# **ZILLOW REAL ESTATE DATA PIPELINE WITH AIRFLOW, AWS LAMBDA, REDSHIFT, AND QUICKSIGHT**

In this data engineering project, we demonstrate how to build and automate a Python-based ETL pipeline that extracts real estate property data from the **Zillow Rapid API**, processes it through AWS services, and visualizes it with **Amazon QuickSight**.

|  |  |
| --- | --- |
| Tool/Service | Purpose |
| Python | API call & scripting |
| Apache Airflow (on EC2) | DAG orchestration |
| BashOperator | Move files from EC2 to S3 |
| AWS Lambda | File copy & transformation automation |
| Amazon S3 | Staging, transforming, storing files |
| S3KeySensorOperator | Detect presence of file before Redshift load |
| Amazon Redshift | Data warehouse |
| Amazon QuickSight | BI tool for visualization |

1. **Python ETL Pipeline Design**

### ****Data Extraction & Staging:****

* **Zillow Rapid API** serves as the primary data source.
* Using **Python**, the API is queried to extract structured real estate property information such as number of bedrooms, bathrooms, price, rent estimates, property type, and location.
* The raw data is then loaded into an **Amazon S3 bucket** referred to as the **Landing Zone**.
  1. **S3 Zones & AWS Lambda Triggers:**
* Once data lands in the **Landing Zone**, it triggers a **Lambda function** that copies the data into an **Intermediate Zone bucket**.
* The purpose of this separation is to preserve the integrity of the raw data in the Landing Zone and prevent unintentional modifications. All further processing is done in the Intermediate Zone.
* A second **Lambda function** is triggered from the Intermediate Zone, which performs data **transformation** and **cleansing**, and writes the **transformed CSV** output to another S3 bucket – the **Transformed Data Zone**.

### ****Data Warehouse Integration:****

* The transformed CSV file is loaded into a provisioned **Amazon Redshift cluster**. Redshift serves as the cloud-based data warehouse for persistent storage and analytical querying of the real estate dataset.

### ****Data Visualization with QuickSight:****

* Once data resides in Redshift, **Amazon QuickSight** is connected to the Redshift cluster.
* This enables creation of dynamic visual dashboards to explore real estate market trends such as average price per ZIP code, distribution of home types, and bedroom-price relationships.

## **Apache Airflow Orchestration**

**Apache Airflow**, running on an **Amazon EC2 instance**, orchestrates the entire ETL process through a well-defined **Directed Acyclic Graph (DAG)**.

### ****DAG Components:****

* **PythonOperator**:  
  This task connects to the Zillow Rapid API and extracts the real estate data using Python. The data is first temporarily stored on the EC2 instance.
* **BashOperator**:  
  This task moves the extracted data from the EC2 local environment to the **S3 Landing Zone** bucket using shell commands.
* **AWS Lambda Triggers** (outside Airflow):  
  Once data is placed in the Landing Zone, a Lambda function is triggered to:
  1. Copy data to the Intermediate Zone.
  2. Trigger another Lambda function to transform the data and save it to the Transformed Zone.
* **S3KeySensor**:  
  Before loading data into Redshift, this sensor continuously monitors the **Transformed Data Zone S3 bucket** for the presence of the transformed CSV file. Only once the file is detected does the pipeline proceed.
* **S3ToRedshiftOperator**:  
  This task loads the transformed CSV data into a specified Redshift table using Redshift’s native COPY command, making the data ready for querying and visualization.

## **AWS Setup**

This section outlines the foundational setup of AWS services to support the Zillow Real Estate Data Pipeline project. All cloud infrastructure used in this project is provisioned within the **AWS ecosystem**, ensuring scalability, security, and seamless integration between services.

### ****IAM User Groups and Permissions****

In a production environment, it is a best practice **not to use the root user** for deploying and managing AWS resources. Instead, we:

1. **Create IAM User Groups** to manage permissions more effectively.
   * For instance, if certain engineers only need read access to an S3 bucket while others require full access, we can:
     + Create a **Read-Only Group**.
     + Create a **Read-Write Group**.
   * Permissions are assigned to the group, and users are added to the appropriate group rather than managing individual permissions manually.
2. **Create a Project-Specific IAM User**:
   * Assign this user to the **Administrative Group** created above.
   * Enable **AWS Management Console access** for the user.
   * Generate and download the **Access Key ID** and **Secret Access Key**. These credentials are required for programmatic access via AWS SDKs or the AWS CLI.
   * Log in using this newly created IAM user for all subsequent operations in the project.

### ****Launch Amazon EC2 Instance for Airflow****

To orchestrate the ETL pipeline using Apache Airflow, we set up an **EC2 instance**.

1. **Choose Amazon Machine Image (AMI)**:
   * Use **Ubuntu Server** for compatibility with Airflow and common Python tools.
2. **Select Instance Type**:
   * Avoid using t2.micro (Free Tier) as it lacks the necessary memory for Airflow.
   * Recommended:
     + t2.small (2 GiB) – may work with minor lag.
     + t2.medium (4 GiB) – ideal for smooth operation.
3. **Create or Select Key Pair**:
   * Key pairs are used for **SSH access** to the instance.
   * Note: These keys are not used for AWS API access (IAM credentials handle that).