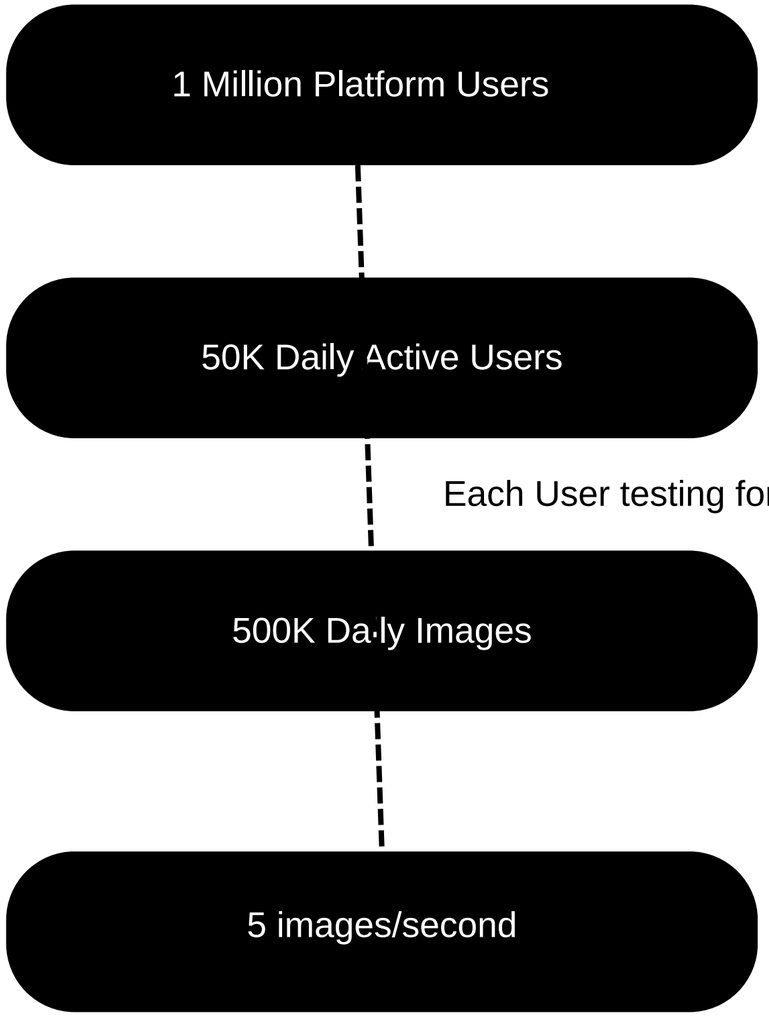
Problem Statement:
Face Detection and Matching Project

Requirements

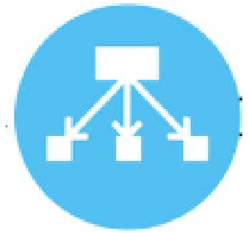
Requirements	
Functional	**Non Functional**
User Uploads the image	Reliability
User Gets the Similarity Score	Low Latency
	Scalable
	High Availability

Setting Assumptions



1 Image Upload & Face Validation

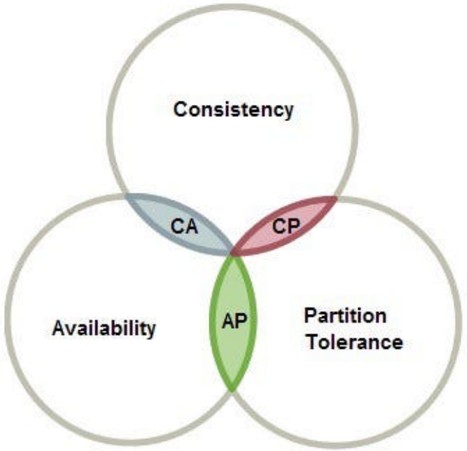
Uploaded Image → API Gateway/Load Balancer → FastAPI APP



Accepts image in memory
Run RetinaFace for Face
if Valid(Store in s3)
if not (Reject with msg)

2 Data Collection Layer

Validated Images



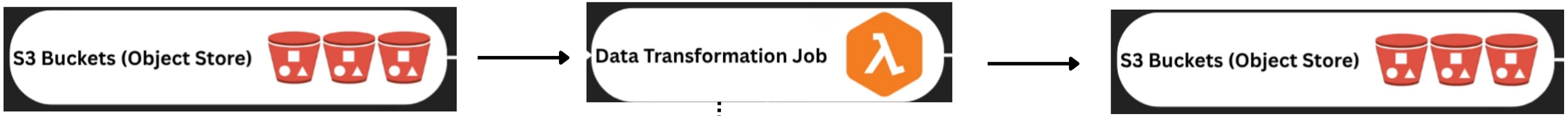
Storage

- AWS S3 Bucket (Raw Image Storage)
- Unprocessed, high-resolution images are stored in S3.
- MongoDB (Metadata)
- Stores metadata for each image such as:

MongoDB MetaData Structure

{
"image_id": "UUID",
"user_id": "xyz123",
"timestamp": "2025-07-20T12:34:56Z",
"s3_raw_path": "s3://bucket-name/raw/xyz123.jpg",
"status": "raw_uploaded"
“s3_processed_path”=”None”
}

3. Data Engineering & Preprocessing



Updates the S3 Processed Path

{
...
"s3_processed_path": "s3://bucket-name/processed/xyz123.jpg",
"status": "preprocessed"
}

⚙️ AWS Lambda Functions (Trigger-based)
Triggered upon new image upload in S3.
🧼 Preprocessing Tasks
Resize Image (to model-required size)
Pixel Value Scaling (e.g., 0–1 range)
Z-Score Normalization of pixel intensities
Convert to RGB (if not already)
Aspect Ratio Handling (padding or cropping)
Denoising (e.g., bilateral filtering)
crop the face region
📤 Output
Store the processed image back in S3 (/processed/)
Update MongoDB:

Why to Use AWS S3 Buckets:
Horizontal Scaling
Create Multiple Instances of AWS Lambda when demand increases

4 Model and Similarity Score

S3 Buckets (Object Store)

ArcFace

Embedding Model 512 Dimensional Vector

Refernce Images embedding store in weaviate

Weaviate VectorDB

(b) ArcFace

AWS SageMaker

Hosted on AWS SageMaker / EKS for scalable inference.
Use horizontal scaling to manage billions of images and high traffic.

Why Weaviate:

1) Can be Partitioned and Sharded for Scalability

Cosine Distance is Calculated for verification image and refernce image

Similarity Score

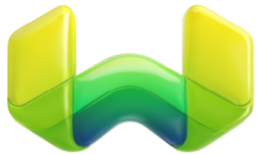
$$\text{score} = \max\left(0,\ 100 \times \left(1 - \frac{\text{distance}}{\text{max\_distance}}\right)\right)$$

5 Caching

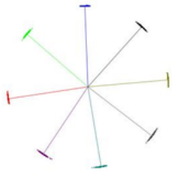
ArcFace

Embedding Model 512 Dimensional Vector

Refernce Images embedding store in weaviate



Weaviate VectorDB



(b) ArcFace

Cosine Distance is Calculated for verification image and refernce image

Similarity Score

$$\text{score} = \max\left(0,\ 100 \times \left(1 - \frac{\text{distance}}{\text{max\_distance}}\right)\right)$$

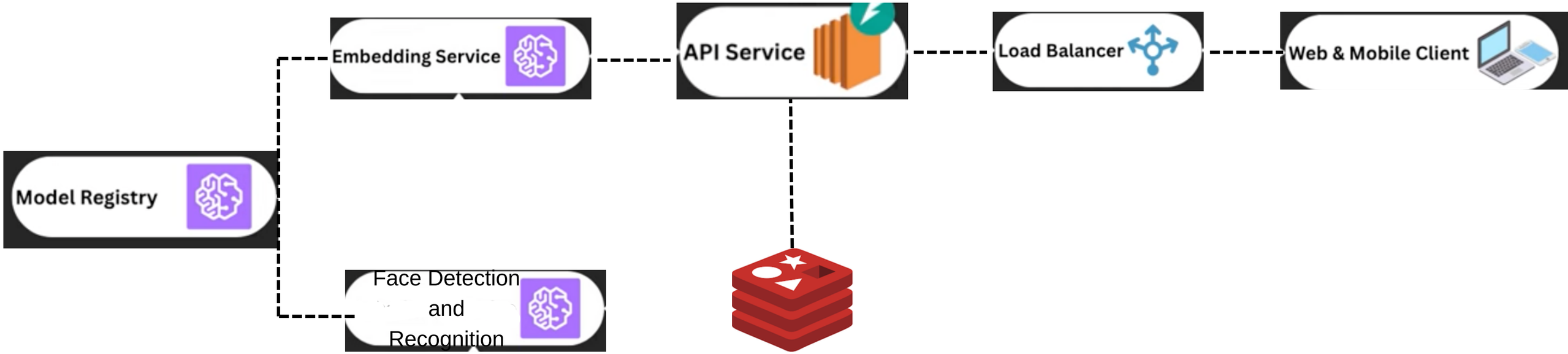
- Cache recent embeddings & similarity scores
- Reduce redundant computation for frequent comparisons

Monitoring & Logging

CloudWatch / Prometheus + Grafana

- Model latency
- API uptime
- Cache hit/miss ratio
- DB write/read health

Serving Layer



Tools Used

Layer	Components
Data Ingestion	S3, Face Validation using **RetinaFace**
Preprocessing	AWS Lambda, OpenCV, NumPy
Metadata DB	MongoDB
Model Inference	DeepFace + ArcFace
Vector Storage	Weaviate / FAISS
Caching	Redis
API Backend	FastAPI + Gunicorn on EC2
Model Registry	AWS Model Registry
Autoscaling	EC2 + Load Balancer
Monitoring & Logs	**Prometheus**, **Grafana**, **AWS CloudWatch**