Smart Media Uploader (SMMU)

Cloud-Native Distributed Media Processing Platform

A production-grade, event-driven media ingestion and processing backend built on **AWS** using **FastAPI, S3, DynamoDB, SQS, Step Functions, Lambda, and ECS Fargate**.

SMMU provides a **secure, scalable, fault-tolerant pipeline** for uploading, validating, classifying, and processing large media files (images, video, audio) without blocking API servers.

📌 What this system solves

Uploading and processing large media files is one of the hardest backend problems because it involves:

- Huge files
- Slow operations
- CPU-heavy workloads
- Retries and failures
- User isolation
- Cost control
- Long-running jobs

SMMU solves this by separating:

Concern	How it is handled
Upload	Direct-to-S3 via presigned URLs
Job control	DynamoDB
Reliability	SQS
Orchestration	Step Functions
Compute	ECS Fargate
API	FastAPI
Security	JWT + IAM
Scaling	Event-driven workers

This architecture is identical in principle to what AWS MediaConvert, Netflix, and SaaS video platforms use.

🧠 High-Level Architecture

```
Client
  ↓
FastAPI (ECS + ALB)
  ↓
Presigned URL → S3 (raw uploads)
  ↓
DynamoDB (Job created)
  ↓
SQS
  ↓
Lambda Dispatcher
  ↓
Step Functions
  ↓
ECS Fargate Workers
  ↓
S3 (processed output)
  ↓
DynamoDB (status, progress, metadata)
```

The API never handles large files and never blocks on processing.

Core Components

Layer	Responsibility
FastAPI	Auth, upload orchestration, job APIs
S3	Raw and processed media
DynamoDB	Job state, ownership, progress
SQS	Durable job queue
Lambda	Workflow entry, lightweight logic
Step Functions	Media pipeline brain
ECS Fargate	Heavy compute workers
Terraform	Infrastructure as code
GitHub Actions	CI/CD

Security Model

Layer	Security

Layer	Security
API	JWT bearer tokens
Jobs	Owned by `userId`
S3	IAM-scoped buckets
Workers	IAM roles
Step Functions	IAM role
Terraform	State locked in DynamoDB

Users **never** access S3 or AWS directly.

End-to-End Data Flow

```
1. Client → POST /media/upload/init
2. API → Creates upload session
3. API → Returns presigned S3 URL
4. Client → Uploads directly to S3
5. Client → POST /media/upload/complete
6. API → Validates file
7. API → Creates job in DynamoDB
8. API → Sends job to SQS
9. SQS → Dispatcher Lambda
10. Lambda → Step Functions
11. Step Functions → ECS worker
12. Worker → Processes file
13. Worker → Writes output to S3
14. Worker → Updates DynamoDB
15. API → GET /jobs/{jobId}
```

Infrastructure

Provisioned using Terraform:

S3

- `smmu-dev-raw-media`
- `smmu-dev-processed-media`

DynamoDB

- `smmu-dev-jobs` (PK: userId, SK: jobId, streams enabled)

- `smmu-dev-upload-sessions`
- `smmu-dev-media`

SQS

- `smmu-dev-jobs-queue`
- `smmu-dev-jobs-dlq`

ECS

- API service
- Transcode worker
- Transcribe worker

Step Functions

- `smmu-dev-pipeline`

🧠 Processing Pipeline

```
ValidateInput
   ↓
DetectType
   ↓
Branch
 ┌─────────────┬────────────┐
 │ Image       │ Audio      │
 │ ECS Worker  │ ECS Worker │
 └─────┬───────┴──────┬─────┘
       ↓              ↓
     UpdateStatus → Done
```

Lambdas decide, ECS executes.

🧪 Failure Handling

Failure	Where handled
Upload expired	Upload session TTL
File too large	API validation
Wrong MIME	API validation
Worker crash	ECS exit → Step Functions

Failure	Where handled
StepFn failure	Catch → UpdateStatus
Lambda failure	Retries
SQS failure	DLQ

No job is ever lost.

💰 Cost-efficient Design

Component	Why
Presigned S3	API never transfers large files
SQS	Cheap durable buffer
Step Functions	Only runs when needed
ECS on demand	No idle workers
Lambda	Only runs on events

📊 Observability

Layer	Logs
API	`/ecs/smmu-dev-api`
Transcode	`/ecs/smmu-dev-transcode`
Transcribe	`/ecs/smmu-dev-transcribe`
Lambda	`/aws/lambda/*`
Step Functions	Execution history

Everything is traceable by `jobId`.

🧠 What this system demonstrates

- Event-driven architecture
- Distributed systems
- IAM and cloud security
- Scalable compute
- Async job processing

- Media pipelines
- Infrastructure as Code
- Real AWS production patterns

📍 Current Status

The platform currently supports:

- Secure uploads
- Job creation
- Distributed processing
- ECS workers
- DynamoDB status tracking
- Step Functions orchestration
- Multi-user isolation
