AWS Lambda Container Image Deployment Guide

This guide provides a step-by-step walkthrough for deploying your Python web scraping application to AWS Lambda using Docker container images. This method is highly recommended for applications with complex dependencies like pdfplumber, which requires system-level libraries.

Project Structure Assumption:

Your project should be organized as follows:

```
sanction_list_processor/
├── main.py
├── config/
│   └── settings.py
├── utils/
│   ├── aws_s3_utils.py
│   └── common_utils.py
├── UNSC/
│   ├── unsc_web_scraper.py
│   └── unsc_xml_parser.py
├── KDN/
│   ├── kdn_web_scraper.py
│   └── kdn_pdf_parser.py
├── requirements.txt
└── Dockerfile
```

Phase 1: Local Docker Operations & Pushing to ECR

This phase involves building your Docker image locally and pushing it to Amazon Elastic Container Registry (ECR), where AWS Lambda can access it.

Prerequisites:

- **Docker Desktop Installed and Running:** Verify your installation by running docker --version in your terminal.
- **AWS CLI Installed and Configured:** Verify with aws --version and aws configure. Ensure your AWS IAM user has permissions to create ECR repositories and push images (e.g., AmazonEC2ContainerRegistryFullAccess policy or a custom policy with specific ecr actions).

- **Navigate to Project Root:** Open your terminal/command prompt and navigate to your sanction_list_processor directory (the one containing your Dockerfile).

1. Build the Docker Image Locally

This command reads your Dockerfile and creates a Docker image on your local machine.

docker build -t sanction-list-processor .

- -t sanction-list-processor: Tags your image with a name (sanction-list-processor) and defaults to the latest version. This is the name you'll refer to locally.
- .: Specifies that the Dockerfile is located in the current directory.
- **What to expect:** You'll see output showing each step defined in your Dockerfile being executed (e.g., downloading base image, installing poppler-utils, installing Python dependencies, copying code). This can take several minutes the first time.

2. Create an Amazon ECR Repository

You need a dedicated repository in AWS ECR to store your Docker image.

aws ecr create-repository --repository-name sanction-list-processor --region <YOUR_AWS_REGION>

- --repository-name sanction-list-processor: The name for your ECR repository. It's good practice to match this with your local Docker image tag.
- --region <YOUR_AWS_REGION>: **Crucial!** Specify the AWS region where you want to create the repository and later deploy your Lambda function (e.g., ap-southeast-1, us-east-1).
- **What to expect:** A JSON output confirming the repository creation, including its repositoryUri. **Copy this URI; you'll need it for tagging and pushing.**
    - Example URI: 123456789012.dkr.ecr.ap-southeast-1.amazonaws.com/sanction-list-processor

3. Authenticate Docker with ECR

Your local Docker client needs temporary credentials to authenticate with your ECR registry before it can push images.

aws ecr get-login-password --region <YOUR_AWS_REGION> | docker login --username AWS --password-stdin

<YOUR_AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_AWS_REGION>.amazonaws.com

- <YOUR_AWS_ACCOUNT_ID>: Your 12-digit AWS account ID.
- <YOUR_AWS_REGION>: The region where your ECR repository is located.
- **What to expect:** Login Succeeded message.

4. Tag the Docker Image for ECR

You need to tag your locally built image with the full ECR repository URI so Docker knows where to push it.

docker tag sanction-list-processor:latest <YOUR_AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_AWS_REGION>.amazonaws.com/sanction-list-processor:latest

- sanction-list-processor:latest: Your local image name and tag.
- <YOUR_AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_AWS_REGION>.amazonaws.com/sanction-list-processor:latest: The full URI of your ECR repository, including the desired image tag (latest).
- **What to expect:** No output on success. This command simply creates an alias.

5. Push the Docker Image to ECR

This command uploads your tagged Docker image from your local machine to your ECR repository.

docker push <YOUR_AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_AWS_REGION>.amazonaws.com/sanction-list-processor:latest

- **What to expect:** Output showing the progress of layers being pushed. This can take several minutes depending on image size and network speed. Once complete, the image is available for Lambda.

Phase 2: AWS Lambda Function Configuration Checklist

Now that your Docker image is in ECR, you can create or update your Lambda function to use it.

Accessing the AWS Lambda Console:

1. Log in to the AWS Management Console.

2. Search for "Lambda" and navigate to the service.
3. Ensure you are in the **correct AWS Region** (top right dropdown) that matches your ECR repository.

Configuration Steps:

1. **Create Function / Update Code:**
   - **New Function:** Click "Create function" -> "Container image" -> "Browse images". Select your sanction-list-processor repository and the latest tag.
   - **Existing Function:** Go to your function's "Code" tab, then click "Deploy new image" or the "Edit" button next to "Container image". Select the new image from ECR.
2. **Function Name:** Choose a descriptive name (e.g., SanctionListProcessorFunction).
3. **Container Image URI:** Select the image you pushed to ECR (e.g., sanction-list-processor:latest).
4. **Architecture:**
   - For most cases, x86_64 is fine and widely compatible.
   - arm64 (Graviton2) can offer cost savings and performance benefits, but ensure your base image and all dependencies are compatible with arm64. Stick to x86_64 if unsure.
5. Execution Role (IAM Permissions):
   This is the most critical part for your function to access other AWS services.
   - **Creation:** When creating a new function, choose "Create a new role with basic Lambda permissions". This grants permissions to write logs to CloudWatch.
   - **Adding Permissions:** After creation, go to the function's "Configuration" tab -> "Permissions" -> Click on the "Role name". This opens the IAM console.
   - **Required Policies:** Attach policies to this role:
     - **CloudWatch Logs:** The default role usually includes AWSLambdaBasicExecutionRole which grants logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents. **Do NOT remove this.**
     - **Amazon S3:** Your function needs permissions to read/write to your S3 bucket.
       - **For testing/learning:** Attach AmazonS3FullAccess.
       - **For production (Recommended - Least Privilege):** Create a custom IAM policy that grants only the specific actions on your specific S3 bucket:
         {
           "Version": "2012-10-17",

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::<YOUR_S3_BUCKET_NAME>",
      "arn:aws:s3:::<YOUR_S3_BUCKET_NAME>/*"
    ]
  }
]
}
```

Replace <YOUR_S3_BUCKET_NAME> with your actual bucket name.

6. **Memory (Configuration -> General configuration):**
   - **Recommendation:** Start with **512 MB to 1024 MB**. pdfplumber and PDF processing can be memory-intensive. You can monitor CloudWatch metrics after runs to fine-tune.
7. **Timeout (Configuration -> General configuration):**
   - **Recommendation:** Start with at least **1 minute (60 seconds)**, or even **3 minutes (180 seconds)**. Web scraping (network latency) and PDF parsing can take time. Adjust based on actual execution times.
8. **Environment Variables (Configuration -> Environment variables):**
   - These are crucial for your Python code to pick up configuration values.
   - Click "Edit" -> "Add environment variable".
   - **Key:** S3_BUCKET_NAME
     - **Value:** The exact name of your S3 bucket (e.g., unsc-kdn-json-bucket).
   - **Key:** APP_AWS_REGION
     - **Value:** The AWS region where your S3 bucket is located and where your Lambda is running (e.g., ap-southeast-1).
     - **Important: Do NOT use AWS_REGION** as the key, as it's a reserved Lambda environment variable.
9. **Triggers (Optional, for Automated Runs):**
   - If you want your Lambda to run on a schedule (e.g., daily), add an **EventBridge (CloudWatch Events)** trigger.
   - Configure a Schedule rule with a cron expression (e.g., cron(0 0 * * ? *) for

daily at midnight UTC).

Phase 3: Amazon S3 Configuration Checklist

Your S3 bucket is where your processed JSON files will be stored.

1. **Bucket Creation:** Ensure you have an S3 bucket created in your AWS account.
2. **Bucket Name:** The name of this bucket **must exactly match** the value you set for the S3_BUCKET_NAME environment variable in your Lambda function.
3. **Region Consistency:** The S3 bucket should ideally be in the **same AWS Region** as your Lambda function for optimal performance and to avoid cross-region data transfer costs.
4. **Permissions:** As mentioned in the Lambda IAM section, your Lambda function's execution role needs appropriate s3:GetObject, s3:PutObject, and s3:ListBucket permissions for this bucket.
5. **Versioning (Optional but Recommended):** Consider enabling S3 bucket versioning to keep multiple versions of your JSON files, providing a safeguard against accidental overwrites or deletions.

Phase 4: Amazon ECR Configuration Checklist

ECR is where your Docker image resides.

1. **Repository Creation:** Ensure you have an ECR repository created (e.g., sanction-list-processor).
2. **Repository Name:** The name used in your docker build and docker tag commands should match the ECR repository name.
3. **Region Consistency:** The ECR repository **must be in the same AWS Region** as your Lambda function.
4. **IAM Permissions for Pushing:** The IAM user/role you use to run the aws ecr commands locally needs permissions to create repositories, authenticate, and push images.
5. **Lifecycle Policies (Optional):** For long-running projects, consider configuring ECR lifecycle policies to automatically clean up old, untagged, or unused image versions to save storage costs.

Phase 5: IAM (Identity and Access Management) Checklist

IAM controls who can do what in your AWS account.

1. **User/Role for Deployment (Your Local User):**
    - The IAM user you configure with aws configure needs permissions to:
        - Create ECR repositories (ecr:CreateRepository).
        - Push images to ECR (ecr:GetAuthorizationToken, ecr:PutImage, etc.).

        - Create Lambda functions (lambda:CreateFunction).
        - Manage Lambda function configurations (lambda:UpdateFunctionConfiguration, lambda:UpdateCode).
        - Create/manage IAM roles and policies if you're setting up the Lambda execution role yourself (iam:CreateRole, iam:AttachRolePolicy, iam:PutRolePolicy).
    - **Recommendation:** For initial setup, AdministratorAccess is often used, but for production, create a custom policy with only the necessary permissions (least privilege).
2. **Lambda Execution Role:**
    - This is the role that your Lambda function *assumes* when it runs. It determines what AWS services your function can interact with.
    - **Required Permissions:**
        - logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents (for CloudWatch Logs).
        - s3:GetObject, s3:PutObject, s3:ListBucket (for your S3 bucket).
    - **Least Privilege Principle:** Always strive to grant only the minimum necessary permissions. Avoid * resources or FullAccess policies in production.

Troubleshooting Tips:

- **Check CloudWatch Logs:** This is your best friend for debugging Lambda functions. Go to your Lambda function -> "Monitor" tab -> "View logs in CloudWatch". Look for error messages and stack traces.
- **Verify IAM Permissions:** Most "Access Denied" errors indicate missing IAM permissions. Double-check both your local user's permissions (for deployment) and your Lambda execution role's permissions (for runtime).
- **Memory/Timeout:** If your function times out or runs out of memory, increase these settings in the Lambda configuration.
- **Region Consistency:** Ensure all services (Lambda, S3, ECR) are in the same AWS region.
- **Environment Variables:** Double-check that S3_BUCKET_NAME and APP_AWS_REGION are correctly set in your Lambda function's environment variables and match your code.
- **Docker Image Integrity:** If the function fails to start, ensure your Dockerfile correctly installs all dependencies, especially poppler-utils. You can test the image locally by running docker run -it --entrypoint /bin/bash <your-image-id> and checking if pdftotext (part of poppler-utils) is available.