## **Definitive Guide: Deploying Python/MoviePy/FFmpeg to AWS Lambda Containers**

This guide focuses on the finalized, stable configuration that overcomes cross-platform (macOS to Linux) and Lambda runtime environment conflicts, resulting in a successful deployment of your video processing pipeline.

## **1. Local Setup and Code Preparation 🛠️**

This stage prepares the source code and necessary binaries on your local machine.

### **A. Project Structure**

1. **Create Project Folder:** Initialize your project directory (e.g., video-processor-lambda).
2. **Code Files:** Place the finalized Python script (lambda\_function.py) and dependencies file (requirements.txt) in this folder.

### **B. FFmpeg Binary Acquisition**

The specific version of FFmpeg must be statically compiled for Linux to run correctly in the Lambda environment.

1. **Download:** Obtain the statically compiled **x86\_64 (amd64) FFmpeg binary**. A reliable source is typically **John Van Sickle's** static FFmpeg builds.
2. **Acquire and Extract (Terminal Commands):** Use curl to download and tar to extract the binary, ensuring you get the amd64-static version.  
   Bash

  
# Download the static build (URL is example/subject to change, but path is similar)

curl -L -o ffmpeg-release.tar.xz https://johnvansickle.com/ffmpeg/releases/ffmpeg-release-amd64-static.tar.xz

# Extract the archive

tar -xf ffmpeg-release.tar.xz

# Find the extracted folder (name varies, e.g., ffmpeg-5.1.2-amd64-static)

FFMPEG\_DIR=$(find . -maxdepth 1 -type d -name "ffmpeg-\*-amd64-static" | head -n 1)

# Copy the binary to the root and rename it to the required file name

cp "${FFMPEG\_DIR}/ffmpeg" .

# Clean up downloaded files

rm -rf "${FFMPEG\_DIR}" ffmpeg-release.tar.xz



1. **Staging:** Ensure the final executable file is named exactly **ffmpeg** and resides in your project root.

## **2. Docker Build and Push to ECR 🐳**

This single-stage build targets the stable Python 3.10 runtime, which is crucial for consistency.

### **A. Dockerfile (Final Working Structure)**

The Dockerfile ensures compiler tools are installed onto the base image to build native dependencies.

Dockerfile

# --- DEFINITIVE FINAL DOCKERFILE ---

FROM public.ecr.aws/lambda/python:3.10-x86\_64

WORKDIR /var/task

# 1. Install necessary system dependencies (GCC, Python Dev)

RUN yum update -y && \

yum install -y \

gcc \

gcc-c++ \

python3-devel \

make \

unzip \

tar && \

yum clean all

# 2. Install ALL Python Dependencies into /var/task

COPY requirements.txt .

RUN pip install -r requirements.txt awslambdaric -t /var/task

# 3. Copy the Lambda handler code and the FFmpeg binary

COPY lambda\_function.py .

COPY ffmpeg .

RUN chmod +x ./ffmpeg

CMD [ "lambda\_function.lambda\_handler" ]

### **B. Build and ECR Tag Correction (Detailed Steps)**

1. **Set Environment Variables (Local Terminal):** Define the critical variables for building and pushing.  
   Bash

  
AWS\_ACCOUNT\_ID="922523344160"

AWS\_REGION="ap-southeast-2"

ECR\_URI="${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_REGION}.amazonaws.com/video-processor-repo"



1. **Authenticate Docker:** Re-authenticate to ECR (necessary as tokens expire).  
   Bash

  
aws ecr get-login-password --region ${AWS\_REGION} | docker login --username AWS --password-stdin ${ECR\_URI}



1. **Build Command:** Build the image, enforcing the correct architecture and using the defined URI.  
   Bash

  
docker build --platform linux/amd64 -t ${ECR\_URI}:python310-final .

1. **Initial Push Command:** Push the multi-architecture image to ECR.  
   Bash

  
docker push ${ECR\_URI}:python310-final



1. **ECR Manifest Fix (Single-Architecture Tagging):** This is mandatory to resolve the Lambda deployment error on macOS-built images.
   * **Get Manifest Digest:** Retrieve the full, complex manifest list to find the single AMD64 digest.  
     Bash

  
aws ecr batch-get-image --repository-name video-processor-repo --image-ids imageTag=python310-final --output json --query 'images[0].imageManifest' > manifest\_list.json

# NOTE: Manually open 'manifest\_list.json' and extract the specific digest value

# for the "amd64" platform (e.g., sha256:XXXXXXXX...).



* + **Create Simple Tag:** Use the extracted digest (<Image-Digest-Value>) to create a new, simplified tag (python310-final-ready).  
    Bash

  
AMD64\_MANIFEST=$(aws ecr batch-get-image \

--repository-name video-processor-repo \

--image-ids imageDigest=<Image-Digest-Value> \

--query 'images[0].imageManifest' \

--output text)

aws ecr put-image \

--repository-name video-processor-repo \

--image-tag python310-final-ready \

--image-manifest "$AMD64\_MANIFEST"



## **3. AWS Lambda Configuration and Testing ✅**

### **A. Lambda Creation**

1. **Create Function:** Select **Container image**.
2. **Name:** VideoProcessorFinalProcessor
3. **Image Tag:** Select the cleaned tag: **python310-final-ready**.
4. **Architecture:** x86\_64.

### **B. Critical Configuration Overrides**

1. **Resources (Configuration → General configuration):**
   * **Memory:** **2048 MB** (Minimum for processing).
   * **Timeout:** **3 mins** (Minimum).
2. **Environment Variables:**
   * **MOVIEPY\_TEMP\_DIR:** /tmp (Solves the Read-only file system error).

### **C. Testing**

1. **Test Execution:** Run a console test event or upload a file matching your S3 trigger.
2. **Verification:** Confirm the output video lands at the root of the designated S3 prefix: output\_videos/final\_video.mp4.