Awesome—here’s a crisp, **end-to-end build plan** you can follow and show as incremental GitHub work. It covers data for the agent, full-stack build, cloud deploy, and CI/CD. I’ve broken it into **milestones** with small, demo-able steps and acceptance checks.

**Build Plan — Agentic RAG Data ETL/Analytics Assistant**

**0) Repo + Project Scaffolding (Day 1–2)**

**Tech**: Python 3.11, FastAPI, Streamlit, LangGraph/LangChain, SQLAlchemy, Pandas, ChromaDB, Azure OpenAI (Chat Completions), Docker.

**Repo layout**

/app

/api # FastAPI (routes, schemas)

/agent # planner, tools, retriever, validator, reporter

/ui # Streamlit app

/core # db, storage, config, logging

/index # corpus building scripts

/tests # unit/integration/e2e

/infra # IaC (optional), container, deploy scripts

/examples # prompts, few-shots, sample queries

**GitHub pushes**: init-repo, dockerfile+compose, fastapi-hello, streamlit-hello.

**1) Data for the “AI agent” (No model training needed)**

You won’t train the LLM; you’ll **curate the retrieval corpus** and **few-shot exemplars**:

* **Demo DB** (Postgres):
  + Start with **Pagila/Chinook/Northwind** or **TPC-H (SF 0.01)**. Add 2–3 “business tables” (customers, orders, products) and 1–2 lookup tables.
* **Tabular files** (for ETL):
  + A few **CSV/Excel/Parquet** datasets from public sources (Kaggle small sets, NYC taxi sample, retail sales samples).
* **Corpus for RAG** (index in Chroma):
  + Data dictionary & table schemas (auto-extract from Postgres → markdown).
  + Business glossary (you write 30–60 entries).
  + **Few-shot NL↔SQL** examples (10–20, tied to your schema).
  + “How to” snippets for Pandas transforms (filter, groupby, join).
* **Optional**: if you want small “training”, fine-tune **embeddings** decision (but still use Azure text-embedding model; no LLM fine-tune required).

**GitHub pushes**: sample-db-seed, raw-datasets-added, build-index-script, glossary-v1, fewshots-v1.

**2) Milestone A — Minimal Query Assistant (Week 1)**

**Goal**: NL → SQL (read-only) → results table.

**Work**

* Planner (simple classifier) → “query” path.
* Retriever: build Chroma index from schema/glossary/examples.
* Generator: prompt template (schema-aware, guardrails), call **Azure Chat Completions**.
* Executor: SQLAlchemy to Postgres (read-only user), enforce LIMIT.
* Return table + the generated SQL.

**Acceptance**

* 10 canned questions return correct results (exact match on rowcounts/columns).
* No DML/DDL executed; blocked by regex/AST.

**GitHub pushes**: planner-v1, retriever-v1, sql-tool-v1, executor-v1, guardrails-v1, e2e-smoke-tests.

**3) Milestone B — Validator & Repair Loop (Week 2)**

**Goal**: Iterative “agentic” loop.

**Work**

* Catch syntax/runtime errors; summarize trace; retry with short error-aware prompt.
* Schema/rowcount checks (expected columns, non-empty).
* Cap retries (N=2–3), timeouts, row caps.

**Acceptance**

* 90% of failing queries recover within 1–2 retries on your test suite.
* All tool calls logged.

**GitHub pushes**: validator-loop, error-summarizer, retry-policy, tool-logging.

**4) Milestone C — ETL (CSV/Excel/Parquet) (Week 3)**

**Goal**: ETL path with Pandas.

**Work**

* New tool: **pandas\_runner** for extract→transform (filter, groupby, join) → **export**.
* S3 integration:
  + s3://bucket/raw/ for uploads
  + s3://bucket/reports/, s3://bucket/charts/ for outputs
* Planner routes “etl” tasks; generator emits Pandas code (guardrails: safe ops only).

**Acceptance**

* Given a CSV in S3, produce a grouped report CSV to reports/ with correct aggregations.
* Large file (>50MB) handled via chunked reads or skip with message.

**GitHub pushes**: etl-tool-v1, s3-client, exports-v1, etl-tests.

**5) Milestone D — Visualizations & Reporting (Week 4)**

**Goal**: Tables + charts + downloadable CSV/Excel.

**Work**

* Reporter module (matplotlib or Plotly).
* Streamlit UI: show table, chart, and **Download** buttons (presigned S3 URLs).
* Add short NL explanation of result.

**Acceptance**

* For 5 analytics tasks, get chart+CSV and explanation that matches expected logic.

**GitHub pushes**: reporter-v1, ui-results-view, presigned-urls.

**6) Milestone E — UX polish & UAT harness (Week 5)**

**Goal**: Demo-ready app.

**Work**

* UI flows: ask → show plan/SQL → results → save report.
* “Citations”: show which schema/glossary snippets were retrieved.
* Add session logs; simple admin page to view audit logs.

**Acceptance**

* UAT script (10 tasks) executable end-to-end by non-technical user.

**GitHub pushes**: ui-polish, citations-pane, audit-view.

**7) Testing & Quality Gates (ongoing; formalize by Week 5)**

* **Unit**: planner, validator, guardrails, exporters (pytest).
* **Contract**: API responses (pydantic models) with **schemathesis** or pytest.
* **Integration**: containerized Postgres + Chroma + minimal S3 (use moto or LocalStack for CI).
* **E2E**: Playwright/Selenium for Streamlit flows.
* **Datasets**: golden CSVs & SQL outputs for assertions.

**GitHub pushes**: tests-unit, tests-integration, e2e-tests, coverage-ci.

**8) Security & Cost Controls (Week 2 then refine)**

* Read-only DB user; SQL AST/regex check to block DML/DDL.
* Timeouts (LLM+DB); row caps/pagination.
* Secrets: **GitHub Actions secrets** (dev) and **AWS Secrets Manager** (prod).
* Costs: **Chroma (local)** > OpenSearch; **RDS** turned on only when needed; S3 lifecycle policy; small ECS task (Fargate spot optional).
* Logging & metrics: CloudWatch (latency, error rate, retry count, LLM tokens).

**GitHub pushes**: policy-guards, secrets-wiring, metrics-logging.

**9) Cloud Deployment (AWS + Azure OpenAI) (Week 4–5)**

**Minimal, credit-friendly topology**

* **ECS Fargate** service (one task) running your Docker image (FastAPI + Streamlit + Chroma).
* **RDS Postgres** (db.t4g.micro) or start with **SQLite** for demos; scale to RDS for UAT week only.
* **S3** bucket for raw/, reports/, charts/.
* **ALB** (or API Gateway + Fargate) in front.
* **Azure OpenAI**: keep external; store endpoint+key in Secrets Manager.

**IaC (optional)**: Terraform in /infra/terraform (VPC, ECS service, ALB, RDS, S3, IAM, Secrets, CloudWatch).

**GitHub pushes**: docker-image-ci, aws-task-def, deploy-scripts, terraform-init (if using IaC).

**10) CI/CD Pipeline (from Day 1; expand over time)**

**GitHub Actions**

* **CI** (on PR): lint (ruff/black), type-check (mypy), unit+integration tests, coverage gate, build Docker.
* **Security**: bandit, Trivy Docker scan.
* **CD** (on main tag): push image to ECR → update ECS service; run DB migrations (if any); smoke test endpoint.

**Workflow files**: .github/workflows/ci.yml, cd.yml.

**GitHub pushes**: ci-basic, ci-tests, cd-ecs, trivy-scan.

**11) Documentation & Demos (ongoing; finalize Week 5)**

* /docs: architecture (use-case, sequence, **C4 container**), API docs (FastAPI OpenAPI), runbooks (dev vs prod), example prompts, UAT checklist.
* Record short GIFs: NL→SQL, ETL, chart export.

**GitHub pushes**: docs-arch, docs-uat, demo-gifs.

**Stretch (if time permits)**

* **Auth** (Cognito or simple bearer tokens).
* **Dataset catalog** (list available tables & files).
* **Query history** & saved dashboards.

**What to show supervisors (incremental evidence)**

* Every milestone ends with a **demo PR** and **README update**.
* Each PR links to: short Loom/GIF, tests passing badge, and checklists (“acceptance met?”).

**Notes about “training” the AI**

* You **do not** need to train an LLM. Your “training” effort is **corpus construction**: schema docs, glossary, examples, and Pandas/SQL templates that are **retrieved** to guide GPT-4.
* If you later want a learning artifact: you can log (NL, context, SQL/Pandas, outcome) and use it to expand few-shots or fine-tune an *embedding* model selection—not GPT-4 itself.

This plan lines up with the structure and expectations you’ve already written in your interim report (sections 3.1–3.3 and deployment notes).

If you want, I can turn this into a **week-by-week calendar** (from **Sep 22 → Nov end**) with concrete dates/tickets you can paste into GitHub Projects.