Smart Media Uploader – Cloud-Native Backend

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

This system allows users to upload images, audio, or video files and processes them asynchronously (validation, classification, thumbnails, transcription, transcoding, metadata) with full observability, retries, and fault-tolerance.

🧠 High-Level Architecture

```
Client
  ↓
FastAPI (ECS Fargate)
  ↓
S3 (Presigned Upload)
  ↓
DynamoDB (Job record)
  ↓
SQS (Jobs Queue)
  ↓
Lambda (Dispatcher)
  ↓
Step Functions (Workflow Brain)
  ↓
Lambda / ECS Workers
  ↓
S3 (Processed Outputs)
  ↓
DynamoDB (Job Status + Metadata)
```

📦 Tech Stack

Layer	Technology
API	FastAPI on ECS Fargate
Storage	Amazon S3
Database	DynamoDB
Messaging	Amazon SQS
Orchestration	AWS Step Functions

Layer	Technology
Lightweight compute	AWS Lambda
Heavy compute	ECS Fargate
Infrastructure	Terraform
CI/CD	GitHub Actions
State & locking	S3 + DynamoDB backend

🚀 Development Phases

Phase 0 — Infrastructure Foundation

Goal: Create a production-grade AWS foundation using Terraform and CI/CD.

What we built

- Remote Terraform state:
  - S3 backend
  - DynamoDB state lock
- Modular Terraform structure:
  - networking
  - compute
  - messaging
  - data
  - orchestration
  - storage
- VPC, subnets, IGW
- S3 buckets:
  - `smmu-dev-raw-media`
  - `smmu-dev-processed-media`
- DynamoDB tables:
  - `smmu-dev-jobs`
  - `smmu-dev-media`
- SQS queues:
  - `smmu-dev-jobs-queue`
  - `smmu-dev-jobs-dlq`

- IAM roles with least-privilege
- GitHub Actions pipeline with remote state

Problems solved

- Terraform state was local → migrated to S3
- CI/CD could not access AWS → fixed credentials
- State lock conflicts → resolved via DynamoDB

What was learned

- How real teams manage Terraform state
- Why remote state + locking prevents infrastructure corruption
- How CI/CD deploys cloud infrastructure safely

Phase 1 — API Layer (FastAPI on ECS)

Goal: Expose a production-grade API.

What we built

- FastAPI service running in ECS Fargate
- Docker image pushed to ECR
- Application Load Balancer
- HTTPS-ready architecture
- ECS service auto-deployable via Terraform

What we validated

- ALB target group health checks
- ECS task networking
- IAM permissions to pull images from ECR
- CloudWatch logs

Problems solved

- ECS tasks stuck in PENDING → fixed VPC + NAT + ECR access
- ALB not routing → fixed target group & security groups

What was learned

- How containers run in AWS
- How ALB connects to ECS
- How production APIs are hosted without servers

Phase 2 — Asynchronous Workflow Engine

This is where the system became **distributed**.

What we built

SQS-driven pipeline

- API (later) sends jobs to SQS
- SQS triggers Lambda

Dispatcher Lambda

- Reads job from SQS
- Starts Step Functions execution

Step Functions (the brain)

```
ValidateInput → DetectType → Branch → UpdateStatus
```

Workflow Lambdas

- `validate_input`
- `detect_type`
- `update_status`

DynamoDB integration

- Job status updated by workflow
- Verified with real data

What we validated

We sent:

```
{"jobId":"test-123","type":"image","inputKey":"s3://fake"}
```

Result:

```
SQS → Lambda → Step Functions → Lambdas → DynamoDB
```

And DynamoDB showed:

```
status = RUNNING
```

This proves:

- Queue → Workflow → Database → Orchestration works

Problems we debugged

Problem	What it taught
Invalid JSON from SQS	How SQS wraps messages
Lambda missing permissions	Real IAM debugging
Step Functions not triggering	Event source mapping
Missing fields in state	How Step Functions passes JSON
CI couldn't find ZIP files	Artifact build vs IaC separation

We solved all of them like production engineers do.

🧩 Current Status

The system can now:

- Accept jobs
- Enqueue them
- Start workflows
- Route by type
- Track status in DynamoDB

The platform is now a **real event-driven backend**.

⏭️ Next: Phase 3

I will add **real media processing**:

- FFmpeg transcoding (ECS)
- Whisper transcription (ECS)
- Step Functions → ECS integration

This will turn the system into a **true media pipeline**.
