Plano do projeto

Lembrar para o Readme

estrutura do projeto no repo:

```
project-root/
│
├── README.md                          ← Ponto de entrada
│
├── src/
│   ├── ingestion_api/
│   ├── crawler/
│   ├── transformations/
│   ├── metadata/
│   └── analysis/
│
├── docs/
│   ├── final_documentation.md         ← Documento técnico completo
│   ├── diagram.png or diagram.md      ← Arquitetura
│   └── presentation.pdf               ← Walkthrough final
│
├── cloud/
│   ├── instructions.md                ← Como rodar na nuvem
│   ├── iam_policies.txt
│   ├── sample_env_vars.txt
│   └── (opcional) Terraform/CloudFormation
│
├── requirements.txt
└── .gitignore
```

Quais serão os inputs e outputs do projeto:

API Source

- **World Bank Indicators — GDP per capita**

  URL:

  `https://api.worldbank.org/v2/country/all/indicator/NY.GDP.PCAP.CD?format=json`

Crawler Source

- **Wikipedia — CO_2 emissions per capita**

  URL:

https://en.wikipedia.org/wiki/List_of_countries_by_carbon_dioxide_emissions_per_capita

obs: vejo na página que as colunas da tabela são: um valor numérico de id, % of global average, emissions per capita (tons per year) (subdividido nos anos 2023 e 2000), % change from 2000.

Curated Dataset

- Tabela combinada country–year–**gdp_per_capita_usd–co2_tons_per_capita** (com country_name como coluna descritiva)

Analytical Output

- **Artefato 1 –** `gdp_vs_co2_scatter.png`
  - Scatterplot usando **apenas 2023**:
    - eixo X: `gdp_per_capita_usd`
    - eixo Y: `co2_tons_per_capita`
    - cor ou tamanho guiado por `co2_per_1000usd_gdp` .
  - Comentário na doc/apresentação:

    "Este gráfico mostra a relação entre renda per capita e emissões de CO_2 per capita em 2023."
- **Artefato 2 –** `correlation_summary.csv`
  - Campos sugeridos:
    - `year`
    - `pearson_correlation_gdp_co2`
    - `top5_countries_highest_co2_per_1000usd_gdp`
    - `top5_countries_lowest_co2_per_1000usd_gdp`
  - Você pode colocar tudo em 2 linhas (ano 2000 e ano 2023).

Observação: a tabela da Wikipedia contém emissões per capita apenas para os anos 2000 e 2023. Assim, o curated dataset e o scatterplot comparam GDP per capita e CO_2 apenas nesses dois anos.

Execução local vs execução na nuvem (linhas gerais)

Durante o desenvolvimento, o pipeline é executado localmente para permitir testes rápidos das funções de ingestão, transformação e curation. A execução local utiliza credenciais AWS configuradas no ambiente do desenvolvedor e grava diretamente no bucket S3 e na tabela DynamoDB reais ou em ambientes de teste.

Após validação local, o projeto é empacotado para execução em nuvem dentro do AWS Lambda. O empacotamento consiste em incluir as dependências Python (requirements.txt) e o código-fonte em um bundle ZIP, respeitando as limitações de tamanho e arquitetura do

Lambda. A função Lambda torna-se o executor oficial do pipeline: recebe agendamentos via EventBridge, lê checkpoints no DynamoDB e grava os outputs no S3 conforme definido nas camadas raw, processed e curated.

Essa separação permite desenvolvimento ágil localmente, mantendo ao mesmo tempo uma execução totalmente automatizada, incremental, rastreável e acionada na nuvem, conforme exigido no case.

Considerações operacionais: erros, retry e idempotência

O pipeline adota práticas padrão de resiliência para garantir execuções consistentes mesmo em caso de falhas na API, instabilidade da rede ou inconsistências no HTML da Wikipedia.

Erros e logging

- Todas as etapas (ingestão, transformação, curation, analytics) registram mensagens estruturadas no CloudWatch.
- Cada execução possui um *ingestion_run_id* que aparece tanto nos logs quanto no DynamoDB, facilitando rastreamento.

Retry automático

- Chamadas à World Bank API utilizam tentativas adicionais em caso de erros temporários (HTTP 429/5xx).
- O crawler reexecuta o download da página uma vez caso encontre HTML inesperado ou falha de parsing.

Fallback behavior

- Se a API estiver indisponível, o pipeline registra falha mas não sobrescreve dados anteriores.
- Se o HTML da Wikipedia vier inválido (problema pontual), a execução é abortada para evitar gerar dados incorretos.

Idempotência

- Cada etapa grava saídas identificadas por *ingestion_date* ou *snapshot_date*, garantindo que reruns não sobrescrevam resultados de runs anteriores.
- O processamento usa *record_hash* para detectar alterações reais e evitar reprocessamento desnecessário.
- Checkpoints no DynamoDB asseguram que execuções repetidas não recarreguem anos já processados.

Esses mecanismos atendem aos requisitos de robustez, rastreabilidade e segurança operacional do case.

Definição dos schemas:

1 - RAW

1.1. RAW – World Bank (GDP per capita)

Nome do dataset/"tabela": `raw_worldbank_gdp_per_capita`

Campos (pensando em um JSON/Parquet simples):

- `ingestion_run_id` – string
- `ingestion_ts` – timestamp
- `data_source` – string (`"world_bank_api"`)
- `countryiso3code` – string (BRA, USA, etc.)
- `country.value` – string (Brazil, United States, etc.)
- `indicator.id` – string (Indicador escolhido: NY.GDP.PCAP.CD (GDP per capita, current US$))
- `indicator.value` – string (nome textual do indicador)
- `date` – string (ex.: `"2020"`)
- `value` – number ou string (valor do GDP per capita)
- `raw_payload` – JSON/texto completo da resposta original
- `record_hash` - Hash (ex.: MD5, SHA1) do registro original.
    - útil para:
        - detectar mudanças entre versões
        - ver se um registro atualizado realmente mudou
        - implementar **incremental updates baseado em checksum**
        - `record_hash = SHA1(json.dumps(record, sort_keys=True))`
- `raw_file_path` - Pra rastrear exatamente de qual arquivo o processed veio

explodir o JSON em linhas, mas guarda o JSON da linha em `raw_payload`. Não precisa guardar a resposta inteira da API num único blob.

1.2. RAW – Wikipedia (CO_2 per capita)

Nome: `raw_wikipedia_co2_per_capita`

Campos:

- `ingestion_run_id` – string
- `ingestion_ts` – timestamp
- `data_source` – string (`"wikipedia_co2"`)
- `page_url` – string
- `table_html` – texto/HTML bruto da tabela

- `raw_table_json` – JSON com as lista de linhas já parseada (ainda “suja”)
- `record_hash` - Hash (ex.: MD5, SHA1) do registro original.
  - útil para:
    - detectar mudanças entre versões
    - ver se um registro atualizado realmente mudou
    - implementar **incremental updates baseado em checksum**
- `raw_file_path` - Pra rastrear exatamente de qual arquivo o processed veio

2. PROCESSED

2.1. PROCESSED - GDP per capita (World Bank)

Nome: `processed_worldbank_gdp_per_capita`

Campos:

- `country_code` – string (BR/USA → BRA/USA)
- `country_name` – string
- `year` – int (ex.: 2020)
- `gdp_per_capita_usd` – float
- `indicator_id` – string (Indicador escolhido: NY.GDP.PCAP.CD (GDP per capita, current US$))
- `indicator_name` – string
- `ingestion_run_id` – string
- `ingestion_ts` – timestamp
- `data_source` – string (fixo `"world_bank_api"`)

Chave lógica:
`(country_code, year)`

Particionamento:
`year=<ano>`
obs: Os campos `countryiso3code` e `country.value` do RAW são renomeados para `country_code` e `country_name` no processed.”

Incremental ingestion – estratégia detalhada (World Bank)

O carregamento da API do World Bank utiliza checkpoint por ano, armazenado no DynamoDB.

Estratégia:

1. DynamoDB mantém o campo **last_year_loaded_world_bank** (ex.: 2020).

2. A cada execução, o pipeline solicita à API a lista de anos disponíveis para o indicador selecionado.
3. O pipeline filtra somente anos **maiores que o checkpoint**.
4. Somente esses anos são ingeridos em RAW e processados em PROCESSED.
5. Ao final da execução bem-sucedida, o checkpoint é atualizado no DynamoDB.

Essa lógica garante ingestão incremental real, evita recarregar anos já processados e cumpre o requisito do case.

2.2. PROCESSED - CO_2 per capita (Wikipedia)

Nome: `processed_wikipedia_co2_per_capita`

Aqui já deixamos resolvido coisas como:

- Footnotes do tipo `5.2[a]`
- Traços para missing data (`–`)
- Conversão para número
- algo mais?

Campos:

- `country_name` – string (nome exato da tabela)
- `country_name_normalized` – string (sem acentos, lower, etc. para join)
- `country_code` – string (ISO3, derivado via mapping a partir de country_name_normalized)
- `year` – int (neste caso, 2000 ou 2023, vindos das colunas de emissões por ano da tabela da Wikipedia)
- `co2_tons_per_capita` – float
- `notes` – string (se você quiser guardar footnote/observações)
- `ingestion_run_id` – string
- `ingestion_ts` – timestamp
- `data_source` – string (fixo `"wikipedia_co2"`)

Chave lógica:
(`country_code, year`)

Observação:
Durante a transformação, criamos um mapping de country_name_normalized → country_code
para conseguir usar sempre country_code como chave de join com o World Bank.

2.2.1. **Despivotagem da tabela de CO_2 (caso específico 2000/2023)**

A tabela da Wikipedia traz, para cada país, duas colunas de emissões per capita: uma para o ano 2000 e outra para o ano 2023.

No processed, queremos um formato "longo", com uma linha por `(country, year)`.

A partir das colunas:

- `emissions_2000`
- `emissions_2023`

fazemos um "melt" gerando as colunas:

- `year` (2000 ou 2023)
- `co2_tons_per_capita` (valor de emissões correspondente àquele ano)

Dessa forma, cada país passa a ter até duas linhas: uma para 2000, outra para 2023, o que facilita o join com o World Bank (que também traz uma linha por `country_code, year`).

2.2.2. Mapping de países (country_name_normalized → country_code)

O objetivo é usar sempre `country_code` (ISO3) como chave de join entre World Bank e Wikipedia. Para isso, transformamos o `country_name` da Wikipedia em `country_code`.

Função de normalização (reutilizável):

Definimos uma função única, usada em ambos os lados, por exemplo:

- lower case
- remover acentos
- remover pontuação
- trocar múltiplos espaços por 1
- trim nas pontas

Poderia ser algo como:

```
def normalize_name(s):
    s = remove_accents(s.lower())
    s = re.sub(r'[^a-z0-9 ]+', '', s)
    s = re.sub(r'\s+', ' ', s)
    return s.strip()
```

Isso gera `country_name_normalized`.

Passo 1 – Gerar tabela de mapping a partir do World Bank

1. No processed do World Bank (`processed_worldbank_gdp_per_capita`), extraímos a lista de países distinta:
    - colunas: `country_code`, `country_name`.

2. Aplicamos a função de normalização em `country_name` para criar:
    - `country_name_normalized`.
3. Gravamos isso como uma pequena tabela de apoio:
    - Nome: `processed_country_mapping`
    - Campos:
        - `country_name_normalized` – string (PK)
        - `country_code` – string (ISO3)
        - `country_name` – string (nome "oficial" usado no curated)
        - `source_precedence` – string (opcional, ex.: `"world_bank"`)

Passo 2 – Overrides manuais (para casos chatos)

Alguns nomes podem não bater exatamente entre World Bank e Wikipedia (ex.: "Côte d'Ivoire" vs "Ivory Coast", "Republic of Korea" vs "South Korea", agregados tipo "European Union"). Então:

1. Mantemos um arquivo estático no repo, algo como:
    - `src/transformations/country_mapping_overrides.csv`.
2. Campos:
    - `country_name_normalized`
    - `country_code`
    - `country_name`
3. Na hora de montar o mapping final, fazemos:
    - mapping_base = mapping do World Bank
    - aplicar overrides por cima (overrides têm prioridade).

Passo 3 – Aplicar mapping na Wikipedia

Na transformação da Wikipedia:

1. Para cada linha de `processed_wikipedia_co2_per_capita`:
    - Já temos `country_name` e `country_name_normalized`.
2. Fazemos um left join com `processed_country_mapping` por `country_name_normalized`.
3. Preenchemos:
    - `country_code` e `country_name` (canonical).
4. Registros sem match:
    - Podem ser:
        - descartados do curated (ex.: agregados regionais como "World", "European Union"), **e**

    - logados em uma tabela de erro ou em um campo `notes` / log no CloudWatch / DynamoDB para auditoria.

Resultado:

- Wikipedia passa a ter `country_code` compatível com o World Bank.
- O curated `curated_econ_environment_country_year` faz join limpo por `(country_code, year)`.

2.2.3. Persistência do mapping de países (decisão final)

Para garantir reprodutibilidade e facilitar auditoria e debugging, o mapping final de países **será persistido no S3** como uma tabela processada.

Local no bucket S3:

```
processed/country_mapping/country_mapping.parquet
```

Conteúdo do arquivo:

- country_name_normalized
- country_code
- country_name
- source_precedence

Esse arquivo é gerado a partir de:

1. Lista de países do World Bank (processed_worldbank_gdp_per_capita)
2. Aplicação dos overrides manuais (arquivo CSV estático abaixo)

2.2.4. Arquivo de overrides manuais (decisão final)

Manteremos um arquivo estático no repositório com possíveis correções manuais:

Local no repo:

```
src/transformations/country_mapping_overrides.csv
```

Campos do CSV:

- country_name_normalized
- country_code
- country_name

Esse arquivo pode começar vazio, mas existe para resolver casos como:

- “Côte d'Ivoire” vs “Ivory Coast”
- “Republic of Korea” vs “South Korea”

- Agregados (“European Union”, “World”)

Precedência:

O mapping final é:

mapping_base (World Bank) → overrides aplicados por cima → gravação final no S3.

3. CURATED

3.1. Economic & Environmental by country-year

Nome: `curated_econ_environment_country_year`

Campos:

- `country_code` – string (chave)
- `country_name` – string
- `year` – int
- `gdp_per_capita_usd` – float
- `co2_tons_per_capita` – float
- `co2_per_1000usd_gdp` – float (campo derivado)
- `gdp_source_system` – string (`"world_bank_api"`)
- `co2_source_system` – string (`"wikipedia_co2"`)
- `first_ingestion_run_id` – string
- `last_update_run_id` – string
- `last_update_ts` – timestamp

Chave lógica:

`(country_code, year)`

Observação: O campo derivado `co2_per_1000usd_gdp` é parte **oficial** do curated dataset. Ele é calculado como:

```
co2_tons_per_capita / (gdp_per_capita_usd / 1000)
```

Esse campo será utilizado posteriormente no Analytical Output (rankings e análises adicionais).

Estratégia de join entre World Bank e Wikipedia

O curated dataset é formado por um **LEFT JOIN** do conjunto econômico (World Bank) com o conjunto ambiental (Wikipedia) usando a chave:

- country_code
- year

Regras de comportamento:

- **Países presentes no World Bank mas ausentes na Wikipedia:**

  são descartados do curated, mas registrados em um log de inconsistências (via notes, CloudWatch ou tabela auxiliar no DynamoDB).

- **Países presentes na Wikipedia mas sem correspondência no World Bank:**

  são automaticamente excluídos, pois não possuem valores de GDP necessários para o indicador derivado.

- **Somente pares (country_code, year) que possuem ambos os indicadores** aparecem no curated, garantindo consistência analítica.

Esse mecanismo assegura que todas as linhas do curated contenham valores válidos de GDP per capita, CO_2 per capita e o indicador combinado co2_per_1000usd_gdp.

Particionamento no bucket:

`curated/env_econ_country_year/year=<ano>/snapshot_date=<YYYY-MM-DD>/curated_econ_environment_country_year.parquet`

Localização no bucket S3:

curated/env_econ_country_year/year=<ano>/snapshot_date=<YYYY-MM-DD>/curated_econ_environment_country_year.parquet

4. Metadados de ingestão (tabela de runs), tabelinha de controle:

Nome: `ingestion_runs`

Campos:

- `ingestion_run_id` – string (PK)
- `run_scope` – string (`"world_bank_api"`, `"wikipedia_co2"`, `"curated_join"`)
- `start_ts` – timestamp
- `end_ts` – timestamp
- `status` – string (`"SUCCESS"`, `"FAILED"`)
- `rows_processed` – int
- `last_checkpoint` – string (ex.: último ano ou página de API)
- `error_message` – string (nullable)

Visão geral da arquitetura

Objetivo:

Um **pipeline em Python**, rodando na **nuvem**, que:

- ingere:
  - **API do World Bank** (GDP per capita – Indicador escolhido: NY.GDP.PCAP.CD (GDP per capita, current US$)
  - **Tabela da Wikipedia** (CO_2 per capita)
- salva em **camadas raw / processed / curated** no S3
- registra **metadata de execução**
- gera um **artefato analítico** (ex.: gráfico + resumo de correlação)

Diagrama:

```
flowchart LR
    subgraph External_Sources[External Data Sources]
        WB[World Bank API\nGDP per capita]
        WIKI[Wikipedia\nCO2 per capita table]
    end

    subgraph AWS
        EB[EventBridge\n(scheduled trigger)]
        LAMBDA[Lambda\nPython Pipeline]
        S3[(S3 Data Lake\nraw / processed / curated)]
        DDB[(DynamoDB\nmetadata & checkpoints)]
        CW[CloudWatch Logs]
        ATHENA[Athena / Notebook\n(consumo analítico)]
    end

    EB --> LAMBDA
    WB --> LAMBDA
    WIKI --> LAMBDA
    LAMBDA --> S3
    LAMBDA --> DDB
    LAMBDA --> CW
    S3 --> ATHENA
```

A ideia é: **um Lambda orquestra tudo**, mas dentro do código você separa em módulos (`api_ingestion.py`, `crawler.py`, `transform.py`, etc.), então fica organizado e fácil de explicar.

Componentes principais

1. Storage: S3 como “data lake”

Um bucket, por exemplo:

`env-econ-pipeline-data`

Estrutura de pastas (prefixos):

```
s3://env-econ-pipeline-data/
  raw/
    world_bank_gdp/
      ingestion_date=2025-11-22/...
    wikipedia_co2/
      ingestion_date=2025-11-22/...
  processed/
    world_bank_gdp/
      year=2020/...
    wikipedia_co2/
      year=2020/...
    curated/
    env_econ_country_year/
        year=2025/
            snapshot_date=2025-11-22/
                curated_econ_environment_country_year.parquet
  analytics/
    snapshot_date=2025-11-22/
      gdp_vs_co2_scatter.png
      correlation_summary.csv
```

- **raw**: resposta bruta da API / HTML da Wikipedia ou JSON/tab separado logo após parsing mínimo.
- **processed**: dados já "limpos" e tipados (colunas padronizadas, tipos numéricos).
- **curated**: join final `country_code, year, gdp_per_capita_usd, co2_tons_per_capita` (com country_name como coluna descritiva).
- **analytics**: artefatos de saída (gráfico, tabelas de resumo).

formato dos arquivos no S3

- RAW → JSONL (API) / HTML + JSON (crawler)
- PROCESSED → Parquet
- CURATED → Parquet
- ANALYTICS → PNG + CSV

2. Compute: AWS Lambda (Python)

Um **AWS Lambda** principal, por exemplo `env-econ-pipeline-lambda`, com código Python contendo:

- `world_bank_ingestion.py`
- `wikipedia_crawler.py`
- `transformations.py`
- `curation.py`
- `analytics.py`
- `metadata.py`
- `handler.py` (ponto de entrada do Lambda)

Fluxo dentro do Lambda (num único run):

1. Lê de DynamoDB o **checkpoint** da API (último ano carregado, por exemplo).
2. Chama World Bank API e baixa apenas anos novos (incremental).

obs: incremental:

- DynamoDB guarda `last_year_loaded_world_bank = 2020`
- Pipeline faz um request para listar os anos disponíveis na API
- Filtra apenas anos > checkpoint
- Processa

1. Salva em `raw/world_bank_gdp/ingestion_date=...`.
2. Faz o parse + limpeza → salva em `processed/world_bank_gdp/`.
3. Faz o crawling da Wikipedia (pode ser full table) e salva o HTML ou tabela crua em `raw/wikipedia_co2/`.
4. Normaliza nomes de países, remove símbolos, converte números → `processed/wikipedia_co2/`.
5. 7. Faz o join para gerar o dataset `curated/env_econ_country_year/year=<ano>/snapshot_date=<YYYY-MM-DD>/...`
6. Gera o **output analítico** (ex.: scatterplot PNG + CSV de correlação) → `analytics/`.
7. Escreve um registro de **metadata de execução** no DynamoDB.
8. Loga tudo no CloudWatch.

Isso tudo acontece numa única execução do Lambda, disparada pelo EventBridge.

3. Orquestração: EventBridge

- **AWS EventBridge Rule** com um cron:
  - `cron(0 2 * * ? *)` → roda todo dia às 02h UTC (justifica como “diário” pra cumprir requisito de agendamento).
- Essa rule chama o Lambda `env-econ-pipeline-lambda`.

Também **execução manual**:

- Rodando o Lambda via console
- Ou via `aws lambda invoke` (bom pra testes)

4. Metadata & Incremental: DynamoDB

Crie uma tabela DynamoDB, por exemplo:
`env_econ_pipeline_metadata`

Com itens do tipo:

1. **Registro de execução (run log)**
    - `PK`: `RUN#2025-11-22T02:00:00Z`
    - `SK`: `METADATA`
    - `run_scope`: `"full_pipeline"`
    - `start_time`, `end_time`
    - `status`: `"SUCCESS"` | `"FAILED"`
    - `rows_world_bank_raw`
    - `rows_world_bank_processed`
    - `rows_wikipedia_raw`
    - `rows_wikipedia_processed`
    - `rows_curated`
    - `last_year_loaded_world_bank`
    - `execution_duration_ms`, etc.
2. **Checkpoints por fonte**
    - `PK`: `CHECKPOINT#WORLD_BANK_GDP`
    - `SK`: `STATE`
    - `last_year_loaded`: `2020`
    - `last_ingestion_ts`: `2025-11-22T02:00:00Z`
    - `PK`: `CHECKPOINT#WIKIPEDIA_CO2`
    - `SK`: `STATE`
    - `last_ingestion_ts`: `2025-11-22T02:00:00Z`

Assim mostramos claramente:

- **Incremental ingestion**: World Bank é carregado só a partir de `last_year_loaded`.
- Web crawler pode ser full load, mostramos que:

- Mantém histórico por `ingestion_date`
- Na transformação/curation sempre considera apenas o snapshot mais recente (ex.: maior `ingestion_date`).

Isso casa com o requisito de **incremental ingestion + metadata tracking**.

5. Logs & Monitoramento: CloudWatch

- **CloudWatch Logs** para o Lambda:
  - Logs de requests na API
  - Logs de erros de parsing do HTML
  - Mensagem final de sucesso + número de linhas processadas

Mencionar na documentação:

- Como encontrar o log group do Lambda
- Como filtrar logs por `RUN_ID` (você pode gerar um UUID no início de cada execução e incluir em todos os logs e na entry do DynamoDB).

6. Camada Analítica: Athena / Notebook

Para consumir o curated dataset:

- Usaremos **AWS Athena** apontando para:
  - `s3://env-econ-pipeline-data/curated/env_econ_country_year/year=<ano>/snapshot_date=<YYYY-MM-DD>/`
- Cria uma tabela externa (Parquet ou CSV)
- Rodar queries tipo:

```
SELECT country_name,
    country_code,
    year,
    gdp_per_capita_usd,
    co2_tons_per_capita
FROM env_econ_country_year
...
```

também citar `co2_per_1000usd_gdp`

E, no código Python, geramos:

- `gdp_vs_co2_scatter.png`
- `correlation_summary.csv` (por exemplo: correlação global + ranking de países).

Também vamos aproveitar o campo `co2_per_1000usd_gdp` para:

- ranking dos países mais eficientes
- mapa ou scatterplot com:
  - eixo X = GDP per capita
  - eixo Y = CO_2 per capita
  - cor ou tamanho = `co2_per_1000usd_gdp`

Isso vira nosso **"Analytical Output" programático**.

IAM, permissões e recursos (para o README / Cloud Execution Instructions)

(essa sessão só precisamos nos preocupar depois, né? Já é algo fora da arquiteturação acima?)

Pra aquela seção "Any IAM roles, permissions, buckets, storage paths, or compute resources" podemos listar:

- **S3 Bucket**
  - `env-econ-pipeline-data`
  - Política: permitir leitura/escrita a partir do Lambda Role
- **IAM Role do Lambda**
  - Nome: `EnvEconPipelineLambdaRole`
  - Permissões:
    - `s3:GetObject`, `s3:PutObject`, `s3:ListBucket` no bucket do pipeline
    - `dynamodb:PutItem`, `dynamodb:GetItem`, `dynamodb:UpdateItem`, `dynamodb:Query` na tabela de metadata
    - `logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents` no CloudWatch
  - (Opcional) `ssm:GetParameter` se quiser pegar configs do Parameter Store
- **DynamoDB table**
  - `env_econ_pipeline_metadata` (provisionado em on-demand mode)
- **Lambda Function**
  - `env-econ-pipeline-lambda`
  - Runtime: Python 3.11
  - Memory: 512–1024 MB (justifique) (como assim justifique?)
  - Timeout: 5–10 min (dependendo do que estimar) (como assim?)
    - **Memória: 512 MB**

        - Justificativa: “512 MB é suficiente para carregar em memória os DataFrames com alguns anos de dados globais do World Bank e a tabela de CO_2 da Wikipedia, realizar joins e gerar os artefatos analíticos, mantendo custo baixo. Para este volume de dados públicos, não é necessário mais do que isso.”
    - **Timeout: 5 minutos**
        - Justificativa: “O pipeline faz chamadas a uma API pública, crawling de uma página estática e algumas transformações moderadas em pandas. Em testes locais, a execução leva bem menos de 1 minuto, então 5 minutos oferecem folga para variações de latência sem risco de timeout prematuro.”
    - mencionar que se o volume crescer ou a API ficar lenta, esses valores podem ser ajustados.
  - Variáveis de ambiente:
    - `S3_BUCKET_NAME`
    - `WORLD_BANK_INDICATOR` = `NY.GDP.PCAP.CD`
    - `WORLD_BANK_COUNTRY` = `all` (ou um conjunto específico)
    - `WIKIPEDIA_URL` = URL da tabela de CO_2
    - `METADATA_TABLE` = `env_econ_pipeline_metadata`
- **EventBridge Rule**
  - Nome: `env-econ-pipeline-daily`
  - Cron: `cron(0 2 * * ? *)`
  - Target: `env-econ-pipeline-lambda`