**Reading PDF Paragraphs — Guía de uso y debug**

Servicio que **extrae y clasifica párrafos** desde un PDF (PyMuPDF) y se ejecuta como **AWS Lambda** empacada en **Docker**. Incluye ejecución local (Python), emulación del runtime de Lambda con Docker y exposición por **API Gateway**.

**Estructura del repo**

.

├─ pdf\_reader/ # parsing del PDF

├─ ml/ # heurísticas de clasificación

├─ pdf\_paragraphs\_lambda.py # handler de Lambda

├─ requirements.txt

├─ Dockerfile

├─ deploy.sh # build + push + update Lambda

├─ sample.pdf # PDF de prueba

├─ payload.json # payload de ejemplo

└─ README.md

**Prerrequisitos**

* Python **3.12**
* **Docker** instalado
* **AWS CLI** configurada (aws configure) con permisos para ECR, Lambda y API Gateway
* Cuenta AWS con **Region** y **Role** para Lambda

**Variables de entorno**

Ajusta y exporta antes de usar:

export AWS\_REGION=us-east-1

export ACCOUNT\_ID=<TU\_ACCOUNT\_ID>

export REPO=reading-pdf-paragraphs

export FUNCTION\_NAME=reading-pdf-paragraphs

# Si tienes API Gateway:

export API\_ID=<api\_id\_http>

export API\_ENDPOINT="https://${API\_ID}.execute-api.${AWS\_REGION}.amazonaws.com"

**Uso local (Python, sin Docker)**

**1) Activar entorno y deps**

source .venv/bin/activate

pip install -r requirements.txt

**2) Generar payload.json (PDF en base64)**

En macOS/zsh, usa --data-binary @payload.json para evitar “argument list too long”.

base64 -i sample.pdf -b 0 > pdf.b64

jq -n --arg pdf "$(cat pdf.b64)" --arg pages "1-5" \

'{pdf\_base64:$pdf, pages:$pages}' > payload.json

**3) Ejecutar el handler como script**

python pdf\_paragraphs\_lambda.py

**Alternativa: invocar la función desde Python**

python - <<'PY'

import json

from pdf\_paragraphs\_lambda import lambda\_handler

event = {

"requestContext": {"http": {"method": "POST"}},

"isBase64Encoded": False,

"body": json.dumps(json.load(open("payload.json","r",encoding="utf-8")))

}

resp = lambda\_handler(event, None)

print(json.dumps(resp, ensure\_ascii=False))

PY

**Guardar salida a archivo**

python pdf\_paragraphs\_lambda.py > response.json

**Build y deploy (Docker + ECR + Lambda)**

El script **deploy.sh** hace **build**, **push** a ECR y **update** de la Lambda:

sh deploy.sh

Cambiar etiqueta (tag):

TAG=lambda-v2 sh deploy.sh

**Emular Lambda localmente (Docker runtime de AWS)**

1. Levantar el runtime con tu imagen:

docker run --rm -p 9000:8080 pdf-paragraphs:lambda-v1

1. Crear un evento HTTP API v2 con tu payload:

printf '{ "requestContext": {"http": {"method":"POST"}}, "isBase64Encoded": false, "body": %s }\n' \

"$(jq -c . payload.json)" > event-local.json

1. Invocar localmente:

curl -s -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" \

-H "Content-Type: application/json" \

--data-binary @event-local.json | python3 -m json.tool

**Invocar la Lambda real (AWS CLI)**

Reutiliza event-local.json (envelope HTTP API v2):

aws lambda invoke \

--function-name "$FUNCTION\_NAME" \

--payload fileb://event-local.json \

--cli-binary-format raw-in-base64-out \

response.json

cat response.json | python3 -m json.tool

**Invocar por API Gateway**

**Endpoint**:  
POST $API\_ENDPOINT/prod/extract\_paragraphs

**Body**: el contenido de payload.json (tiene pdf\_base64 y pages).

curl -s -X POST "$API\_ENDPOINT/prod/extract\_paragraphs" \

-H "Content-Type: application/json" \

--data-binary @payload.json | python3 -m json.tool

**Postman**

* **Method**: POST
* **URL**: https://<API\_ID>.execute-api.<REGION>.amazonaws.com/prod/extract\_paragraphs
* **Headers**: Content-Type: application/json
* **Body**: raw → JSON → pega payload.json.

**Logs en CloudWatch**

**Tail en vivo:**

aws logs tail /aws/lambda/$FUNCTION\_NAME --follow --region "$AWS\_REGION"

**Último log stream:**

aws logs describe-log-streams \

--log-group-name /aws/lambda/$FUNCTION\_NAME \

--order-by LastEventTime \

--descending \

--limit 1 --region "$AWS\_REGION"

**Ver eventos de un stream:**

aws logs get-log-events \

--log-group-name /aws/lambda/$FUNCTION\_NAME \

--log-stream-name '<LOG\_STREAM\_NAME>' \

--region "$AWS\_REGION"

**Troubleshooting**

* **Internal Server Error** vía API Gateway
  + Revisa los **logs** en CloudWatch.
  + Usa el campo **pages** (no page\_range).
  + Si invocas Lambda directa con AWS CLI: el evento debe ser un **envelope** HTTP API v2.
  + Evita pegar base64 inline en el shell; usa --data-binary @payload.json.
* **“zsh: argument list too long”**
  + Usa payload.json + --data-binary @payload.json.
* **Imagen no soportada por Lambda (media type)**
  + La etiqueta que uses en ECR debe ser **single-arch** y ECR debe mostrar  
    application/vnd.oci.image.manifest.v1+json.
  + Si ves un *manifest list*, borra esa imagen y vuelve a pushear la single-arch.
* **Permisos API Gateway → Lambda**
  + Asegúrate de haber agregado el lambda:add-permission con source-arn que  
    apunte a tu API\_ID y ruta POST /extract\_paragraphs.
* **Timeout / rendimiento**
  + Ajusta memoria/timeout:
  + aws lambda update-function-configuration \
  + --function-name "$FUNCTION\_NAME" \
  + --memory-size 1024 \
  + --timeout 60 \
  + --region "$AWS\_REGION"

**Limpieza (opcional)**

**Borrar una imagen/tag en ECR:**

aws ecr batch-delete-image \

--repository-name "$REPO" \

--image-ids imageTag=lambda-v1 \

--region "$AWS\_REGION"

**Borrar una API Gateway de prueba:**

aws apigatewayv2 delete-api --api-id "$API\_ID" --region "$AWS\_REGION"

**Notas**

* Mantén el README actualizado si cambian el **payload** o la **ruta del API**.
* Agrega un .env.example con las variables de entorno que sueles exportar.
* Para cada cambio de código: ejecuta sh deploy.sh (o TAG=lambda-vN sh deploy.sh) y vuelve a probar.