5432",
    "DB_NAME":     "synthdilogqa"
  }' \
  --region us-west-2
```

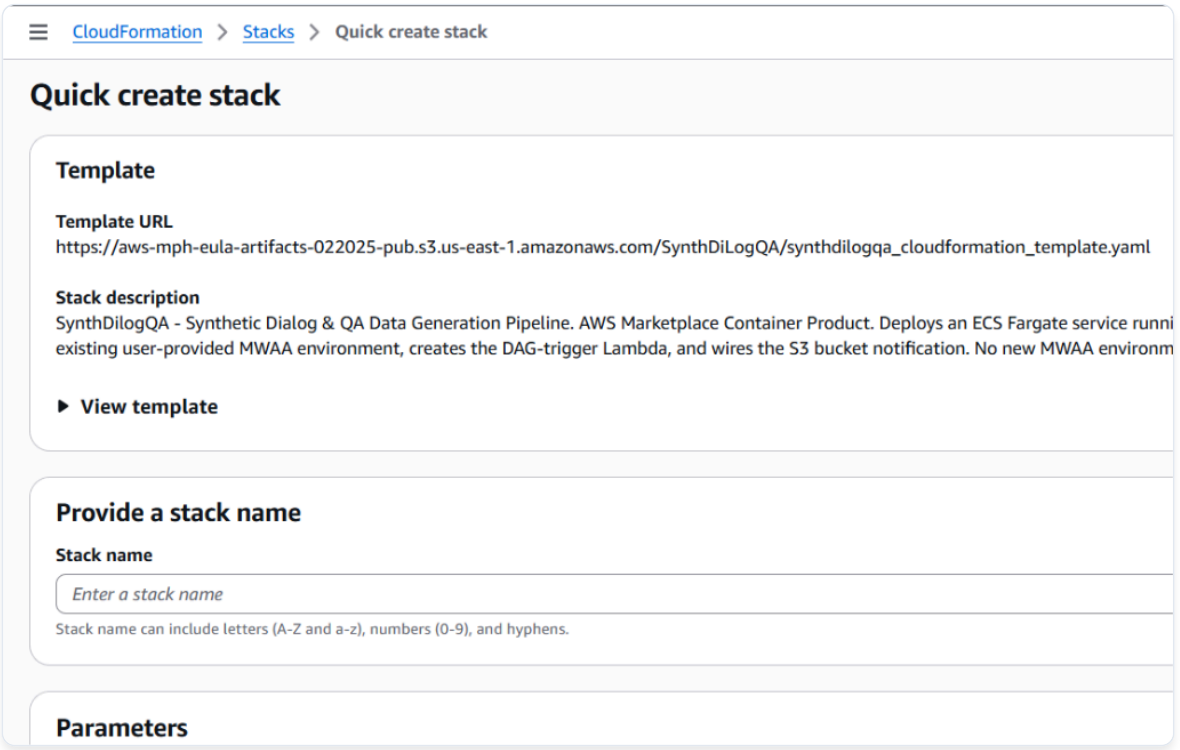
Copy the returned `ARN` — you will enter it as the `DBPasswordSecretArn` CloudFormation parameter.

Step 4 — Deploy the CloudFormation Stack

Option A: AWS Console

1. Open the **CloudFormation** console and click *Create stack → With new resources*.

2 Upload the CloudFormation template or paste the Marketplace-provided template URL.



3 Enter a unique **Stack name** (e.g. `synthdilogqa-prod`).

4 Fill in all parameters (see the Parameters Reference section below).

5 On the *Capabilities* page, check *I acknowledge that AWS CloudFormation might create IAM resources with custom names*.

6 Click *Submit* and wait for status `CREATE_COMPLETE` (~5–8 minutes).

CloudFormation Parameters Reference

Container Image

Parameter	Required	Description
`ContainerImageUri`	Yes	ECR image URI provided by AWS Marketplace after subscribing. Format: `<account>.dkr.ecr.<region>.amazonaws.com/mphasis/synthdilogqa:<tag>`
`SetupImageUri`	Yes	ECR image URI for the one-shot setup container. Construct by using the same registry prefix and version tag, but replacing the repository name with `mphasis/synthdilog-setup` (Format: `<account>.dkr.ecr.<region>.amazonaws.com/mphasis/synthdilog-setup:<tag>`).

Networking

Parameter	Required	Description
`VpcId`	Yes	VPC ID. Must be the same VPC as your MWAA environment so Cloud Map DNS resolution works.
`ECSSubnetIds`	Yes	Comma-separated list of ≥ 2 **private** subnets in different AZs for ECS Fargate tasks. Must have NAT Gateway routes for outbound internet access (Bedrock, ECR, Secrets Manager).
`ECSSecurityGroupId`	Yes	Security group for ECS Fargate tasks with appropriate inbound and outbound rules. See Prerequisites.
`MWAASecurityGroupId`	Yes	Security group ID attached to your MWAA environment workers. Used to automatically add an inbound rule to the ECS security group.

Storage – S3

Parameter	Required	Description
`S3BucketName`	Yes	Name of your existing S3 bucket. Used for SOP document input and pipeline artifact/output storage.

Database – RDS Credentials

Parameter	Required	Description
`DBPasswordSecretArn`	Yes	ARN of a pre-existing Secrets Manager secret containing your RDS credentials. The secret **must** contain these exact JSON keys: `DB_TYPE`, `DB_USER`, `DB_PASSWORD`, `DB_HOST`, `DB_PORT`, `DB_NAME`. See Step 2 for how to create it.

Workflow – Amazon MWAA

Parameter	Default	Required	Description
`MWAAEnvName`	—	Yes	Name of your MWAA environment (not the ARN). Case-sensitive. Found in the Amazon MWAA console under *Environments*.

Step 5 — Verify DAG Availability

The CloudFormation stack automatically uploads the orchestration DAG and requirements configuration to your S3 bucket during deployment.

1. Wait ~20-30 minutes for the MWAA environment to finish creating and for the Airflow scheduler to pick up the new files.
2. Open the **Airflow UI** (MWAA console → your environment → *Open Airflow UI*).
3. Confirm that `sop_upload_synth` appears in the DAGs list.
4. **Enable the DAG:** Click the toggle switch next to the DAG name to turn it **On**.

Step 6 — (Optional) Set DAG Display Name in MWAA

You can set a cosmetic display name for the DAG shown in the Airflow UI by adding an MWAA environment variable.

How to Set It (Optional)

1. Open the **Amazon MWAA** console and select your environment.

2. Click **Edit**.

3. Under *Airflow configuration options*, add:

```
Key:   env.DAG_DISPLAY_NAME
Value: SOP Conversational Agent Testing
```

4. Click **Save** and wait for the environment update.

Step 7 — Verify the Pipeline

1. **Verify the stack deployed successfully** — open the CloudFormation console, select your stack, and confirm the status shows `CREATE_COMPLETE`.

2. **Trigger the pipeline** by uploading a SOP document to your S3 bucket under the `input/sop_documents/` prefix.

3. **Monitor the Airflow DAG run** in the Airflow UI. Open the MWAA console → your environment → *Open Airflow UI* and check the DAG runs list. Tasks will progress through *queued* → *running* → *success*. The full pipeline takes approximately 15–40 minutes depending on document size.

4. **Check S3 for output artifacts** — the finished QA dataset and intermediate files are written back to your S3 bucket under the `artifacts/` prefix.

Security & Encryption

SynthDilogQA is designed with security as a top priority, adhering strictly to the AWS Well-Architected framework and least-privilege access rules.

Data Storage & Encryption

Encryption Configurations

- **AWS Secrets Manager:** Database connection credentials are stored encrypted in Secrets Manager using default or custom AWS KMS keys. The ECS container reads the credentials dynamically in memory at startup; plaintext credentials are never written to environment variables, files, or task definitions.
- **S3 Bucket Storage:** SOP documents and generated QA pipeline artifacts are encrypted at rest using Amazon S3 Managed Keys (SSE-S3) or customer-managed AWS KMS keys (SSE-KMS) according to your bucket policies.
- **RDS Database storage:** Database storage is encrypted at rest natively by AWS RDS using KMS keys.
- **Credentials Rotation:** Credentials in Secrets Manager can be rotated manually or dynamically. The ECS service automatically resolves updates on service redeployment or periodic task refresh.

Security Best Practices Applied

- **Secrets Manager** — RDS credentials are stored encrypted and fetched at runtime by the container. They never appear in the task definition, CloudFormation exports, or CloudWatch logs.
- **Least-privilege IAM** — Each role (ECS task, ECS execution, Lambda) has only the minimum permissions it needs.
- **No public endpoint** — The Flask API is accessed only via Cloud Map private DNS within your VPC. AssignPublicIp is disabled on ECS tasks.
- **No IAM keys** — The ECS container authenticates to AWS (S3, Bedrock, Secrets Manager) via the ECS Task Role — no long-term credentials are stored anywhere.
- **MWAA token-based auth** — Lambda uses short-lived web login tokens to call the Airflow REST API; no Airflow username or password is stored.
- **S3 source account guard** — The Lambda resource policy restricts invocations to your specific account ID, preventing confused-deputy attacks.

Environment Variable Security Guide

The table below shows every environment variable used across the pipeline components, its sensitivity level, and how it is stored and accessed.

Variable	Sensitivity	Storage Method	Notes
`DB_PASSWORD_SECRET_ARN`	LOW	CloudFormation → ECS environment variable	The ARN itself is not sensitive. The container calls Secrets Manager at runtime to retrieve the full credentials JSON (DB_TYPE, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT, DB_NAME). Credentials are never stored in the task definition.
`S3_BUCKET_NAME`	LOW	CloudFormation → ECS environment variable	Bucket name. Access is controlled by the ECS Task Role IAM policy.
`CLAUDE_MODEL_ID`	NONE	CloudFormation → ECS environment variable	Bedrock model ID for LLM inference. Defaults to cross-region Claude Sonnet 4.5.
`AWS_REGION`	NONE	CloudFormation → ECS environment variable	Required to configure boto3 clients for S3, Bedrock, and Secrets Manager.
`API_BASE_URL`	NONE	CloudFormation → ECS environment variable	Set to http://flask-app.mwaa-internal.local:5000 (the Cloud Map DNS name). Also passed to the DAG via Lambda conf.
`FLASK_APP`, `FLASK_ENV`	NONE	CloudFormation → ECS environment variable	Static configuration values.
`DAG_ID`	NONE	CloudFormation → Lambda environment	Airflow DAG identifier. Defaults to sop_upload_synth.
`MWAA_ENV_NAME`	LOW	CloudFormation → Lambda environment	Used only to call airflow:CreateCliToken; not a credential.

IAM Permissions Summary

CloudFormation creates three IAM roles. No AWS-managed policies with broad access are attached beyond the minimum required standard policies.

Role	Purpose	Key Permissions
`<stack>-ecs-execution-role`	ECS agent: pull image, write CloudWatch logs, read secrets	AmazonECSTaskExecutionRolePolicy + secretsmanager:GetSecretValue on your RDS secret ARN
`<stack>-ecs-task-role`	Runtime identity of the Flask API container	S3 (GetObject, PutObject, ListBucket, DeleteObject) on your bucket; Bedrock InvokeModel; Secrets Manager GetSecretValue on your RDS secret
`<stack>-lambda-dag-trigger-role`	Lambda DAG trigger function	AWSLambdaBasicExecutionRole + airflow:CreateCliToken on your MWAA environment ARN

Service Quotas

Ensure your AWS account regional service quotas are configured appropriately before deploying the stack:

AWS Service	Quota Name	Required Limit	Action if exceeded
Amazon Bedrock	Inference Profile requests per minute (RPM) for Claude Sonnet	Min 50 RPM (varies by document sizes & workflow parallelism)	Request limit increase via AWS Service Quotas (Anthropic Claude models)
Amazon Bedrock	Inference Profile tokens per minute (TPM) for Claude Sonnet	Min 80,000 TPM	Request limit increase via AWS Service Quotas
Amazon ECS	Fargate tasks per region	Min 2 concurrent tasks	Standard quotas are 10+, request increase if region is highly utilized
AWS Secrets Manager	GetSecretValue API requests per second	Default (10,000/sec) is sufficient	No action needed; credentials are loaded at start and cached

Resource Pricing Estimates

Deploying this product launches additional paid AWS infrastructure within your AWS account. Estimated monthly costs under normal operations are detailed below:

AWS Resource	Pricing Type	Estimated Base Cost	Usage Dimension Notes
Amazon MWAA	Environment size (mw1.small)	~$350.00 / month	Fixed base fee for running Airflow (workers and scheduler)
ECS Fargate Task	vCPU & Memory runtime (1 task, 0.5 vCPU, 1 GB RAM)	~$15.00 / month	Charged per second of running Flask API tasks (desired count of 1)
AWS NAT Gateway	Gateway hourly charge + data processed	~$32.00 / month	Required for private subnet outbound traffic (billing + ECR pulls). Extra data fee: $0.045 per GB
AWS Secrets Manager	Active secret count + API calls	~$0.40 / month	$0.40 per secret per month, plus $0.05 per 10,000 API calls (negligible)
AWS Cloud Map	Private namespace + DNS queries	~$0.50 / month	$0.50 per namespace per month, plus query fees (negligible)
Amazon Bedrock	Claude Sonnet API usage	Pay-as-you-go	Varies by document size. Approx. $0.003 / 1K input tokens and $0.015 / 1K output tokens

Usage Billing & Metering

SynthDilogQA is an AWS Marketplace **Usage-Based** product. Billing is calculated based on the number of successful pipeline runs triggered.

> ℹ️ Metering only occurs if the `MarketplaceProductCode` parameter is provided during CloudFormation deployment. If left blank, the pipeline will function normally but no marketplace usage will be recorded (useful for internal testing or BYOL scenarios).

Troubleshooting

ECS task keeps restarting / fails to start

- Check CloudWatch logs for the ECS task via the CloudWatch console.
- Confirm the ECS task role has `secretsmanager:GetSecretValue` on your `DBPasswordSecretArn`. An `AccessDeniedException` in logs means the role lacks this permission.
- Confirm the ECS subnets have NAT Gateway routes (ECR image pulls require outbound HTTPS 443).
- Verify `Amazon Bedrock model access` is enabled for Claude Sonnet in your account/region.

Airflow DAG tasks fail with "connection refused" or timeout

- Confirm MWAA and ECS are in the same VPC — Cloud Map DNS (`mwaa-internal.local`) only resolves within the same VPC.
- Verify the ECS security group allows inbound traffic from the MWAA worker security group.
- Confirm the ECS service has at least 1 running task (`runningCount: 1`).
- Check Cloud Map in the AWS console: **Service Discovery** → **Namespaces** → **mwaa-internal.local** → **flask-app** — the service instance should show a healthy IP.

Lambda is not triggered on S3 upload

- Confirm the setup container completed successfully during stack deployment (check the `synthdilogqa-setup` log group).
- Confirm the object key starts with `input/sop_documents/` (prefix must include the trailing slash).
- Check that the S3 bucket notification exists: **S3 Console** → **your bucket** → **Properties** → **Event notifications**.
- Verify the Lambda resource policy allows `s3.amazonaws.com` from your account ID.

Lambda error: "Airflow returned HTTP 403"

- Confirm the Lambda execution role has `airflow:CreateCliToken` on your MWAA environment ARN.
- Verify `MWAA_ENV_NAME` (Lambda environment variable) matches exactly the name of your MWAA environment — it is case-sensitive.
- Ensure the MWAA environment is in `AVAILABLE` state.

DAG import error: "No module named 'requests'"

- The `requests` library must be available to MWAA workers. Add `requests` to your MWAA environment's requirements file in the MWAA S3 bucket and update the MWAA environment to pick it up.

Secret read fails at runtime ("Invalid secret JSON")

- The Secrets Manager secret must contain all six keys exactly as named: `DB_TYPE`, `DB_USER`, `DB_PASSWORD`, `DB_HOST`, `DB_PORT`, `DB_NAME`. Missing or differently named keys will cause a runtime error in the container.
- Verify with: `aws secretsmanager get-secret-value --secret-id <YOUR-ARN> --query SecretString --output text`

Bedrock inference error ("ValidationException" or throttling)

- Ensure **Amazon Bedrock model access** is enabled for Anthropic Claude Sonnet 4.5 in your AWS account and region. Open the Bedrock console → *Model access* and enable the model.
- If using cross-region inference (the default), confirm the region you deployed into supports cross-region inference profiles.
- For throttling errors, the pipeline will retry automatically. For persistent quota issues, request a Bedrock service quota increase.

Support

For issues with this AWS Marketplace product, use the **Support** link on the Marketplace product page. When contacting support, provide:

- Your CloudFormation stack name and AWS region.
- The relevant CloudWatch log streams (ECS task logs and Lambda logs).
- The error message and task ID from the Airflow UI.

> ✅ Before contacting support, check the Troubleshooting section above — most common issues are covered there.