Here is your complete **Azure Data Engineering Mastery Roadmap**. This document details 14 end-to-end projects, structured to take you from a novice to a Senior Data Architect.

Each project is designed to solve a specific, high-value business problem using the Azure ecosystem.

PHASE 1: THE FOUNDATION (EASY)

Focus: Understanding the core mechanics of moving and storing data in the cloud.

1. The "Legacy Migration" (Lift & Shift)

- **Use Case:** Automate the migration of on-premise local files to a cloud database.
- **Real World Scenario:** A regional retail chain, "QuickMart," currently saves daily sales reports as CSV files on a local file server. The Analytics team needs this data in a SQL database every morning to run Power BI dashboards. Currently, they copy-paste it manually.
- **Data Source: Synthetic Sales Data**. I will provide a Python script (generate_sales.py) that creates daily CSV files with headers like Product_ID, Quantity, Price, Date.
- **Tech Stack:**
  - **Orchestration:** Azure Data Factory (ADF).
  - **Storage:** Azure Data Lake Gen2 (ADLS) - Bronze Layer.
  - **Compute:** Azure SQL Database.
  - **Language:** SQL (Stored Procedures for Merge/Upsert).
- **Key Learnings:**
  - Setting up Azure Resources (Resource Groups, Linked Services).
  - Understanding the "Copy Activity" in ADF.
  - Difference between "Blob Storage" and "Data Lake Gen2" (Hierarchical Namespace).
- **End Goal:** A pipeline that triggers automatically when a new file lands, copies it to the Lake, and loads it into a SQL Table.

2. The "API Collector" (Serverless Ingestion)

- **Use Case:** Ingest small, frequent data chunks from a REST API into a Data Lake.
- **Real World Scenario:** A logistics company, "GlobalShip," wants to correlate delivery delays with weather conditions. They need to archive weather data for 50 cities every hour.
- **Data Source: OpenWeatherMap API** (Free Tier - 1000 calls/day). You will sign up for a free API Key.
- **Tech Stack:**
  - **Orchestration:** ADF (Web Activity).
  - **Storage:** ADLS Gen2 (Partitioned by Year/Month/Day).
  - **Compute:** Azure Synapse Serverless SQL Pool.
  - **Format:** JSON.

- **Key Learnings:**
  - Handling HTTP Requests (GET/POST) inside ADF.
  - **Partitioning Strategy:** Why organizing files by date is critical for performance.
  - **Schema-on-Read:** Querying raw JSON files using SQL without creating a database table (OPENROWSET).
- **End Goal:** A folder structure in ADLS (weather/2023/10/25/12_00.json) that can be queried instantly via SQL.

3. The "Data Cleaner" (PySpark Transformation)

- **Use Case:** Convert raw, "dirty" data into a clean, query-optimized format using code.
- **Real World Scenario:** The Data Science team at a ride-sharing startup needs clean trip data to train a pricing model. The raw logs contain null passenger counts, negative trip distances, and duplicate records.
- **Data Source: NYC Taxi & Limousine Commission** (Azure Open Datasets or Public Parquet files).
- **Tech Stack:**
  - **Processing:** Azure Databricks (Standard Cluster).
  - **Language:** PySpark (Python).
  - **Storage:** ADLS Gen2 (Silver Layer).
- **Key Learnings:**
  - **PySpark Basics:** Reading/Writing Parquet, DataFrames.
  - **Data Quality:** Filtering nulls, casting string to int, removing duplicates (dropDuplicates).
  - **Mounting:** Connecting Databricks to ADLS securely.
- **End Goal:** A clean Parquet dataset in the "Silver" container where every row has valid data.

PHASE 2: THE BUILDER (MEDIUM)

Focus: Advanced Logic, Data Modeling, and Automation.

4. The "Modern Data Warehouse" (SCD Type 2)

- **Use Case:** Track historical changes in data over time (Slowly Changing Dimensions).
- **Real World Scenario:** A HR department needs to track employee promotions. If "John Doe" moves from "Analyst" to "Manager," we must keep the old record (for historical reporting) and add a new one.
- **Data Source: WHO COVID-19 Data** (CSV) or a custom "Employee" CSV I will help you generate.
- **Tech Stack:**
  - **Compute:** Azure Synapse Dedicated SQL Pool (formerly SQL DW).
  - **Orchestration:** ADF (Data Flows).
  - **Language:** T-SQL (Merge Statements).

- **Key Learnings:**
  - **Dimensional Modeling:** Fact vs. Dimension tables.
  - **SCD Type 2:** Managing Valid_From and Valid_To dates.
  - **Watermarking:** Only processing *new* files since the last run.
- **End Goal:** A Star Schema in Synapse where you can query "What was the infection rate on X date?" accurately.

5. The "Medallion Architect" (Lakehouse Pattern)

- **Use Case:** Implement the industry-standard "Bronze-Silver-Gold" architecture.
- **Real World Scenario:** An e-commerce giant needs a single platform that serves raw data to Data Scientists (Bronze), cleaned data to Analysts (Silver), and aggregated KPIs to Management (Gold).
- **Data Source: Olist E-Commerce Dataset** (Public on Kaggle - Orders, Customers, Products).
- **Tech Stack:**
  - **Platform:** Azure Databricks.
  - **Format:** Delta Lake.
  - **Language:** PySpark & SQL.
- **Key Learnings:**
  - **Delta Lake:** ACID transactions, Time Travel (querying previous versions of data).
  - **Refinement:** Transforming Raw -> Clean -> Aggregated.
  - **Optimization:** OPTIMIZE and Z-ORDER for fast queries.
- **End Goal:** A Databricks workspace where a single notebook runs the full pipeline from raw CSV to a Business-Ready Dashboard table.

6. The "Complex JSON Wrangler" (Hierarchical Data)

- **Use Case:** Flatten and normalize deeply nested, semi-structured data.
- **Real World Scenario:** You are analyzing GitHub activity. The logs are complex JSON: one "Event" contains an array of "Commits," each having an "Author" object. SQL cannot read this easily.
- **Data Source: GH Archive** (Public GitHub Event Data - JSON).
- **Tech Stack:**
  - **Processing:** Databricks or Synapse Spark.
  - **Functions:** explode(), arrays_zip(), from_json().
- **Key Learnings:**
  - **Handling Arrays/Structs:** Breaking down nested lists into rows.
  - **Schema Drift:** What happens when the JSON structure changes tomorrow?
- **End Goal:** Converting a complex JSON file into 3 relational tables: Events, Commits, and Authors linked by IDs.

7. The "Data Quality Guardian" (Automated Testing)

- **Use Case:** Prevent "bad data" from entering the warehouse by implementing a "quarantine" pattern.

- **Real World Scenario:** A Fintech company receives loan applications. If "Loan Amount" is negative or "Interest Rate" is null, the pipeline must **STOP**, alert the team, and reject the record.
- **Data Source: Lending Club Loan Data** (Open). We will manually inject errors (e.g., -5000 loan amount).
- **Tech Stack:**
    - **Orchestration:** ADF.
    - **Validation: Great Expectations** (Python Library) or PySpark Assertions.
    - **Alerting:** Logic Apps (Email).
- **Key Learnings:**
    - **Data Contracts:** Enforcing schema and value constraints.
    - **Error Handling:** The "Dead Letter Queue" pattern.
    - **Observability:** Alerting when data quality drops below 99%.
- **End Goal:** A pipeline that separates "Good Data" from "Bad Data" and sends an email report on failure.

8. The "Fabric Pioneer" (Unified Analytics)

- **Use Case:** Build an end-to-end solution using Microsoft's newest SaaS platform, Fabric.
- **Real World Scenario:** The CIO wants to consolidate tools. Instead of paying for separate ADF, Synapse, and Power BI services, they want to move to Microsoft Fabric.
- **Data Source: Hugging Face Datasets** (Text/NLP data) or similar public dataset.
- **Tech Stack:**
    - **Platform:** Microsoft Fabric.
    - **Storage:** OneLake.
    - **Compute:** Fabric Notebooks / Data Factory Gen2.
- **Key Learnings:**
    - **OneLake:** The "OneDrive for Data."
    - **Direct Lake:** Power BI reading directly from Delta tables (no import mode).
    - **Shortcuts:** Virtualizing data without moving it.
- **End Goal:** A Power BI report powered by data that was never physically moved, only "Referenced" via Fabric Shortcuts.

PHASE 3: THE EXPERT (HARD)

Focus: Scale, Infrastructure, Security, and Optimization.

9. The "Speed Demon" (Real-Time Streaming)

- **Use Case:** Process and analyze data in sub-second latency.
- **Real World Scenario:** Fraud Detection. If a credit card is swiped twice in 5 seconds in different countries, block it immediately. Batch processing (daily) is too slow.
- **Data Source: Python Event Generator** (Simulates a stream of credit card transactions).

- **Tech Stack:**
  - **Ingestion:** Azure Event Hubs.
  - **Processing:** Spark Structured Streaming (Databricks).
  - **Sink:** Delta Lake (Append Mode).
- **Key Learnings:**
  - **Micro-batches:** How Spark Streaming works.
  - **Window Functions:** Aggregating data over time (e.g., "Count transactions in the last 5 mins").
  - **Checkpointing:** Resuming streams after failure.
- **End Goal:** A live dashboard that updates in real-time as you run the Python generator script.

10. The "Metadata-Driven Framework" (The 1-to-N Pipeline)

- **Use Case:** Build **one** pipeline to ingest **50** tables dynamically.
- **Real World Scenario:** You are migrating an SAP system with 2,000 tables. You cannot write 2,000 ADF pipelines. You need a "Master Pipeline" that reads a list of tables and loops through them.
- **Data Source: Azure Open Datasets** (Treating "Holidays", "Weather", and "Census" as 3 different tables).
- **Tech Stack:**
  - **Control Logic:** Azure SQL (Control Table with Table Name, Source Path, Destination).
  - **Orchestration:** ADF (Lookup, ForEach, GetMetadata).
- **Key Learnings:**
  - **Dynamic Parameters:** Passing table names as variables.
  - **Abstraction:** Building frameworks, not just scripts.
- **End Goal:** To add a new dataset, you simply insert a row into SQL—you **never** touch the ADF code.

11. The "Custom Connector" (Azure Functions)

- **Use Case:** Extract data from a source that ADF/Spark cannot handle natively (e.g., Parsing PDFs, Web Scraping).
- **Real World Scenario:** A Legal Tech firm needs to download court rulings (PDFs) from a government website and extract the text for search.
- **Data Source: US Court Opinions** (Public PDFs) or **Common Crawl**.
- **Tech Stack:**
  - **Compute:** Azure Functions (Python Code).
  - **Orchestration:** ADF.
  - **Library:** PyPDF2 or BeautifulSoup.
- **Key Learnings:**
  - **Serverless Compute:** Running arbitrary Python code in the cloud.
  - **Integration:** Calling Functions from ADF.
  - **Unstructured Data:** Handling binary files (PDF/Images).

- **End Goal:** A folder of PDFs is processed, and the text content is saved into a SQL table.

12. The "Fort Knox" (Governance & Security)

- **Use Case:** Secure PII (Personally Identifiable Information) and manage access control.
- **Real World Scenario:** A Hospital system. Doctors can see "Patient Name," but Data Analysts should only see "Patient_ID" (Masked).
- **Data Source: Synthea** (Open Source Patient Generator).
- **Tech Stack:**
    - **Governance:** Unity Catalog (Databricks).
    - **Security:** Azure Key Vault (Secrets Management).
    - **Database:** Azure SQL (Dynamic Data Masking).
- **Key Learnings:**
    - **RBAC:** Role-Based Access Control.
    - **Encryption:** Column-level encryption.
    - **Audit Logs:** Tracking who queried what.
- **End Goal:** Two users query the same table; one sees real names, the other sees *****.

13. The "Performance Architect" (Optimization)

- **Use Case:** Optimize a slow, expensive pipeline to run faster and cheaper.
- **Real World Scenario:** A nightly job takes 4 hours and costs $200. The CFO wants it cut to 1 hour and $50.
- **Data Source: TPC-DS** (Industry Standard Big Data Benchmark - 100GB scale).
- **Tech Stack:**
    - **Platform:** Databricks / Spark UI.
    - **Metrics:** Ganglia / Azure Monitor.
- **Key Learnings:**
    - **Internals:** Shuffle Partitions, Skew, Broadcast Joins.
    - **File Layout:** Partition Pruning, Small File Problem.
    - **Caching:** When (and when not) to cache.
- **End Goal:** Improving a query's execution time by 5x-10x purely through code/config changes.

14. The "Capstone" (Greenfield End-to-End)

- **Use Case:** Build a complete Data Platform from scratch for a fictional startup.
- **Real World Scenario:** You are the Lead Architect for "NeoBank." You must ingest transaction logs (Stream), customer profiles (CSV), and currency rates (API), merge them, and serve a Power BI dashboard.
- **Data Source:** Combination of Project 1, 2, and 9 sources.
- **Tech Stack: Full Stack** (ADF, Databricks, Synapse, Power BI, Key Vault, DevOps).
- **Key Learnings:**
    - **System Integration:** Making all tools work together.
    - **CI/CD:** Deploying via Azure DevOps Pipelines.

    - **IaC:** Deploying resources via ARM/Bicep (Optional but recommended).
- **End Goal:** A production-ready GitHub repository and a live demo you can show in interviews.

Are you ready to begin your journey?

Reply with **"Start Project 1"** to receive the **Step-by-Step Implementation Guide** and **Data Generation Script** for the first project.

PHASE 4: THE ENTERPRISE ARCHITECT (MEDALLION SPECIALIZATION)

Focus: Managing the Software Development Lifecycle (SDLC) for Data.

15. The "Code-First Lakehouse" (Batch CI/CD)

- **Use Case:** Build a standard batch ETL pipeline (Bronze -> Silver -> Gold) where code is version-controlled and deployed automatically to higher environments.
- **Real World Scenario:** A bank's risk team needs a daily report.
    - **Dev Env:** Developers write code using "Mock Data" (100 rows).
    - **UAT Env:** QA team runs the code against "Last Month's Data" (1M rows) to check performance.
    - **Prod Env:** The automated system runs the code against "Live Data."
    - *Problem:* You cannot hardcode paths like `abfss://dev-storage@...`. The code must be dynamic.
- **Data Source: NYC Taxi Trip Data** (Yellow Taxi).
- **Tech Stack:**
    - **Compute:** Azure Databricks (Workflows).
    - **DevOps:** Azure DevOps (Git Repos).
    - **Tooling: Databricks Asset Bundles (DABs)** or **Databricks Repos**.
    - **Orchestration:** ADF (parameterized for environments).
- **Key Learnings:**
    - **Parameterization:** Writing code that accepts `env` as a variable (e.g., reading from `db_bronze_{env}`).
    - **Databricks Asset Bundles (DABs):** The modern way to define "Job as Code" (YAML) and deploy it.
    - **Pull Requests:** Merging code from `feature-branch` to `main` to trigger a deployment.
- **End Goal:** You commit a change to a SQL file in GitHub. An Azure DevOps Pipeline automatically picks it up and updates the Job in the **UAT** workspace.

16. The "Resilient Stream" (Delta Live Tables & DLT)

- **Use Case:** Implement a Streaming Medallion Architecture using **Delta Live Tables (DLT)**, propagated through environments.
- **Real World Scenario:** An e-commerce site tracks user clicks.
  - **Bronze:** Raw JSON clicks (Append Only).
  - **Silver:** Cleaned clicks (Deduplicated).
  - **Gold:** "Clicks per Product" (Aggregated Live).
  - *Challenge:* How do you update the "Silver" logic in Production without stopping the stream and losing data?
- **Data Source: Synthetic Clickstream Generator** (Python script generating UserID, ProductID, Timestamp).
- **Tech Stack:**
  - **Platform:** Azure Databricks (Delta Live Tables).
  - **Language:** Python (DLT decorators `@dlt.table`) or SQL.
  - **Lifecycle:** DLT Pipelines (JSON definition).
- **Key Learnings:**
  - **Declarative ETL:** Defining *what* you want (the table) rather than *how* to compute it.
  - **Expectations:** "Allow", "Drop", or "Fail" on bad data (DLT's built-in quality controls).
  - **Pipeline Promotion:** Promoting a DLT Pipeline JSON configuration from Dev to Prod.
- **End Goal:** A running DLT pipeline where you can see the "Lineage Graph" visually. You will update the code to add a new metric, deploy it, and watch the pipeline restart and backfill automatically.

17. The "Schema Doctor" (Evolution & Backfilling)

- **Use Case:** Handling "Schema Drift" (upstream changes) across environments without breaking Gold reports.
- **Real World Scenario:** You ingest data from an IoT Sensor.
  - **Month 1:** Sensor sends `{temp, humidity}`.
  - **Month 2:** Firmware update! Sensor now sends `{temp, humidity, battery_level}`.
  - *Crash:* Your Production pipeline expects 2 columns but gets 3. It fails.
  - *Fix:* You must update the schema in Dev, test if it breaks the Gold report, and deploy to Prod.
- **Data Source: Azure Open Datasets (Weather)** + Custom script to inject a "New Column" halfway through.
- **Tech Stack:**
  - **Storage:** Delta Lake (Feature: Schema Evolution).
  - **Compute:** Spark / Databricks.

    - **Orchestration:** ADF.
- **Key Learnings:**
    - **Schema Evolution:** Using `.option("mergeSchema", "true")`.
    - **Backfilling:** When a new column is added, how do you fill `nulls` for the historical data?
    - **Blue/Green Deployment:** Running the "New" pipeline alongside the "Old" one in UAT to compare results before switching.
- **End Goal:** A pipeline that survives a structural change in the source data automatically, with the new column appearing in the Gold report.