# Serverless News Aggregator - AWS Architecture

This document outlines the serverless architecture for the News Aggregator application. The design prioritizes cost-effectiveness, scalability, and minimal operational overhead by leveraging managed AWS services.

The architecture is split into two primary workflows:

1. **Data Ingestion (Scheduled Task):** An automated process that periodically fetches and stores news articles.
2. **User-Facing Application (API & Frontend):** The components that serve the news to the end-user's browser.

## Core AWS Services

### 1. Amazon S3 (Simple Storage Service)

* **Role:** Frontend Hosting
* **Function:** The index.html file, which contains all the HTML, CSS, and JavaScript for the user interface, is stored in an S3 bucket. This bucket is configured for **static website hosting**, making the frontend accessible to users via a public URL. This is an extremely cheap and durable way to host a static site.

### 2. Amazon DynamoDB

* **Role:** NoSQL Database
* **Function:** DynamoDB is a fully managed NoSQL database that serves as the central repository for all news articles. Each article is stored as an item in a table, with the article's unique URL (link) acting as the primary key. It provides fast, single-digit millisecond performance at any scale.

### 3. AWS Lambda

The project uses two separate Lambda functions, which are small, on-demand compute services that run code without needing to manage servers.

* **FetchNewsFunction (The Collector):**
  + **Purpose:** This function contains the Python code (fetch\_news\_lambda.py) responsible for fetching news from various RSS feeds.
  + **Trigger:** It is triggered automatically by Amazon EventBridge on a schedule (e.g., once every hour).
  + **Action:** When triggered, it fetches articles, parses them, and writes them into the DynamoDB NewsArticles table.
* **ApiHandlerFunction (The Server):**
  + **Purpose:** This function (api\_handler\_lambda.py) acts as the backend for the website.
  + **Trigger:** It is invoked by Amazon API Gateway whenever a user's browser makes a request.
  + **Action:** It handles two main tasks:
    1. Retrieving the list of news articles from DynamoDB to display on the website.
    2. Incrementing the viewCount for an article in DynamoDB when a user clicks on it.

### 4. Amazon API Gateway

* **Role:** Managed API Endpoint
* **Function:** API Gateway provides a public HTTP endpoint (a URL) for the ApiHandlerFunction. It receives requests from the user's browser (e.g., "get all news") and securely routes them to the correct Lambda function. It also handles crucial tasks like Cross-Origin Resource Sharing (CORS) to allow the S3-hosted website to access the API.

### 5. Amazon EventBridge (CloudWatch Events)

* **Role:** Serverless Scheduler
* **Function:** EventBridge is used to create a "rule" that runs on a fixed schedule. This rule is configured to trigger the FetchNewsFunction Lambda every hour, automating the entire data collection process.

## Workflow Summary

1. **Automated Fetching:** Every hour, **EventBridge** triggers the **FetchNewsFunction**. This Lambda fetches news from external RSS feeds and stores the articles in the **DynamoDB** table.
2. **User Visits Site:** A user navigates to the website URL hosted on **S3**. Their browser loads the index.html file.
3. **Frontend Requests Data:** The JavaScript in index.html sends a request to the **API Gateway** endpoint.
4. **API Responds:** **API Gateway** forwards the request to the **ApiHandlerFunction**. This Lambda queries the **DynamoDB** table, retrieves all the articles, and sends them back through the API Gateway to the user's browser.
5. **Display and Interaction:** The frontend renders the articles. When a user clicks an article link, another request is sent via **API Gateway** to the ApiHandlerFunction to track the view, updating the viewCount in DynamoDB.