El nombre de las lambdas debe iniciar por **iepc-lambda-**

Comprehend

Código Python

```
import json
import boto3

REGION = "us-east-1"
comprehend = boto3.client("comprehend", region_name=REGION)

DEFAULT_TEXT = (
    "El pasado 12 de febrero de 2026 acudí a la oficina regional del Instituto Electoral en Durango. "
    "El personal fue amable, pero el sistema presentó fallas durante casi una hora. "
    "Aun así, completé el trámite y recibí un comprobante firmado por la Lic. Mariana Torres."
)

def lambda_handler(event, context):
    # Permite mandar texto por event, o usar el default
    text = (event or {}).get("text") or DEFAULT_TEXT

    # 1) Idioma dominante
    lang_resp = comprehend.detect_dominant_language(Text=text)
    lang = max(lang_resp["Languages"], key=lambda x: x["Score"])["LanguageCode"]

    # 2) Sentimiento
    sent_resp = comprehend.detect_sentiment(Text=text, LanguageCode=lang)

    # 3) Entidades
    ent_resp = comprehend.detect_entities(Text=text, LanguageCode=lang)

    # 4) Frases clave
    kp_resp = comprehend.detect_key_phrases(Text=text, LanguageCode=lang)

    result = {
        "language": lang,
        "sentiment": sent_resp["Sentiment"],
        "sentimentScore": sent_resp["SentimentScore"],
        "entities": [
```

```
            {"text": e["Text"], "type": e["Type"], "score":
float(e["Score"])}
            for e in ent_resp["Entities"]
        ],
        "keyPhrases": [
            {"text": k["Text"], "score": float(k["Score"])}
            for k in kp_resp["KeyPhrases"]
        ],
    }

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json; charset=utf-8"},
        "body": json.dumps(result, ensure_ascii=False)
    }
```

Permisos Comprehend

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowComprehendRealTime",
      "Effect": "Allow",
      "Action": [
        "comprehend:DetectDominantLanguage",
        "comprehend:DetectSentiment",
        "comprehend:DetectEntities",
        "comprehend:DetectKeyPhrases"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowCloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

SageMaker

Código Python

```
import os
import json
import boto3

runtime = boto3.client("sagemaker-runtime", region_name="us-east-1")

ENDPOINT_NAME = os.environ.get(
    "ENDPOINT_NAME",
    "jumpstart-***"
)

def _resp(status, body_obj):
    return {
        "statusCode": status,
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Headers":
"content-type,authorization",
            "Access-Control-Allow-Methods": "POST,OPTIONS"
        },
        "body": json.dumps(body_obj, ensure_ascii=False)
    }

def lambda_handler(event, context):
    if event.get("requestContext", {}).get("http", {}).get("method") ==
"OPTIONS":
        return _resp(200, {"ok": True})

    try:
        body_raw = event.get("body") or "{}"
        body = json.loads(body_raw) if isinstance(body_raw, str) else
body_raw

        prompt = body.get("prompt") or body.get("inputs")
        if not prompt or not isinstance(prompt, str):
            return _resp(400, {"error": "Body must include 'prompt'
(string) or 'inputs' (string)."})

        params_in = body.get("parameters") or {}
```

```
        parameters = {
            "max_new_tokens": int(params_in.get("max_new_tokens",
180)),
            "temperature": float(params_in.get("temperature", 0.3)),
            "top_p": float(params_in.get("top_p", 0.9)),
            "do_sample": bool(params_in.get("do_sample", True)),
            "return_full_text": bool(params_in.get("return_full_text",
False))
        }

        sm_payload = {"inputs": prompt, "parameters": parameters}

        resp = runtime.invoke_endpoint(
            EndpointName=ENDPOINT_NAME,
            ContentType="application/json",
            Body=json.dumps(sm_payload).encode("utf-8"),
        )

        raw = resp["Body"].read().decode("utf-8")

        try:
            parsed = json.loads(raw)
        except Exception:
            parsed = raw

        return _resp(200, {
            "endpoint": ENDPOINT_NAME,
            "request": sm_payload,
            "response": parsed
        })

    except Exception as e:
        return _resp(500, {"error": str(e)})
```

Permisos ejecutar SageMaker

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Sid": "InvokeSageMakerEndpoint",
                        "Effect": "Allow",
                        "Action": "sagemaker:InvokeEndpoint",
```

```
                "Resource":
"arn:aws:sagemaker:us-east-1:251342503808:endpoint/jumpstart-***"
            }
        ]
}
```