awesome — let’s go full “bootcamp instructor” mode. keep Cursor open. follow these steps exactly and you’ll have one integrated project that hits FastAPI → Docker → DB → Vault → LLM → 12-Factor → Kubernetes.

**0) Project scaffold (you’re on Windows / PowerShell)**

In your Cursor terminal (it already shows your stack-1-fastapi folder):

python -m venv venv

venv\Scripts\activate

pip install --upgrade pip

Create folders/files:

stack-1-fastapi/

├─ app/

│ ├─ \_\_init\_\_.py

│ ├─ main.py

│ ├─ config.py

│ ├─ db.py

│ ├─ models.py

│ ├─ schemas.py

│ └─ routers/

│ ├─ \_\_init\_\_.py

│ └─ tasks.py

├─ requirements.txt

├─ .env.example

├─ .gitignore

├─ Dockerfile

├─ .dockerignore

├─ docker-compose.yml

└─ k8s/

├─ postgres.yaml

├─ app-deployment.yaml

└─ app-service.yaml

**1) FastAPI basics (run locally first)**

**app/main.py**

from fastapi import FastAPI

from .config import Settings

from .routers import tasks

settings = Settings()

app = FastAPI(title=settings.APP\_NAME)

@app.get("/health", tags=["infra"])

def health():

return {"status": "ok"}

# routes

app.include\_router(tasks.router)

**app/config.py**

from pydantic\_settings import BaseSettings

class Settings(BaseSettings):

APP\_NAME: str = "Stack Task API"

ENV: str = "dev"

LOG\_LEVEL: str = "INFO"

# DB defaults (overridden in Docker/K8s)

DB\_HOST: str = "localhost"

DB\_PORT: int = 5432

DB\_NAME: str = "appdb"

DB\_USER: str = "appuser"

DB\_PASSWORD: str = "changeme"

# Vault (used later)

VAULT\_ADDR: str | None = None

VAULT\_TOKEN: str | None = None

VAULT\_DB\_SECRET\_PATH: str = "db-creds"

# LLM (used later)

OPENAI\_API\_KEY: str | None = None

class Config:

env\_file = ".env"

extra = "ignore"

**app/schemas.py**

from pydantic import BaseModel, Field

class TaskCreate(BaseModel):

title: str = Field(..., min\_length=1, max\_length=200)

description: str | None = Field(None, max\_length=500)

class TaskOut(TaskCreate):

id: int

done: bool = False

class Config:

from\_attributes = True

**app/db.py**

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, DeclarativeBase

from .config import Settings

from loguru import logger

import hvac

settings = Settings()

def \_get\_db\_password() -> str:

# Prefer Vault if configured

if settings.VAULT\_ADDR and settings.VAULT\_TOKEN:

try:

client = hvac.Client(url=settings.VAULT\_ADDR, token=settings.VAULT\_TOKEN)

secret = client.secrets.kv.v2.read\_secret\_version(path=settings.VAULT\_DB\_SECRET\_PATH)

return secret["data"]["data"]["POSTGRES\_PASSWORD"]

except Exception as e:

logger.warning(f"Vault fetch failed; falling back to env DB\_PASSWORD. Reason: {e}")

return settings.DB\_PASSWORD

def \_make\_url() -> str:

pwd = \_get\_db\_password()

return f"postgresql+psycopg://{settings.DB\_USER}:{pwd}@{settings.DB\_HOST}:{settings.DB\_PORT}/{settings.DB\_NAME}"

class Base(DeclarativeBase):

pass

engine = create\_engine(\_make\_url(), pool\_pre\_ping=True)

SessionLocal = sessionmaker(bind=engine)

**app/models.py**

from sqlalchemy.orm import Mapped, mapped\_column

from sqlalchemy import String, Boolean, Integer

from .db import Base

class Task(Base):

\_\_tablename\_\_ = "tasks"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

title: Mapped[str] = mapped\_column(String(200))

description: Mapped[str | None] = mapped\_column(String(500), nullable=True)

done: Mapped[bool] = mapped\_column(Boolean, default=False)

**app/routers/tasks.py**

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from ..db import SessionLocal, Base, engine

from ..models import Task as TaskModel

from ..schemas import TaskCreate, TaskOut

router = APIRouter(tags=["tasks"])

# dev convenience: create tables if not exist

Base.metadata.create\_all(bind=engine)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

@router.post("/tasks", response\_model=TaskOut, summary="Create a new task")

def create\_task(payload: TaskCreate, db: Session = Depends(get\_db)):

task = TaskModel(title=payload.title, description=payload.description, done=False)

db.add(task)

db.commit()

db.refresh(task)

return task

@router.get("/tasks", response\_model=List[TaskOut], summary="List all tasks")

def list\_tasks(db: Session = Depends(get\_db)):

return db.query(TaskModel).order\_by(TaskModel.id.desc()).all()

@router.get("/tasks/{task\_id}", response\_model=TaskOut, summary="Get a task by id")

def get\_task(task\_id: int, db: Session = Depends(get\_db)):

task = db.get(TaskModel, task\_id)

if not task:

raise HTTPException(status\_code=404, detail="Task not found")

return task

@router.delete("/tasks/{task\_id}", status\_code=204, summary="Delete a task")

def delete\_task(task\_id: int, db: Session = Depends(get\_db)):

task = db.get(TaskModel, task\_id)

if not task:

raise HTTPException(status\_code=404, detail="Task not found")

db.delete(task)

db.commit()

**requirements.txt**

fastapi==0.115.0

uvicorn==0.30.0

pydantic==2.8.2

pydantic-settings==2.3.4

SQLAlchemy==2.0.32

psycopg[binary]==3.2.1

alembic==1.13.2

python-dotenv==1.0.1

loguru==0.7.2

hvac==2.3.0

httpx==0.27.0

openai==1.40.0

tenacity==8.5.0

**.gitignore**

venv/

\_\_pycache\_\_/

\*.pyc

.env

.DS\_Store

**Run (local, no DB yet — uses memory for table creation but needs Postgres soon)**

pip install -r requirements.txt

uvicorn app.main:app --reload

Open [http://127.0.0.1:8000/docs](http://127.0.0.1:8000/docs?utm_source=chatgpt.com) → try /health (✅), then we’ll move to Docker+DB.

**2) Dockerize the API (12-Factor: run as a stateless process)**

**Dockerfile**

FROM python:3.11-slim

ENV PYTHONUNBUFFERED=1 PYTHONDONTWRITEBYTECODE=1 PIP\_NO\_CACHE\_DIR=1

WORKDIR /app

COPY requirements.txt /app/

RUN pip install -r requirements.txt

COPY app /app/app

# optional in dev only: COPY .env /app/.env

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

**.dockerignore**

venv

\_\_pycache\_\_

.git

.gitignore

.env

Build & run:

docker build -t stack-1-fastapi:dev .

docker run -p 8000:8000 stack-1-fastapi:dev

Hit http://localhost:8000/docs — you should see the same Swagger, now from a container.

**3) Add Postgres (DB runs in its own container)**

**docker-compose.yml**

services:

db:

image: postgres:16

environment:

POSTGRES\_DB: appdb

POSTGRES\_USER: appuser

POSTGRES\_PASSWORD: changeme # dev only; Vault will override for API later

ports:

- "5432:5432"

volumes:

- pgdata:/var/lib/postgresql/data

api:

build: .

environment:

DB\_HOST: db

DB\_PORT: 5432

DB\_NAME: appdb

DB\_USER: appuser

DB\_PASSWORD: changeme # temporary; we’ll switch to Vault in the next step

depends\_on:

- db

ports:

- "8000:8000"

volumes:

pgdata:

Bring it up:

docker compose up --build

Test:

* http://localhost:8000/docs
* POST /tasks → create a few; GET /tasks → you should see them persisted (restart api and they should still be there — DB persistence).

**4) Secrets with Vault (no hardcoded creds)**

Add Vault to docker-compose.yml:

vault:

image: hashicorp/vault:1.15

environment:

VAULT\_DEV\_ROOT\_TOKEN\_ID: root

VAULT\_DEV\_LISTEN\_ADDRESS: "0.0.0.0:8200"

ports:

- "8200:8200"

cap\_add:

- IPC\_LOCK

Switch API env to use Vault (replace API env block):

api:

build: .

environment:

DB\_HOST: db

DB\_PORT: 5432

DB\_NAME: appdb

DB\_USER: appuser

DB\_PASSWORD: "" # ignored if Vault is set

VAULT\_ADDR: http://vault:8200

VAULT\_TOKEN: root

VAULT\_DB\_SECRET\_PATH: db-creds

depends\_on: [db, vault]

ports:

- "8000:8000"

Recreate everything:

docker compose down

docker compose up --build

Seed the secret **once** (new terminal):

curl --header "X-Vault-Token: root" ^

--request POST ^

--data "{\"data\": {\"POSTGRES\_PASSWORD\":\"changeme\"}}" ^

http://127.0.0.1:8200/v1/secret/data/db-creds

Reload API container (or docker compose up --build again).  
Now the API reads DB password from Vault (hvac in app/db.py), satisfying your credential-management requirement.

**5) 12-Factor checkpoints (apply now)**

* **Config via env**: done (config.py loads .env, Docker uses env, K8s will use ConfigMap/Secret)
* **Backing services**: DB & Vault are attached resources (compose services)
* **Logs to stdout**: Uvicorn logs go to stdout by default
* **Stateless process**: API has no local state; DB is persistent volume
* **Dev/prod parity**: same image runs everywhere

Add a sample .env.example:

APP\_NAME=Stack Task API

ENV=dev

DB\_HOST=localhost

DB\_PORT=5432

DB\_NAME=appdb

DB\_USER=appuser

DB\_PASSWORD=changeme

# Optional:

# VAULT\_ADDR=http://127.0.0.1:8200

# VAULT\_TOKEN=root

# VAULT\_DB\_SECRET\_PATH=db-creds

# OPENAI\_API\_KEY=sk-...

**6) (Optional but useful) LLM endpoint**

If you have an OpenAI key, add it to Vault or .env as OPENAI\_API\_KEY.  
Create app/routers/summarize.py:

from fastapi import APIRouter, HTTPException

from pydantic import BaseModel

from ..config import Settings

router = APIRouter(tags=["ai"])

class SummarizeIn(BaseModel):

text: str

max\_words: int = 60

class SummarizeOut(BaseModel):

summary: str

@router.post("/summarize", response\_model=SummarizeOut)

def summarize(payload: SummarizeIn):

key = Settings().OPENAI\_API\_KEY

if not key:

raise HTTPException(status\_code=500, detail="OPENAI\_API\_KEY not configured")

# OpenAI v1 client

from openai import OpenAI

client = OpenAI(api\_key=key)

resp = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "Summarize the user's text succinctly."},

{"role": "user", "content": f"Summarize in under {payload.max\_words} words: {payload.text}"}

]

)

summary = resp.choices[0].message.content

return {"summary": summary}

Include it in app/main.py:

from .routers import tasks

from .routers import summarize

# ...

app.include\_router(tasks.router)

app.include\_router(summarize.router)

Rebuild and test /summarize in Swagger.

MCP note: your app is now cleanly layered—adding an MCP server later is just exposing your “tools” (e.g., create/list tasks, summarize) via the MCP protocol. You can explore modelcontextprotocol servers once you’re comfy here.

**7) Kubernetes (Kind)**

Install Kind (if not already). Then:

kind create cluster --name stack-demo

Load your local image into the Kind cluster:

docker build -t stack-1-fastapi:dev .

kind load docker-image stack-1-fastapi:dev --name stack-demo

**k8s/postgres.yaml**

apiVersion: v1

kind: Secret

metadata:

name: pg-secret

type: Opaque

data:

POSTGRES\_PASSWORD: Y2hhbmdlbWU= # base64("changeme")

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pg-pvc

spec:

accessModes: ["ReadWriteOnce"]

resources:

requests:

storage: 1Gi

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: pg

spec:

replicas: 1

selector: { matchLabels: { app: pg } }

template:

metadata:

labels: { app: pg }

spec:

containers:

- name: postgres

image: postgres:16

env:

- name: POSTGRES\_DB

value: appdb

- name: POSTGRES\_USER

value: appuser

- name: POSTGRES\_PASSWORD

valueFrom: { secretKeyRef: { name: pg-secret, key: POSTGRES\_PASSWORD } }

ports: [{ containerPort: 5432 }]

volumeMounts: [{ name: data, mountPath: /var/lib/postgresql/data }]

volumes:

- name: data

persistentVolumeClaim:

claimName: pg-pvc

---

apiVersion: v1

kind: Service

metadata:

name: pg

spec:

selector: { app: pg }

ports:

- port: 5432

targetPort: 5432

**k8s/app-deployment.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

DB\_HOST: "pg"

DB\_PORT: "5432"

DB\_NAME: "appdb"

DB\_USER: "appuser"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: stack-app

spec:

replicas: 1

selector: { matchLabels: { app: stack-app } }

template:

metadata:

labels: { app: stack-app }

spec:

containers:

- name: api

image: stack-1-fastapi:dev

ports: [{ containerPort: 8000 }]

envFrom:

- configMapRef: { name: app-config }

env:

- name: DB\_PASSWORD

valueFrom: { secretKeyRef: { name: pg-secret, key: POSTGRES\_PASSWORD } }

# (optional) to use Vault on k8s, you'd add Vault Agent/sidecar or CSI here later.

**k8s/app-service.yaml**

apiVersion: v1

kind: Service

metadata:

name: stack-app-svc

spec:

type: NodePort

selector: { app: stack-app }

ports:

- port: 8000

targetPort: 8000

nodePort: 30080

Apply:

kubectl apply -f k8s/postgres.yaml

kubectl apply -f k8s/app-deployment.yaml

kubectl apply -f k8s/app-service.yaml

# wait for pods

kubectl get pods

Open the app:  
<http://localhost:30080/docs> (Kind maps NodePort → your host)

**Troubleshooting quickies**

* **Port already in use**: change the host port (8001:8000) or free it (netstat -ano | findstr :8000 → kill PID).
* **Pip/psycopg build errors**: you’re using psycopg[binary] to avoid compiler hassles.
* **Vault not reachable from API**: confirm VAULT\_ADDR=http://vault:8200 in compose and that the vault service is healthy (docker compose ps).

**What you’ve learned (mapped to your list)**

* Swagger/OpenAPI via FastAPI auto-docs ✔️
* Run with Uvicorn/Hypercorn (we used Uvicorn; Hypercorn works similarly) ✔️
* Docker image & container runtime, not python main.py ✔️
* Credentials via Vault (no hardcoding) ✔️
* DB in a separate container; connectivity via Compose network ✔️
* LLM integration endpoint + key management ✔️
* 12-Factor practices (config/env/logs/stateless) ✔️
* Kubernetes deploy with Kind ✔️

When you’re ready, start with **Step 1 run** (local), then go straight to **compose up**. If anything fails, paste the error and I’ll debug with you like we’re pair-programming.