Enterprise Data Integration, Workflow Orchestration and Pipeline Automation

Duration: 2 Weeks

Week 1: Data Integration Foundations & ETL/ELT

Day 1: Enterprise Data Integration Concepts & Environment Setup

Module 1: Enterprise Data Integration Landscape

- Data integration patterns: ETL, ELT, CDC, event streaming
- Enterprise integration architecture: hub-spoke, point-to-point, service bus
- Data lakes, data warehouses, and lakehouse architectures
- Integration tools landscape: Informatica, Talend, Azure Data Factory, dbt
- Choosing the right integration pattern for a given use case

Lab Exercises:

1. Map an enterprise's data sources and destinations to an integration reference architecture
2. Compare ETL vs ELT for three business scenarios; justify the recommended approach for each

Module 2: Environment Setup & Source System Exploration

- Python, Pandas, psycopg2, SQLAlchemy setup and configuration
- PostgreSQL: database setup, schema exploration, sample data loading
- VS Code, Jupyter Notebook, and Git setup for pipeline development
- Connecting to source systems: databases, APIs, file stores
- Source system profiling: row counts, data types, nulls, distributions

Lab Exercises:

3. Set up the full development environment: Python, PostgreSQL, VS Code, Jupyter, Git
4. Connect to PostgreSQL source database; profile 3 tables and document schema and data quality

Day 2: Data Extraction Strategies

Module 3: Database & File Extraction

- Full extraction vs incremental extraction strategies
- Extracting from PostgreSQL: full table, filtered, and joined extracts
- Reading CSV, Excel, JSON, and Parquet files with Pandas
- Handling encoding issues, delimiters, and header variations
- Staging layer: preserving raw data before transformation

Lab Exercises:

5. Extract data from PostgreSQL tables and CSV/Excel files; load all into a staging schema
6. Write a reusable extract function with logging: source, row count, timestamp, and status

Module 4: API & Multi-Source Extraction

- REST API extraction: endpoints, authentication, pagination, rate limiting
- Python requests library: GET, headers, params, response handling
- Parsing JSON responses and nested structures into DataFrames
- Error handling and retry logic for API extraction
- Multi-source orchestration: combining DB, file, and API extracts

Lab Exercises:

7. Extract data from a paginated REST API; parse JSON and load raw responses to staging

8. Build a multi-source extractor combining DB, CSV, and API data into a unified staging area

Day 3: Data Transformation & Cleansing

Module 5: Data Cleaning & Standardisation

- Missing values: detect, fill, drop strategies
- Duplicate record detection and removal
- Data type conversion and standardisation
- String cleaning: strip, lower, regex, TRIM
- Date and time transformations: parsing, formatting, timezone handling

Lab Exercises:

9. Clean staging data: fix nulls, remove duplicates, standardise formats, convert data types
10. Join data from 3 staging tables (DB + CSV + API) into a single consolidated DataFrame

Module 6: Business Rules & Data Validation

- Applying business rules: conditional logic, derived columns, flags
- Data validation rules: range checks, referential integrity, format checks
- Data quality dimensions: completeness, accuracy, consistency, timeliness
- Reject tables and quarantine pattern for failed records
- Reconciliation: comparing source counts vs transformed output

Lab Exercises:

11. Implement 5 business rules and 5 validation checks; log all failures to a reject table
12. Build a reconciliation report: source counts vs staging vs transformed — flag discrepancies

Day 4: Data Warehouse Design & Loading

Module 7: Dimensional Modelling & Warehouse Design

- Dimensional modelling: facts vs dimensions, grain definition
- Star schema: fact tables, dimension tables, surrogate keys
- Slowly Changing Dimensions: Type 1, Type 2, Type 3
- Date dimension: building a reusable calendar table
- Warehouse schema design for the capstone project dataset

Lab Exercises:

13. Design a star schema for the capstone dataset: fact table + 4 dimensions with keys and relationships
14. Create warehouse schema in PostgreSQL: CREATE TABLE scripts for all fact and dimension tables

Module 8: Data Loading Strategies

- Full load vs incremental load: when to use each
- Upsert pattern: INSERT ... ON CONFLICT in PostgreSQL
- Bulk loading with COPY and SQLAlchemy bulk_insert_mappings
- SCD Type 2 implementation: effective dates and active flags
- Post-load quality checks: row counts, key integrity, null checks

Lab Exercises:

15. Load dimension tables using upsert; implement SCD Type 2 for one slowly changing dimension
16. Load fact table incrementally; run post-load reconciliation and quality checks

Day 5: Modular Pipeline Development & Testing

Module 9: Building a Modular ETL Pipeline

- Pipeline design principles: modularity, reusability, configurability

- Configuration management: config files and environment variables
- Logging best practices: structured logs, log levels, log rotation
- Error handling patterns: try/except, retry logic, dead-letter queues
- Pipeline metadata table: tracking runs, status, row counts, duration

Lab Exercises:

17. Refactor Day 2–4 code into a modular pipeline: extract.py, transform.py, load.py, utils.py
18. Add structured logging and metadata tracking; run pipeline end-to-end and verify audit trail

Module 10: Pipeline Testing & Idempotency

- Testing ETL pipelines: unit, integration, and data tests
- Data testing with Great Expectations or custom assertions
- Idempotency: re-running a pipeline safely without duplicates
- Handling schema changes from source systems
- Pipeline documentation: README, data dictionary, flow diagram

Lab Exercises:

19. Write pytest unit tests for 3 transform functions and 2 validation checks
20. Make pipeline idempotent; test by running it twice and confirming no duplicate records loaded

Week 2: Workflow Orchestration & Pipeline Automation

Day 6: Apache Airflow – Fundamentals

Module 11: Airflow Architecture & DAG Basics

- Airflow architecture: scheduler, webserver, executor, metadata DB
- DAGs: defining workflows as directed acyclic graphs
- Operators: PythonOperator, BashOperator, PostgresOperator
- Task dependencies: set_upstream, set_downstream, >> syntax
- Airflow UI: DAG view, graph view, task logs, and run history

Lab Exercises:

21. Install Airflow with Docker; create a DAG that runs extract → transform → load as 3 tasks
22. Add task dependencies, explore Airflow UI; trigger a manual run and inspect task logs

Module 12: Scheduling, Retries & Alerts

- Scheduling: cron expressions, catchup, and backfill
- Retries, retry delay, and exponential backoff
- SLA misses and timeout configuration
- Email and Slack alerting on task failure or SLA breach
- Task concurrency and parallelism settings

Lab Exercises:

23. Configure daily scheduling with catchup disabled; set retries and email alert on failure
24. Simulate an SLA breach; configure and verify SLA miss alert notification

Day 7: Advanced Airflow Patterns

Module 13: Dynamic DAGs & Cross-DAG Dependencies

- Dynamic DAG generation from config files or database tables
- TaskFlow API: decorators and XCom for task communication
- Cross-DAG dependencies: TriggerDagRunOperator, ExternalTaskSensor
- Branching: BranchPythonOperator for conditional workflow logic
- SubDAGs and TaskGroups for workflow organisation

Lab Exercises:

25. Build a dynamic DAG that generates one task per data source from a YAML config file
26. Implement cross-DAG dependency: DAG B waits for DAG A to complete before running

Module 14: Airflow Executors & Production Deployment

- Executor types: SequentialExecutor, LocalExecutor, CeleryExecutor, KubernetesExecutor
- CeleryExecutor setup: Redis broker, worker scaling
- KubernetesExecutor: pod-per-task execution and resource management
- Airflow variables, connections, and secrets backend
- Airflow in production: monitoring, logging, and maintenance

Lab Exercises:

27. Switch from LocalExecutor to CeleryExecutor; scale to 3 workers and run a parallel DAG
28. Configure Airflow connections and variables via UI and secrets backend; test in a DAG

Day 8: Change Data Capture & Streaming Integration

Module 15: Change Data Capture (CDC)

- CDC concepts: log-based, trigger-based, timestamp-based
- Timestamp-based incremental extraction with watermark columns
- Log-based CDC with Debezium and PostgreSQL WAL
- CDC use cases: real-time sync, event sourcing, audit trails
- Handling deletes and schema changes in CDC pipelines

Lab Exercises:

29. Implement timestamp-based CDC incremental extract for one source table; load delta to warehouse
30. Set up Debezium CDC on PostgreSQL; capture INSERT, UPDATE, DELETE events to a Kafka topic

Module 16: Streaming Integration with Kafka

- Apache Kafka concepts: topics, partitions, producers, consumers
- Kafka Connect for source and sink connectors
- Stream processing basics: filtering, transforming, aggregating
- Exactly-once semantics and delivery guarantees
- Integrating Kafka streams into batch pipeline workflows

Lab Exercises:

31. Produce synthetic transactional data to a Kafka topic; consume and load to PostgreSQL
32. Build a Kafka Connect pipeline from PostgreSQL source to a data lake sink using a connector

Day 9: PySpark, Data Lakes & Pipeline DevOps

Module 17: PySpark for Large-Scale ETL

- Apache Spark architecture: driver, executors, partitions
- PySpark DataFrames: read, transform, write operations
- Parquet and Delta Lake: columnar storage for analytical workloads
- PySpark transformations: filter, join, groupBy, window functions
- Optimising PySpark jobs: partitioning, caching, broadcast joins

Lab Exercises:

33. Use PySpark to read a large CSV, apply transformations, and write output as Parquet
34. Optimise a PySpark job using partitioning and broadcast join; compare execution time

Module 18: Pipeline DevOps & CI/CD

- Infrastructure as Code for pipelines: Docker Compose, Terraform basics
- CI/CD for data pipelines: GitHub Actions workflow for ETL

- Automated testing in CI: run pytest on every pull request
- Containerisation of pipeline components with Docker
- Secrets management and environment-specific configuration

Lab Exercises:

35. Set up a GitHub Actions CI workflow that runs pipeline unit tests on every push
36. Containerise the full ETL pipeline with Docker Compose; run end-to-end in a clean environment

Day 10: Governance, Monitoring & Capstone

Module 19: Pipeline Governance, Monitoring & Security

- Pipeline observability: metrics, logs, and distributed traces
- Data lineage tracking across pipeline stages
- PII and sensitive data handling: masking, encryption, pseudonymisation
- Access control: role-based access in PostgreSQL and Airflow
- Compliance: GDPR data flow mapping for pipelines

Lab Exercises:

37. Add PII masking to the pipeline for email and phone fields; verify masked output in warehouse
38. Build a pipeline monitoring dashboard using the Airflow metadata DB and pipeline run logs

Module 20: Capstone Project – End-to-End Pipeline

- Capstone overview: multi-source pipeline with full orchestration
- Sources: PostgreSQL database + CSV files + REST API
- Pipeline: Extract → Stage → Validate → Transform → Load → Quality Check
- Orchestration: Airflow DAG with scheduling, retries, and alerts
- Presentation: architecture diagram, code walkthrough, data quality results

Lab Exercises:

39. Complete and run the full end-to-end pipeline; verify data in warehouse with SQL queries
40. Present pipeline to cohort: architecture, code, data quality results, and lessons learned

KOENIG
step forward