**AWS services used**

In order to migrate the existing application to cloud I used the below AWS services

**AWS Glue**

AWS Glue is a fully managed Extract, Transform, and Load (ETL) service that provides capabilities for data preparation and integration.

Based on the provided requirements automation, scalability, and integration

Automation:

* **Serverless Architecture**: AWS Glue eliminates the need to manage and provision servers, ensuring minimal manual intervention.
* **Job Automation**: AWS Glue offers **Workflows**, **Triggers**, and **Schedulers** to automate ETL jobs and manage dependencies across multiple tasks.

Scalability:

* **Elastic Scaling**: AWS Glue can scale horizontally to handle data processing workloads of varying sizes without requiring manual scaling configurations.
* **Distributed Processing**: Glue jobs run on Apache Spark clusters, which are inherently distributed, enabling high performance for large-scale datasets.

Integration:

* **AWS Ecosystem**: Glue integrates natively with other AWS services such as:
* **S3** for storage
* **Lambda** for serverless orchestration

**For the application I use AWS Glue to run the trainer model which will access the S3 input folder then train the file and put the trained files on S3 model/ and output the file to S3 output/ folder, we can use AWS Glue on cloud without the need to manage and provision servers**

**Lambda Function**

AWS Lambda is a serverless compute service that enables the execution of code in response to events.

Automation:

* **Event-Driven Execution**: AWS Lambda automatically triggers code execution in response to events from over 200 AWS and external services (e.g., S3 uploads, API Gateway calls, or custom events).
* **Serverless Model**: No need to manage servers, operating systems, or runtime environments—AWS Lambda automatically provisions infrastructure and scales resources.

Scalability:

* **Automatic Scaling**: AWS Lambda automatically scales the number of concurrent function executions in response to traffic spikes or drops, making it inherently scalable.
* **Concurrency Limits**: Lambda provides controls to define concurrency thresholds, ensuring critical workloads remain prioritized.

Integration:

* **AWS Ecosystem**: Glue integrates natively with other AWS services such as:
* **S3** for storage
* **SNS** and **SQS** for event-driven messaging.

**For the application I mapped the refresh\_function.py with AWS Lambda to trigger a glue job when a new csv file is uploaded to the S3 input/ folder**

**ECS**

Amazon ECS (Elastic Container Service) is a fully managed container orchestration service that is well-suited for running containerized applications, including a FastAPI app.

Automation:

* **Managed Container Orchestration**: ECS handles container lifecycle management, including deployment, scaling, and availability.
* **Task Scheduling**: ECS allows you to define tasks and services, automating container deployment based on user-defined configurations.

Scalability:

* **Elastic Scaling**: ECS automatically adjusts the number of container instances (tasks) to handle variable workloads.
* **Support for Multiple Deployment Models**: ECS runs on:
* **EC2 Instances**: For fine-grained control over infrastructure.
* **AWS Fargate**: A serverless option where AWS manages the compute resources, enabling seamless scaling without managing infrastructure.
* **Load Balancing**: ECS integrates with Application Load Balancers (ALB) for distributing traffic across multiple FastAPI containers, ensuring scalability.

Integration:

* **AWS Ecosystem**: ECS integrates seamlessly with:
* **Elastic Load Balancing (ELB)** for traffic distribution.
* **CloudWatch** for monitoring and logging.
* **IAM** for fine-grained role-based access control.
* **Networking and Security**: ECS works within your **VPC** to secure your app, with support for public and private subnets

**For the application, from Cloudformation, I created a VPC for my environment then created ECS cluster to manage the container lifecycle, from the ECS cluster I created IAM role for the app to access S3 bucket and run ecs tasks, then created a service with a load balancer which will accept input traffic and balance it across the containers, after that I tagged and pushed my fastapi app to the elastic container registry so that the task definition can run based on my uploaded image**