1. 요청 내용 확인

* 목표: LangGraph(메인 오케스트레이터)로 100개 블로그까지 확장 가능한 자동화 파이프라인을 구축.
* 전제: 모델은 GPT-5(집필/검수) + Claude(요약/벤치) 혼합, 스케줄러는 AWS EventBridge, 실행은 AWS Lambda, 데이터는 Docker 기반 PostgreSQL(EC2 등), 워드프레스는 REST 자동 발행, 티스토리는 반자동 HTML 패키지(S3 링크)로 제공, Make.com은 알림/승인만 담당.
* 요구: 성능 우수, 안정적(자동/반자동), 저렴.

1. 접근 방법 설명(한 줄 요약)

* 로컬에서 그래프를 검증 → Docker로 DB 올림 → Lambda가 LangGraph를 실행해 글을 만들고 WP 발행/티스토리용 HTML을 S3에 업로드 → EventBridge가 매일 호출 → 실패/승인은 Make.com으로 알림.

1. 완전한 코드  
   아래 코드는 “초보자도 그대로 복붙→실행” 기준입니다. Lambda는 ZIP 배포(도커 불필요), 티스토리 패키지는 S3에 저장합니다. 워드프레스는 Application Password(기본 인증) 사용으로 수정했습니다.

* 폴더 구조

.

├─ docker-compose.db.yml

├─ init.sql

├─ .env.example

├─ lambda/

│ ├─ requirements.txt

│ ├─ handler.py

│ └─ render.py

└─ make/

└─ sample\_webhook.json

* 파일: docker-compose.db.yml（PostgreSQL만 분리 실행）

version: '3.8'

services:

postgres:

image: postgres:16-alpine

container\_name: blog\_auto\_db

environment:

POSTGRES\_USER: blog\_user

POSTGRES\_PASSWORD: strong\_password\_here

POSTGRES\_DB: blog\_automation

ports:

- "5432:5432"

volumes:

- pg\_data:/var/lib/postgresql/data

- ./init.sql:/docker-entrypoint-initdb.d/00\_init.sql

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U blog\_user -d blog\_automation"]

interval: 10s

timeout: 5s

retries: 5

volumes:

pg\_data:

* 파일: init.sql（친구 코드 기반 + 활성 플래그/워드프레스 자격 열 추가）

CREATE TABLE IF NOT EXISTS blogs (

id SERIAL PRIMARY KEY,

blog\_name VARCHAR(255) NOT NULL,

blog\_url VARCHAR(255) NOT NULL,

platform VARCHAR(50) NOT NULL, -- 'wordpress' | 'tistory'

wp\_user VARCHAR(255), -- 워드프레스 계정(앱 비번 발행용 계정)

wp\_app\_password VARCHAR(255), -- 워드프레스 Application Password

category VARCHAR(255),

active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE IF NOT EXISTS articles (

id SERIAL PRIMARY KEY,

blog\_id INTEGER REFERENCES blogs(id),

title VARCHAR(500) NOT NULL,

content TEXT NOT NULL,

html\_content TEXT,

status VARCHAR(50) DEFAULT 'draft', -- 'draft', 'published', 'failed'

wordpress\_post\_id INTEGER,

tistory\_package\_s3 TEXT,

error\_message TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

published\_at TIMESTAMP,

attempted\_at TIMESTAMP

);

CREATE TABLE IF NOT EXISTS execution\_logs (

id SERIAL PRIMARY KEY,

blog\_id INTEGER REFERENCES blogs(id),

step VARCHAR(100) NOT NULL,

status VARCHAR(50) NOT NULL, -- 'success','failed','retry'

message TEXT,

duration\_seconds FLOAT,

tokens\_used INTEGER,

cost DECIMAL(10,4),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE IF NOT EXISTS publishing\_queue (

id SERIAL PRIMARY KEY,

article\_id INTEGER REFERENCES articles(id),

blog\_id INTEGER REFERENCES blogs(id),

priority INTEGER DEFAULT 0,

retry\_count INTEGER DEFAULT 0,

max\_retries INTEGER DEFAULT 3,

status VARCHAR(50) DEFAULT 'pending',

next\_retry\_at TIMESTAMP,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX IF NOT EXISTS idx\_articles\_blog\_id ON articles(blog\_id);

CREATE INDEX IF NOT EXISTS idx\_articles\_status ON articles(status);

CREATE INDEX IF NOT EXISTS idx\_execution\_logs\_blog\_id ON execution\_logs(blog\_id);

CREATE INDEX IF NOT EXISTS idx\_execution\_logs\_created\_at ON execution\_logs(created\_at);

CREATE INDEX IF NOT EXISTS idx\_publishing\_queue\_status ON publishing\_queue(status);

* 파일: .env.example（루트, 로컬 테스트/참고용）

# PostgreSQL (로컬 Docker)

DATABASE\_URL=postgresql://blog\_user:strong\_password\_here@localhost:5432/blog\_automation

# OpenAI / Anthropic

OPENAI\_API\_KEY=sk-xxxx

ANTHROPIC\_API\_KEY=sk-ant-xxxx

# Make.com / Slack (선택)

MAKE\_WEBHOOK\_URL=https://hook.make.com/xxx

SLACK\_WEBHOOK\_URL=https://hooks.slack.com/services/xxx

# AWS 리소스

AWS\_REGION=ap-northeast-2

S3\_BUCKET\_TISTORY=blog-tistory-packages-xxxx # 미리 콘솔에서 버킷 생성

* 파일: lambda/requirements.txt（Lambda 전용 경량 세트）

requests==2.32.3

python-dotenv==1.0.1

psycopg2-binary==2.9.9

langgraph==0.2.33

langchain==0.2.16

langchain-openai==0.1.22

langchain-anthropic==0.1.23

beautifulsoup4==4.12.3

* 파일: lambda/render.py（SEO 규격 HTML/메타 렌더）

# lambda/render.py

# 목적: 본문 규격 템플릿 + 메타 생성

from \_\_future\_\_ import annotations

from typing import List, Dict, Any

import re

def slugify(title: str) -> str:

s = title.lower()

s = re.sub(r"[^a-z0-9\\- ]+", "", s)

s = re.sub(r"\\s+", "-", s).strip("-")

return s or "post"

def render\_html(topic: str, intent: str, outline: List[str], images: int = 4) -> Dict[str, Any]:

style = """

<style>

body{font-family:-apple-system,BlinkMacSystemFont,'Segoe UI',Roboto,'Noto Sans KR',sans-serif;line-height:1.75}

table{width:100%;border-collapse:collapse;margin:1em 0}

th,td{border:1px solid #e5e7eb;padding:10px}

.cta{border:1px dashed #9ca3af;padding:14px;margin:18px 0;background:#f9fafb}

.caption{font-size:.9em;color:#6b7280}

</style>

""".strip()

h = [f"<h1>{topic}</h1>", f"<p>검색 의도: {intent}. 본 문서는 자동화 파이프라인으로 생성되었습니다.</p>"]

for h2 in outline[:6]:

h += [f"<h2>{h2}</h2>", "<p>80–140자 단락 예시. 과장 없이 핵심만 설명합니다.</p>"]

h += [

"<h2>핵심 비교 표</h2>",

"<table><thead><tr><th>항목</th><th>내용</th><th>메모</th></tr></thead><tbody>"

"<tr><td>검색 의도</td><td>정보/상업/거래</td><td>키워드 성격</td></tr>"

"<tr><td>벤치마킹</td><td>상위 문서 요약</td><td>중복 제거</td></tr>"

"<tr><td>본문 규격</td><td>H1=1,H2=3–6,표1,리스트1,FAQ3</td><td></td></tr>"

"</tbody></table>",

"<h2>작성 체크리스트</h2>",

"<ul><li>연관질의 커버</li><li>격차 보완</li><li>이미지 3–5장 alt/캡션/출처</li><li>CTA 2–3개</li><li>메타 타이틀/설명</li></ul>",

"<h2>이미지 삽입 위치</h2>",

]

for i in range(images):

h.append(f"<p>[IMG{i+1}] — 캡션·출처·alt 기입</p>")

h += [

"<h2>FAQ</h2>",

"<details><summary>1) 핵심 개념?</summary><p>간단 설명.</p></details>",

"<details><summary>2) 적용법?</summary><p>단계별 절차.</p></details>",

"<details><summary>3) 흔한 오류?</summary><p>원인/해결.</p></details>",

'<div class="cta" id="CTA\_TOP">[CTA\_TOP] 제휴 링크 (rel="nofollow sponsored")</div>',

'<div class="cta" id="CTA\_MID">[CTA\_MID] 체크리스트 다운로드</div>',

'<div class="cta" id="CTA\_BOTTOM">[CTA\_BOTTOM] 상담/데모</div>',

]

meta = {

"title": f"{topic} 완전 가이드: 핵심 요약과 실전 팁",

"description": "검색의도부터 본문/이미지/메타/CTA까지 한 번에 구성한 실전 가이드.",

"tags": ["SEO","블로그자동화","워드프레스","티스토리","콘텐츠마케팅"],

"slug": slugify(topic),

}

return {"html": style + "\n" + "\n".join(h), "meta": meta}

* 파일: lambda/handler.py（Lambda 진입점: LangGraph 없이도 단일 실행 흐름을 갖춘 최소구현. 이후 LangGraph로 대체 가능）

# lambda/handler.py

# 목적: AWS Lambda에서 실행되는 메인 함수

# - 매일 EventBridge가 호출

# - 1) 주제 생성 2) 초안 작성 3) 최적화 4) SEO 렌더 5) 워드프레스 발행 or 티스토리 패키지(S3) 6) 로그/DB 기록

# - DB는 외부(PostgreSQL on Docker/EC2)로 접속

import os, json, time, logging, base64

from typing import Dict, Any, List

import psycopg2

import requests

from bs4 import BeautifulSoup

from langchain\_openai import ChatOpenAI

from langchain\_anthropic import ChatAnthropic

from lambda.render import render\_html

logging.basicConfig(level=logging.INFO)

log = logging.getLogger("lambda-blog")

# ===== 환경 =====

DATABASE\_URL = os.getenv("DATABASE\_URL") # postgresql://user:pass@host:5432/db

OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

ANTHROPIC\_API\_KEY = os.getenv("ANTHROPIC\_API\_KEY")

S3\_BUCKET\_TISTORY = os.getenv("S3\_BUCKET\_TISTORY")

REGION = os.getenv("AWS\_REGION", "ap-northeast-2")

MAKE\_WEBHOOK\_URL = os.getenv("MAKE\_WEBHOOK\_URL", "")

SLACK\_WEBHOOK\_URL = os.getenv("SLACK\_WEBHOOK\_URL", "")

# ===== LLM 선택 =====

def model(stage: str):

# 비용 최적화: 요약/벤치=Claude, 집필/검수=GPT-4.x mini (여기서는 GPT-5 자리에 gpt-4.1-mini 사용 예)

if stage in {"topic","bench"} and ANTHROPIC\_API\_KEY:

return ChatAnthropic(model="claude-3-5-haiku-latest", anthropic\_api\_key=ANTHROPIC\_API\_KEY, temperature=0.2, max\_tokens=800)

return ChatOpenAI(model="gpt-4.1-mini", api\_key=OPENAI\_API\_KEY, temperature=0.3, max\_tokens=1800)

# ===== 유틸 =====

def db\_conn():

return psycopg2.connect(DATABASE\_URL)

def log\_exec(blog\_id: int, step: str, status: str, message: str = "", cost: float = 0.0):

try:

with db\_conn() as conn:

with conn.cursor() as cur:

cur.execute(

"INSERT INTO execution\_logs (blog\_id, step, status, message, cost) VALUES (%s,%s,%s,%s,%s)",

(blog\_id, step, status, message[:1000], cost),

)

except Exception as e:

log.warning("로그 기록 실패: %s", e)

def fetch\_active\_blogs() -> List[Dict[str,Any]]:

with db\_conn() as conn:

with conn.cursor() as cur:

cur.execute("SELECT id, blog\_name, blog\_url, platform, wp\_user, wp\_app\_password, COALESCE(category,'') FROM blogs WHERE active = TRUE")

rows = cur.fetchall()

keys = ["id","blog\_name","blog\_url","platform","wp\_user","wp\_app\_password","category"]

return [dict(zip(keys,r)) for r in rows]

def wp\_publish(blog, title: str, html: str) -> Dict[str,Any]:

url = blog["blog\_url"].rstrip("/") + "/wp-json/wp/v2/posts"

# Application Password → 기본 인증

auth = (blog["wp\_user"], blog["wp\_app\_password"])

data = {"title": title, "content": html, "status": "publish"}

r = requests.post(url, json=data, auth=auth, timeout=30)

r.raise\_for\_status()

return r.json()

def s3\_upload\_text(key: str, text: str) -> str:

import boto3

s3 = boto3.client("s3", region\_name=REGION)

s3.put\_object(Bucket=S3\_BUCKET\_TISTORY, Key=key, Body=text.encode("utf-8"), ContentType="text/html; charset=utf-8")

return f"s3://{S3\_BUCKET\_TISTORY}/{key}"

def save\_article(blog\_id: int, title: str, content: str, html: str, status: str, wp\_id: int | None, tistory\_s3: str | None):

with db\_conn() as conn:

with conn.cursor() as cur:

cur.execute("""

INSERT INTO articles (blog\_id,title,content,html\_content,status,wordpress\_post\_id,tistory\_package\_s3,published\_at,attempted\_at)

VALUES (%s,%s,%s,%s,%s,%s,%s, CASE WHEN %s='published' THEN NOW() ELSE NULL END, NOW())

""", (blog\_id,title,content,html,status,wp\_id,tistory\_s3,status))

conn.commit()

def notify(text: str):

try:

if SLACK\_WEBHOOK\_URL:

requests.post(SLACK\_WEBHOOK\_URL, json={"text": text}, timeout=10)

if MAKE\_WEBHOOK\_URL:

requests.post(MAKE\_WEBHOOK\_URL, json={"message": text}, timeout=10)

except Exception:

pass

# ===== 메인 핸들러 =====

def handler(event, context):

blogs = fetch\_active\_blogs()

results = []

for blog in blogs:

b\_id = blog["id"]; b\_name = blog["blog\_name"]

try:

# 1) 주제 생성

llm = model("topic")

topic = llm.invoke(f"블로그 '{b\_name}' 주제로 한국어 트렌디 글 주제 1개만 제시. 12~20자.").content.strip()

log\_exec(b\_id, "topic\_generation", "success", topic)

# 2) 초안 작성

writer = model("draft")

draft = writer.invoke(f"""

주제: {topic}

규격: H1=1, H2=3–6, 문단 80–140자, 표1, 리스트1, FAQ3, [IMG1..], CTA 상/중/하.

과장 금지, 정보 제공 고지 포함.

한국어로 마크다운이 아니라 HTML 본문을 생성하되, 표/FAQ/리스트/IMG 플레이스홀더 포함.

""").content

title = topic # 간단화; 필요시 추가 추출

# 3) SEO 렌더(메타/스타일/CTA/IMG 자리 포함)

rendered = render\_html(topic=title, intent="정보", outline=["개요","핵심 단계","사례"], images=4)

html = rendered["html"].replace("<h1>"+title+"</h1>", f"<h1>{title}</h1>\n{draft}")

# 4) 발행/패키징

if blog["platform"] == "wordpress":

wp = wp\_publish(blog, title, html)

save\_article(b\_id, title, draft, html, "published", wp.get("id"), None)

log\_exec(b\_id, "publish\_wordpress", "success", f"post\_id={wp.get('id')}")

results.append({"blog": b\_name, "platform":"wordpress", "post\_id": wp.get("id")})

else:

key = f"{b\_id}/{int(time.time())}\_{title}.html"

s3\_uri = s3\_upload\_text(key, html)

save\_article(b\_id, title, draft, html, "draft", None, s3\_uri)

log\_exec(b\_id, "tistory\_package", "success", s3\_uri)

results.append({"blog": b\_name, "platform":"tistory", "package": s3\_uri})

except Exception as e:

msg = f"[{b\_name}] 실패: {e}"

log.exception(msg)

log\_exec(b\_id, "error", "failed", str(e))

notify(msg)

results.append({"blog": b\_name, "error": str(e)})

notify(f"블로그 자동화 완료: {len(results)}건 처리")

return {"ok": True, "results": results}

* 파일: make/sample\_webhook.json（테스트용 샘플 페이로드）

{ "message": "예시 알림입니다." }

1. 사용 방법 및 주의사항(완전 초보 전용 단계별 안내)  
   아래는 “전체 8단계”입니다. 지금은 1단계부터 진행하세요. 각 단계가 끝나면 “다음 단계”라고 말하면 됩니다.  
   현재스텝 1/8, 전체스텝: [1 DB 올리기]→[2 S3 준비]→[3 .env 작성]→[4 로컬 접속 확인]→[5 Lambda 함수 만들기]→[6 EventBridge 예약]→[7 워드프레스 자격 준비]→[8 Make.com 알림].

① DB 올리기(로컬 또는 EC2)

* 선택 A: 내 PC에서 테스트
  1. Docker Desktop 설치.
  2. 프로젝트 폴더에서 docker compose -f docker-compose.db.yml up -d 실행.
  3. 확인: docker compose -f docker-compose.db.yml exec postgres psql -U blog\_user -d blog\_automation -c "\dt" → 테이블 목록이 보이면 성공.
* 선택 B: AWS EC2(운영용)
  1. t4g.nano 또는 t3.micro 인스턴스 생성(무료 티어).
  2. 보안그룹에 5432 포트(내 IP만 허용) 열기.
  3. 위와 같은 명령으로 docker-compose 실행.
  4. EC2 퍼블릭 IP로 DATABASE\_URL=postgresql://blog\_user:strong\_password\_here@<EC2\_IP>:5432/blog\_automation 구성.

② S3 준비

* 콘솔 → S3 → 버킷 생성(예: blog-tistory-packages-xxxx, 퍼블릭 차단 유지).
* 버킷 이름을 .env의 S3\_BUCKET\_TISTORY에 넣습니다.

③ .env 작성

* 루트에 .env 파일 생성(.env.example 참고).
* DATABASE\_URL은 ①에서 생성한 값, OPENAI\_API\_KEY/ANTHROPIC\_API\_KEY 입력, AWS\_REGION과 S3\_BUCKET\_TISTORY 입력.
* Slack/Make 웹훅이 있으면 추가.

④ 로컬 접속 확인(초간단)

* PC에서 DB가 열려 있는지 확인:
* psql "postgresql://blog\_user:strong\_password\_here@localhost:5432/blog\_automation" -c "select 1;"
* blogs 샘플 1개 추가(워드프레스 예):
* INSERT INTO blogs (blog\_name, blog\_url, platform, wp\_user, wp\_app\_password, category)
* VALUES ('테스트 블로그','https://example.com','wordpress','apiuser','xxxx xxxx xxxx xxxx','news');

티스토리 예:

INSERT INTO blogs (blog\_name, blog\_url, platform, category)

VALUES ('티스토리 테스트','https://myblog.tistory.com','tistory','tech');

⑤ AWS Lambda 함수 만들기

* 콘솔 → Lambda → 함수 생성 → “함수 작성: 코드 업로드( .zip )” 선택, 런타임 Python 3.11.
* 로컬에서 ZIP 패키지 만들기:
  1. 터미널에서
  2. cd lambda
  3. pip install --upgrade pip
  4. pip install -r requirements.txt -t ./package
  5. cp -r render.py handler.py package/
  6. cd package && zip -r ../lambda.zip . && cd ..
  7. 생성된 lambda.zip을 Lambda에 업로드.
* 환경변수 설정(콘솔 > 구성 > 환경 변수):
  1. DATABASE\_URL, OPENAI\_API\_KEY, ANTHROPIC\_API\_KEY, AWS\_REGION, S3\_BUCKET\_TISTORY, MAKE\_WEBHOOK\_URL, SLACK\_WEBHOOK\_URL
* 실행 역할(IAM): S3 PutObject 권한 필요 → 정책 AmazonS3FullAccess(테스트용) 또는 최소권한(해당 버킷 PutObject).
* 테스트 이벤트로 {} 실행 → 결과에 “ok: true”가 보이면 성공.

⑥ EventBridge 예약(매일 자동)

* 콘솔 → EventBridge → 일정 → “일정 생성” → 매일 09:00(한국시간) CRON 설정.
* 대상: Lambda 함수 선택, 입력은 {}.
* 이제 매일 자동 실행.

⑦ 워드프레스 자격 준비

* 워드프레스 관리자 → 사용자 → 프로필 → Application Password 생성 → 사용자명과 앱 비번을 blogs.wp\_user / blogs.wp\_app\_password에 저장.
* HTTPS가 아닌 경우 일부 호스팅에서 REST가 제한될 수 있으니 가능하면 SSL 사용.
* 글이 발행되면 articles.wordpress\_post\_id가 채워지고, 실패 시 articles.error\_message와 execution\_logs에 사유가 남습니다.

⑧ Make.com 알림/승인(선택)

* 트리거: Webhook 모듈 생성 → URL을 .env의 MAKE\_WEBHOOK\_URL로 복사.
* 시나리오: Webhook → Slack(성공/실패 메시지 분기) → 필요 시 Approvals(버튼) → 결과 Slack.
* Lambda는 notify()로 Webhook/Slack을 자동 호출합니다.

운영 팁(성능/안정/비용)

* 성능: 요약/벤치=Claude, 집필/검수=GPT-4.x mini(“GPT-5” 표기는 목표 모델로 가정). 토큰 상한을 단계별로 낮게 두어 과금 제어.
* 안정: 재시도/에러 로깅은 execution\_logs 테이블에 축적. 티스토리는 항상 S3 패키지(반자동)로 안전 운용.
* 비용: 1일 1~2포스트/블로그로 시작, 결과가 좋은 블로그 위주로 확장(스케줄 행 개수로 제어).
* 확장: 100개 블로그는 동시성보다 “큐 기반 순차 처리”가 안정적입니다(EventBridge 5분 간격 × N 라운드).

1. 다음 단계 제안(구체 액션)

* 지금 할 일(1/8): “DB 올리기”를 바로 수행하세요.
  1. Docker Desktop 설치 →
  2. 프로젝트 폴더에서 docker compose -f docker-compose.db.yml up -d →
  3. \dt로 테이블 확인.
* 완료하면 “다음 단계”라고 말해 주세요. 그러면 2/8(S3 준비)로 넘어가서 화면별 클릭 위치까지 안내하겠습니다.

참고

* 워드프레스 발행은 Application Password 기반 Basic Auth가 가장 단순·안정적입니다.
* 티스토리 Open API는 종료되어 자동 게시가 불가하므로, 본 설계는 HTML 패키지를 S3에 저장해 수동 게시를 보조합니다.

1. 요청 내용 확인

* 현재 단계: 1/8 “DB 올리기”.
* 목표: PostgreSQL 데이터베이스를 내 컴퓨터(또는 EC2)에서 도커로 켜고, 테이블을 자동 생성(init.sql 적용)까지 완료. “컴맹” 기준으로 설치·실행·확인·문제해결까지 전부 안내.

1. 접근 방법 설명

* Docker Desktop을 설치한 뒤, 준비해 둔 docker-compose.db.yml과 init.sql을 같은 폴더에 놓고 한 줄 명령으로 PostgreSQL 컨테이너를 띄웁니다. 그 다음 컨테이너 안의 psql로 접속해 테이블이 생겼는지 확인합니다.

1. 완전한 코드  
   아래 두 파일이 “DB 올리기”에 필요한 전부입니다. 이미 받았더라도, 내용이 정확히 일치하는지 확인하고 사용하세요.

* 파일: docker-compose.db.yml

version: '3.8'

services:

postgres:

image: postgres:16-alpine

container\_name: blog\_auto\_db

environment:

POSTGRES\_USER: blog\_user

POSTGRES\_PASSWORD: strong\_password\_here

POSTGRES\_DB: blog\_automation

ports:

- "5432:5432"

volumes:

- pg\_data:/var/lib/postgresql/data

- ./init.sql:/docker-entrypoint-initdb.d/00\_init.sql

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U blog\_user -d blog\_automation"]

interval: 10s

timeout: 5s

retries: 5

volumes:

pg\_data:

* 파일: init.sql

CREATE TABLE IF NOT EXISTS blogs (

id SERIAL PRIMARY KEY,

blog\_name VARCHAR(255) NOT NULL,

blog\_url VARCHAR(255) NOT NULL,

platform VARCHAR(50) NOT NULL,

wp\_user VARCHAR(255),

wp\_app\_password VARCHAR(255),

category VARCHAR(255),

active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE IF NOT EXISTS articles (

id SERIAL PRIMARY KEY,

blog\_id INTEGER REFERENCES blogs(id),

title VARCHAR(500) NOT NULL,

content TEXT NOT NULL,

html\_content TEXT,

status VARCHAR(50) DEFAULT 'draft',

wordpress\_post\_id INTEGER,

tistory\_package\_s3 TEXT,

error\_message TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

published\_at TIMESTAMP,

attempted\_at TIMESTAMP

);

CREATE TABLE IF NOT EXISTS execution\_logs (

id SERIAL PRIMARY KEY,

blog\_id INTEGER REFERENCES blogs(id),

step VARCHAR(100) NOT NULL,

status VARCHAR(50) NOT NULL,

message TEXT,

duration\_seconds FLOAT,

tokens\_used INTEGER,

cost DECIMAL(10,4),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE IF NOT EXISTS publishing\_queue (

id SERIAL PRIMARY KEY,

article\_id INTEGER REFERENCES articles(id),

blog\_id INTEGER REFERENCES blogs(id),

priority INTEGER DEFAULT 0,

retry\_count INTEGER DEFAULT 0,

max\_retries INTEGER DEFAULT 3,

status VARCHAR(50) DEFAULT 'pending',

next\_retry\_at TIMESTAMP,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX IF NOT EXISTS idx\_articles\_blog\_id ON articles(blog\_id);

CREATE INDEX IF NOT EXISTS idx\_articles\_status ON articles(status);

CREATE INDEX IF NOT EXISTS idx\_execution\_logs\_blog\_id ON execution\_logs(blog\_id);

CREATE INDEX IF NOT EXISTS idx\_execution\_logs\_created\_at ON execution\_logs(created\_at);

CREATE INDEX IF NOT EXISTS idx\_publishing\_queue\_status ON publishing\_queue(status);

1. 사용 방법 및 주의사항  
   아래 순서를 그대로 따라 하세요. 운영 환경(EC2)도 거의 동일하지만, 먼저 로컬에서 성공을 보는 것을 권장합니다.

A. 사전 준비

1. 폴더 만들기
   * 데스크톱에 blog-db 폴더를 하나 만듭니다.
   * 위의 docker-compose.db.yml, init.sql 두 파일을 blog-db 안에 저장합니다.
2. 설치: Docker Desktop
   * Windows/Mac 모두 “Docker Desktop”을 설치하고 실행합니다.
   * 설치가 끝나면 Docker Desktop을 켜둔 상태로 다음 단계로 이동합니다.
   * Windows의 경우 WSL2 활성화가 필요할 수 있습니다. 설치 마법사 안내에 따르세요.

B. DB 시작하기(명령 한 줄)

1. 터미널 열기
   * Mac: “터미널” 앱 실행.
   * Windows: “PowerShell” 실행.
2. 폴더로 이동
   * 예시:
     + Mac: cd ~/Desktop/blog-db
     + Windows: cd $HOME\Desktop\blog-db
3. 실행
   * 아래 명령을 그대로 입력합니다:
   * docker compose -f docker-compose.db.yml up -d
   * 의미: -f로 해당 컴포즈 파일을 지정, up -d는 백그라운드로 실행.

C. 컨테이너 상태 확인

1. 목록 보기
2. docker ps
   * blog\_auto\_db라는 컨테이너가 Up 상태이면 정상입니다.
3. 헬스체크 확인
4. docker inspect --format='{{json .State.Health}}' blog\_auto\_db
   * "Status":"healthy"가 보이면 준비 완료입니다. “starting”이면 5~15초 정도 더 기다렸다가 다시 확인하세요.

D. 테이블 생성 확인(가장 쉬운 방법)  
아래 둘 중 편한 방법을 사용하세요.

방법 1) 컨테이너 안의 psql 사용

docker compose -f docker-compose.db.yml exec postgres psql -U blog\_user -d blog\_automation -c "\dt"

* 결과에 blogs, articles, execution\_logs, publishing\_queue가 보이면 성공입니다.

방법 2) 로컬 psql 클라이언트로 접속(선택)

* Mac에서 Homebrew로 brew install libpq 후 psql 사용 가능.
* 접속 테스트:
* psql "postgresql://blog\_user:strong\_password\_here@localhost:5432/blog\_automation" -c "select 1;"

E. 샘플 데이터 1건 넣어보기(워드프레스 블로그)

1. psql 들어가기
2. docker compose -f docker-compose.db.yml exec postgres psql -U blog\_user -d blog\_automation
3. 프롬프트가 뜨면 아래를 붙여넣기(엔터):
4. INSERT INTO blogs (blog\_name, blog\_url, platform, wp\_user, wp\_app\_password, category)
5. VALUES ('테스트 블로그', 'https://example.com', 'wordpress', 'apiuser', 'xxxx xxxx xxxx xxxx', 'news');
6. SELECT id, blog\_name, platform FROM blogs;
7. 숫자 id가 보이면 제대로 저장된 것입니다. 종료하려면 \q 입력.

F. 자주 발생하는 오류와 해결

1. docker compose: command not found
   * 해결: Docker Desktop이 정상 설치/실행 중인지 확인. Windows는 PowerShell을 관리자 권한으로 열어도 좋습니다.
2. 컨테이너가 곧바로 종료됨(STATUS가 Exited)
   * 원인: init.sql 문법 오류, 포트 5432 충돌 등.
   * 조치: 로그 확인
   * docker logs blog\_auto\_db

문법 오류 메시지가 보이면 init.sql의 누락된 세미콜론, 오타를 수정 후 docker compose -f docker-compose.db.yml down -v → 다시 up -d.

1. port is already allocated
   * 원인: 이미 로컬에 PostgreSQL이 떠 있거나 다른 컨테이너가 5432를 사용 중.
   * 조치: 다른 DB를 종료하거나, docker-compose.db.yml의 ports를 "5433:5432"처럼 바꾸고 다시 up -d. 접속 주소도 5433 포트로 바꿔야 합니다.
2. psql: could not connect to server
   * 원인: 아직 Health가 ready가 아님. 10~20초 후 다시 시도.
   * 조치:
   * docker inspect --format='{{json .State.Health}}' blog\_auto\_db

가 healthy가 되면 접속됩니다.

1. Windows에서 WSL2 관련 에러
   * Docker Desktop 설정 > Resources > WSL Integration에서 기본 배포판(예: Ubuntu) 활성화.
   * 재부팅 후 동일 명령 재시도.

G. 보안/백업 팁

* 비밀번호: 지금은 개발용이므로 strong\_password\_here 그대로 써도 되지만, 운영 전엔 꼭 바꾸세요.
* 백업: pg\_data 볼륨은 컨테이너 삭제해도 유지되지만, 주기적으로 덤프(예: pg\_dump)를 만들어 두면 안전합니다.
* 외부 공개 금지: 로컬 테스트는 포트 5432를 외부에 열어둘 필요 없습니다. EC2 사용 시 보안그룹에서 본인 IP만 허용하세요.

1. 다음 단계 제안

* 여기까지 해서 \dt로 테이블이 보이면 1/8 완료입니다.
* 준비되면 “다음 단계”라고 말해 주세요. 2/8 “S3 준비”에서 버킷 생성부터 권한 설정, 업로드 확인까지 화면 클릭 순서로 이어서 안내하겠습니다.