Data Platform Local Environment Walkthrough & Proficiency Checklist

This document provides a step-by-step learning resource and practical checklist to set up a robust local data platform environment. By following these steps, you will gain hands-on experience with key technologies and be able to demonstrate proficiency for two critical job roles: **Lead Data Engineer** and **AWS Engineer**.

This walkthrough leverages the architectural principles, setup guides, and best practices detailed in the "Building Enterprise-Ready Data Platforms: Core Handbook" and its associated Deep-Dive Addendums.

1. General Setup: Laying the Foundation

These steps are foundational and apply to both job roles.

1.1. Prerequisites Installation

Action: Ensure your local machine has the necessary software installed.

- Install Docker Desktop (or Docker Engine if on Linux).
- Install Git.
- Install Python 3.x with pip.
- Verify docker-compose is installed (usually included with Docker Desktop, or install separately if not).

1.2. Project Repository Setup

Action: Clone the project mono-repo, which contains all necessary code and configuration files.

- Navigate to your desired development directory in your terminal.
- Execute: git clone <your-repo-url>/data-ingestion-platform
- Change into the cloned directory: cd data-ingestion-platform

2. Local Environment Setup: The Progressive Path

This section guides you through building the local data platform incrementally, mirroring the "Progressive Complexity Path" outlined in the Core Handbook. Each track builds upon the previous one.
Reference: For detailed docker-compose.yml configurations and specific instructions, refer to the **Progressive Path Setup Guide Deep-Dive Addendum**.

2.1. Starter Track: Minimal Single-Machine Setup

Purpose: Understand foundational data ingestion and structured storage.
Components: FastAPI (Ingestor), PostgreSQL, MinIO (S3 compatible data lake).
Steps:

1. **Configure docker-compose.yml**:
   - Open the docker-compose.yml file in the project root.
   - **Uncomment** the services for fastapi_ingestor, postgres, and minio.
   - **Comment out** all other services (Kafka, Spark, Airflow, etc.) to keep the setup minimal.
   - Ensure the data/postgres and data/minio directories exist in your project root for persistent volumes (Docker will create them if they don't).
2. **Bring Up Services**:
   - Execute the onboard.sh script (from **Progressive Path Setup Guide Deep-Dive Addendum**) or manually run: docker compose up --build -d
3. **Verify Setup**:
   - Access FastAPI health check: http://localhost:8000/health
   - Access MinIO Console: http://localhost:9001 (login with minioadmin/minioadmin)
   - Connect to PostgreSQL: Use a client (e.g., psql) to connect to localhost:5432 with user user, password password, database main_db.
   - Check Docker logs for all services: docker compose logs -f

2.2. Intermediate Track: Adding Streaming Capabilities

Purpose: Introduce real-time data streams and distributed transformations.
Components (in addition to Starter): Apache Kafka, Apache Spark.
Steps:

1. **Configure docker-compose.yml**:
   - Open docker-compose.yml.
   - **Uncomment** (or keep uncommented) fastapi_ingestor, postgres, minio.
   - **Uncomment** the services for zookeeper, kafka, and spark (and optionally spark-history-server).

   - **Comment out** other Advanced Track services.
   - Review fastapi_ingestor's environment variables to ensure it publishes to Kafka (KAFKA_BROKER: kafka:29092).
   - Verify spark service is configured to connect to Kafka and MinIO.
   - Ensure data/spark-events exists for Spark history server logs.
2. **Bring Up Services**:
   - Run the onboard.sh script again or docker compose up --build -d. The onboard.sh script will initialize Kafka topics.
3. **Generate External Data**:
   - Run the simulate_data.py script (from **Progressive Path Setup Guide Deep-Dive Addendum**). This will continuously send mock financial and insurance data to your FastAPI endpoint.
   - python3 simulate_data.py
4. **Trigger Spark Job**:
   - Manually submit a Spark streaming job from the spark container to consume from Kafka and write to Delta Lake in MinIO.
   - Example: docker exec -it spark spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0,io.delta:delta-core_2.12:2.4.0 --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf spark.hadoop.fs.s3a.endpoint=http://minio:9000 --conf spark.hadoop.fs.s3a.access.key=minioadmin --conf spark.hadoop.fs.s3a.secret.key=minioadmin --conf spark.hadoop.fs.s3a.path.style.access=true pyspark_jobs/streaming_consumer.py raw_financial_transactions kafka:29092 s3a://raw-data-bucket/financial_data_delta (adjust script path and arguments).
5. **Verify Setup**:
   - Check Kafka topic creation: docker exec -it kafka kafka-topics --bootstrap-server localhost:9092 --list
   - Observe data flowing into MinIO (Delta Lake raw zone) using the MinIO console.
   - Check Spark History Server: http://localhost:18080 (if enabled) to see your Spark job status.

2.3. Advanced Track: The Full Production-Ready Stack

Purpose: Integrate orchestration, observability, lineage, and metadata management for a comprehensive environment.
Components (in addition to Intermediate): Apache Airflow, OpenTelemetry & Grafana Alloy, Grafana, Spline, OpenMetadata, MongoDB, cAdvisor.
Steps:

1. **Configure docker-compose.yml**:
   - Open docker-compose.yml.

   - **Uncomment ALL** services including airflow-init, airflow-webserver, airflow-scheduler, airflow-worker, mongodb, openmetadata, grafana, grafana-alloy, cAdvisor, spline.
   - Ensure all environment variables for inter-service communication are correctly set.
   - Ensure necessary data/ subdirectories for persistent volumes exist.
   - Mount airflow_dags and observability directories as volumes for Airflow DAGs and Grafana configurations.

2. **Bring Up Services**:
   - Run the onboard.sh script or docker compose up --build -d. The airflow-init service will set up Airflow's database.
3. **Generate External Data (if not already running)**:
   - python3 simulate_data.py
4. **Verify Setup**:
   - Access Airflow UI: http://localhost:8080 (login admin/admin)
   - Access Grafana UI: http://localhost:3000 (initially anonymous or configure admin user)
   - Access OpenMetadata UI: http://localhost:8585
   - Verify Spline UI: http://localhost:8081
   - Check for container metrics in Grafana dashboards (e.g., CPU, memory of individual services).
   - Ensure Airflow DAGs (e.g., data_ingestion_dag.py, data_transformation_dag.py) appear and run as expected in the Airflow UI, triggering Spark jobs.
   - Observe data flowing through the entire pipeline, including lineage in Spline and metadata in OpenMetadata.

3. Job Role 1: Lead Data Engineer - Proficiency Checklist

This section outlines key areas and activities to demonstrate expertise relevant to a Lead Data Engineer, leveraging the local environment.

Lead Data Engineer Responsibility/Requirement	**Demonstration Activity in Local Environment**	**Relevant Deep-Dive Addendums / Core Handbook Sections**
Lead an engineering team to meet project deadlines and priorities.	Orchestrate complex data pipelines using Airflow (Advanced Track). Define DAGs with dependencies and monitor runs in Airflow UI.	Progressive Path Setup Guide, IaC & CI/CD Recipes (Project Structure)
Ensure the quality, completeness, security, privacy, and integrity of data throughout the data lifecycle.	Implement data quality checks (e.g., Pydantic validation in FastAPI, conceptual Great Expectations in Spark jobs). Understand mergeSchema for Delta Lake. Articulate data encryption concepts.	Testing & Observability Patterns (Data Quality, Security), IaC & CI/CD Recipes (Security Best Practices)
Document critical workflows and operational support aspects of team's responsibilities.	Create and use conceptual runbooks (e.g., for Kafka consumer lag, Delta Lake recovery) and incident review templates. Maintain Airflow DAG documentation.	DR & Runbooks, Testing & Observability Patterns (Alert Mitigation, Incident Review)
Develop understanding of data sources, granularity, availability, and limitations.	Observe data flow from simulate_data.py through FastAPI and Kafka. Examine raw data in MinIO. Discuss how data contracts (schemas) would ensure quality.	Progressive Path Setup Guide (External Data Generator), Core Handbook (Data Sources, Data Lakehouse)
Provide proactive technical oversight and advice to application architecture and development teams promoting re-use, design for scale, stability, and operational efficiency.	Explain architectural choices (ELT vs. ETL, Kafka vs. Kinesis) and their implications (Core Handbook). Demonstrate scaling Spark and Kafka (Intermediate/Advanced Track, Benchmarking).	Core Handbook (Decision Frameworks, Architectural Overview), Testing & Observability Patterns (Benchmarking)
Create maintainable, scalable code to load and manipulate	Develop PySpark batch and streaming jobs (Intermediate	Progressive Path Setup Guide (PySpark jobs), IaC & CI/CD

data in the data warehouse.	Track) that read from Kafka/MinIO and write to curated Delta Lake zones. Emphasize modularity and reusability.	Recipes (Mono-repo Skeleton)
Facilitate communication upward and across project teams and partners.	Present the overall platform architecture (PlantUML diagram). Discuss how OpenMetadata enables data discovery for stakeholders.	Core Handbook (Architectural Overview), Advanced Track (OpenMetadata)
Requirements (Expertise Demonstration)		
Expert level Python programming experience, with an emphasis towards building scalable ETL pipelines.	Develop/modify FastAPI code, PySpark jobs, and Airflow DAGs. Explain error handling and logging.	FastAPI code, PySpark jobs, Airflow DAGs (in project repo)
Demonstrated experience providing customer-oriented solutions or support.	Discuss how observability dashboards (Grafana) and runbooks improve support for data consumers.	Testing & Observability Patterns (Observability, Runbooks)
In-depth knowledge of SQL or NoSQL and experience using a variety of data stores (e.g. RDBMS, analytic database, scalable document stores).	Interact with PostgreSQL (Starter Track), MongoDB (Advanced Track), and query Delta Lake files (via Spark).	Progressive Path Setup Guide (DB interactions)
Employ design patterns and generalize code to address common use cases.	Showcase reusable functions/classes in src/common/utils.py and modular PySpark jobs.	IaC & CI/CD Recipes (Project Structure)
Author high quality, reusable code and contributing to the division's inventory of libraries.	Same as above. Discuss the project's mono-repo structure and CI/CD for quality assurance.	IaC & CI/CD Recipes (Project Structure, CI/CD)
Expertise in big data batch computing tools (e.g. Hadoop or Spark), with demonstrated experience developing distributed data processing solutions.	Run and monitor Spark jobs (Intermediate/Advanced Track). Explain Spark concepts (executors, partitions).	Progressive Path Setup Guide (Spark), Core Handbook (Spark Deep Dive)
Applied knowledge of cloud computing (AWS, GCP, Azure).	Discuss the "Cloud Migration + Terraform Snippets Deep-Dive Addendum" and how local	Cloud Migration + Terraform Snippets

	components map to cloud services.	
Knowledge of open source machine learning toolkits, such as sklearn, Spark ML, or H2O.	While not explicitly implemented, discuss how the curated Delta Lake provides clean data for Spark ML.	Core Handbook (Analytical Layer, Spark Deep Dive)
Solid data understanding in the data rich industries like insurance or financial.	Work with the mock financial and insurance data generated by simulate_data.py.	Progressive Path Setup Guide (External Data Generator)
Applied knowledge of data modeling principles (e.g. dimensional modeling and star schemas).	Discuss the "raw" vs. "curated" Delta Lake zones and how data would be modeled in the curated layer.	Core Handbook (Storage Layer)
Experience with database internals, such as indexes, binary logging, and transactions.	Discuss PostgreSQL settings in docker-compose.yml and the ACID properties of Delta Lake.	Progressive Path Setup Guide (PostgreSQL), Core Handbook (Delta Lake)
Experience using tools for infrastructure-as-code (e.g. Docker, CloudFormation, Terraform).	Demonstrate Docker Compose usage. Discuss the conceptual Terraform snippets in terraform_infra/.	Progressive Path Setup Guide (Docker Compose), IaC & CI/CD Recipes (IaC), Cloud Migration + Terraform Snippets (Terraform Snippets)
Experience with software engineering tools and workflows (i.e. Jenkins, CI/CD, git).	Explain the Conceptual GitHub Actions Release Workflow. Discuss pull requests, linting, unit/integration tests.	IaC & CI/CD Recipes (CI/CD)
Practical experience authoring and consuming web services.	Interact with FastAPI via curl or simulate_data.py. Explain FastAPI endpoints.	Progressive Path Setup Guide (FastAPI), FastAPI app code

4. Job Role 2: AWS Engineer - Proficiency Checklist

This section outlines key areas and activities to demonstrate expertise relevant to an AWS Engineer, with a focus on data pipelines and cloud infrastructure.

AWS Engineer Skill/Responsibility	**Demonstration Activity in Local Environment & Cloud Concepts**	**Relevant Deep-Dive Addendums / Core Handbook Sections**
Key Skills: Cloud Platforms (AWS)		
Mandatory 5 years of hands-on experience: Cloud Platforms (AWS).	Understand the migration paths from local Docker Compose components to AWS managed services. Discuss the conceptual Terraform snippets for AWS resources.	Cloud Migration + Terraform Snippets (Overview of AWS Service Replacements, Step-by-Step AWS Migration Guide, Appendix I)
Key Skills: Data Storage (Data lakes, data warehouses, cloud storage service (S3))		
Data lakes, data warehouses, cloud storage service (S3).	Work with MinIO locally to simulate S3. Discuss how S3 would replace MinIO in the cloud for Delta Lake storage. Understand RDS/DocumentDB as managed databases.	Progressive Path Setup Guide (MinIO), Cloud Migration + Terraform Snippets (Amazon S3, RDS, DocumentDB)
Key Skills: Data Pipelines (Developing and maintaining data pipelines for ETL processes)		
Developing and maintaining data pipelines for ETL processes.	Develop PySpark ETL jobs (Intermediate Track) that read from Kafka/MinIO and write to Delta Lake. Use Airflow for orchestration.	Progressive Path Setup Guide (Spark, Airflow), IaC & CI/CD Recipes (Project Structure)
Key Skills: Programming Languages (Python)		
Python.	Develop/modify FastAPI code, PySpark jobs, and Airflow DAGs.	FastAPI code, PySpark jobs, Airflow DAGs (in project repo)
Key Skills: Data Management Systems (SQL, NoSQL,		

Postgres)		
SQL, NoSQL, Postgres.	Interact with PostgreSQL (Starter Track) and MongoDB (Advanced Track) locally. Explain their roles in the data platform.	Progressive Path Setup Guide (DB interactions)
Key Skills: Data Security and Governance		
Understanding of data security best practices and compliance regulations.	Discuss Data Encryption (in transit/at rest) and Secure Credential Management (local .env, Docker secrets). Explain IAM roles in cloud context.	IaC & CI/CD Recipes (Security Best Practices), Cloud Migration + Terraform Snippets (IAM Setup)
Key Skills: Problem-Solving and Analytical Skills		
Ability to troubleshoot issues and analyze data to identify patterns and trends.	Use Common Gotchas & Debug Playbooks (Kafka stuck consumers, Delta schema drift, Docker networking) to troubleshoot local issues.	Testing & Observability Patterns (Common Gotchas & Debug Playbooks)
Key Responsibilities:		
Designing and implementing scalable and secure data storage solutions in the cloud, ensuring optimal performance and accessibility.	Discuss the terraform_infra/modules for S3, RDS, MSK. Explain VPC, subnets, and security groups from a design perspective.	Cloud Migration + Terraform Snippets (Phase 1, Phase 2, Appendix I)
Developing and maintaining robust data pipelines for the ingestion, transformation, and distribution of large datasets.	Hands-on with PySpark and Airflow. Discuss how Glue/EMR would replace Spark in the cloud.	Progressive Path Setup Guide (Spark, Airflow), Cloud Migration + Terraform Snippets (Phase 4)
Automating data processes and integrating third-party services.	Automate services with Docker Compose. Discuss CI/CD for automated deployments to cloud. Explain Airflow's role in orchestrating openmetadata_ingestion_scripts.	Progressive Path Setup Guide (Docker Compose), IaC & CI/CD Recipes (CI/CD), Advanced Track (OpenMetadata orchestration)
Utilizing cloud services and tools to automate data workflows and streamline the data engineering process.	Discuss AWS services like Lambda, MSK, Glue, MWAA as fully managed replacements.	Cloud Migration + Terraform Snippets (Overview, Phases)
Ensuring compliance with data	Review Data Encryption and	IaC & CI/CD Recipes (Security

governance and security policies, including data encryption and access controls.	Secure Credential Management. Discuss IAM Roles and Policies in AWS.	Best Practices), Cloud Migration + Terraform Snippets (IAM Setup)
Monitoring cloud data systems' performance (Cloud Watch), identifying bottlenecks, and implementing improvements to enhance efficiency.	Use Grafana locally with cAdvisor/Grafana Alloy. Explain how this maps to AWS CloudWatch, ADOT, and Managed Grafana. Interpret Observed Throughput and Latency data.	Testing & Observability Patterns (Observability, Benchmarking), Cloud Migration + Terraform Snippets (Monitoring & Logging)
Conducting data quality checks and implementing measures to ensure data accuracy and integrity.	Implement and understand Data Quality Tests (Pact, Pydantic).	Testing & Observability Patterns (Data Quality Tests)
Optimizing data retrieval and developing APIs for data consumption by various enterprise consumers.	Interact with FastAPI locally. Discuss its replacement by Lambda + API Gateway for scalability.	Progressive Path Setup Guide (FastAPI), Cloud Migration + Terraform Snippets (Lambda + API Gateway)
Providing technical expertise and support for data-related issues, including troubleshooting and resolving data pipeline failures.	Apply Common Gotchas & Debug Playbooks. Discuss how Runbooks improve incident response.	Testing & Observability Patterns (Common Gotchas & Debug Playbooks), DR & Runbooks
Collaborating with IT and security teams to plan and execute disaster recovery strategies for cloud-based data systems.	Understand RPO and RTO. Review DR Runbook Examples. Discuss DR Planning in Cloud.	DR & Runbooks (All Sections), Cloud Migration + Terraform Snippets (DR Planning)
Documenting data engineering processes, creating data flow diagrams, and maintaining metadata for data lineage and cataloging.	Use OpenMetadata and Spline locally (Advanced Track). Explain their role in lineage and cataloging. Reference the PlantUML diagram.	Advanced Track (OpenMetadata, Spline), Core Handbook (Architectural Overview)
Collaborating with architects, analysts, and other engineers to support data modeling, analysis, and reporting needs.	Engage with the multi-disciplinary nature of the data platform. Discuss how curated data supports analytics.	Core Handbook (Purpose, Analytical Layer)
Staying current with emerging cloud technologies and data engineering practices to	The entire progressive path and the cloud migration addendum demonstrate this	All Deep-Dive Addendums and Core Handbook

recommend and adopt innovations that improve data systems.	principle.	

5. Conclusion

By actively engaging with the setup, operation, and troubleshooting of this local data platform environment, and by systematically addressing the points in the proficiency checklists, you will develop a strong, demonstrable understanding of enterprise-ready data platforms from both a Lead Data Engineer and an AWS Engineer perspective. This hands-on experience, coupled with a solid theoretical foundation from the provided documentation, will be invaluable for your career growth.