Run Book: Deploying the H2V Pipeline on AWS

This run book is a walk-through on the steps I'd take to containerize the application, push it to AWS ECR, and then deploy it using one of two AWS orchestrators:

- **Option A: AWS ECS with Fargate (Serverless Containers)**
- **Option B: AWS EKS (Managed Kubernetes)**

But before that, I'd containerize the application

1. Create a Dockerfile

Create a file named `Dockerfile` at the project root with the following content:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Install system dependencies (FFmpeg is needed for video processing)
RUN apt-get update && \
    apt-get install -y ffmpeg && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file and install Python dependencies
COPY requirements.txt /app/
RUN pip install --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app/
```

```
# Define environment variables (if needed)
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# (Optional) Expose a port if your application includes a web
interface (not the case here)
# EXPOSE 8080

# Set the container entrypoint to run your pipeline (adjust as needed)
ENTRYPOINT ["python", "main.py"]
```

Notes:

- We install FFmpeg in the container so that our subprocess calls can work.
- We can adjust the `ENTRYPOINT` if the pipeline or microservice requires additional arguments.

2. Create a `requirements.txt`

I'd create a `requirements.txt` includes all required packages. For example:

```
scenedetect==0.6.1
opencv-python==4.7.0.72
```

At that point, we can now start to push to AWS ECR (Elastic Container Registry), which is a managed Docker container registry service that makes it simple to store, manage, and deploy container images.

3. Create an ECR Repository

Log in to the AWS Management Console or use the AWS CLI:

```
aws ecr create-repository --repository-name my-h2v-project
```

4. Authenticate Docker to the ECR Registry

5. Build and Tag our Docker Image

Build the Docker image locally:

```
docker build -t my-h2v-project .
```

Then tag the image for ECR:

```
docker tag my-h2v-project:latest
123456789012.dkr.ecr.us-east-1.amazonaws.com/my-h2v-project:latest
```

6. Push the Image to ECR

Push the tagged image:

```
docker push
123456789012.dkr.ecr.us-east-1.amazonaws.com/my-h2v-project:latest
```

We can deploy the containerized application using either **AWS ECS (Fargate)** or **AWS EKS (Kubernetes)**. I will choose fargate

7. Deploy Using AWS ECS (Fargate)

Create an ECS Cluster

We use the AWS Management Console to create an ECS cluster

Define a Task Definition

We create a task definition for your container, for example:

```
{
  "family": "my-h2v-task",
  "executionRoleArn":
"arn:aws:iam::<ACCOUNT_ID>:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "my-h2v-container",
```

```
      "image":
"123456789012.dkr.ecr.us-east-1.amazonaws.com/my-h2v-project:latest",
      "cpu": 512,
      "memory": 1024,
      "essential": true,
      "command": ["python", "main.py"],
      "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
              "awslogs-group": "/ecs/my-h2v-project",
              "awslogs-region": "us-east-1",
              "awslogs-stream-prefix": "ecs"
          }
      }
    }
  ],
  "requiresCompatibilities": ["FARGATE"],
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024"
}
```

Then, we upload or register this task definition in ECS using the AWS Console or AWS CLI, then we run the task.

Another option is to use Kubernetes.

Deploy Using AWS EKS (Kubernetes)

1. Create or Use an Existing EKS Cluster

We can follow AWS EKS Getting Started documentation or use tools like eksctl to create your cluster

2. Create Kubernetes Manifests

Create a deployment manifest `deployment.yaml`

Then can we create a Service if we need external access (e.g., load balancer) but for batch processing we might not need it.

3. Deploy the Application

Apply our manifests to the cluster

Monitor your pods with:

```
kubectl get pods -l app=my-h2v-app
kubectl logs <pod-name>
```

Monitoring and Scaling

In ECS, we will set up CloudWatch Logs for the task.
In EKS, we will use Kubernetes logging tools (or integrate with CloudWatch Container Insights).