AN NIIT VENTURE

Capstone Project:

Smart Manufacturing: Fuel Efficiency and Turbine Health Analytics on Azure Cloud

Business Scenario

Modern manufacturing environments—especially in sectors like shipping, energy, and aerospace—are equipped with industrial machines like gas turbines that continuously generate high-frequency telemetry data from embedded sensors. These sensors track vital parameters such as torque, pressure, fuel flow, and decay coefficients, which are critical to monitoring machine health and optimizing fuel efficiency.

However, this raw sensor data is often siloed in edge devices or local storage. There is a growing need to centralize this data in the cloud to build scalable, analytics-ready platforms that support:

- **Remote monitoring of turbine health**
- **Anomaly detection and efficiency benchmarking**
- **Automated data ingestion and transformation pipelines**
- **Real-time API-based access to machine KPIs**
- **Cloud-based storage and deployment for reliability and scale**

This capstone simulates the journey of building a **cloud-native monitoring platform** using **Azure services (ADLS, ADF, Databricks)** and **FastAPI**, integrating software engineering best practices and analytics to generate operational insights.

Project Objectives

1. Ingest and transform turbine sensor data using Python and SQL.
2. Design SRS, HLD, and UML diagrams for a cloud-based turbine monitoring system.
3. Build modular ETL pipelines that clean, transform, and store telemetry data.
4. Develop REST APIs using FastAPI to access turbine health KPIs.
5. Perform exploratory data analysis (EDA) and visualize trends.
6. Deploy data pipelines and dashboards on Azure (ADLS, ADF, Power BI).
7. Build unit tests and deployment readiness documentation.

Project Dataset

Sensor Telemetry Dataset (CSV format)

Field Name	Description
Lp	Lever position
V	Ship speed (knots)
GTT	Gas Turbine shaft torque (kN·m)
GTn	Gas Turbine revolutions per minute (rpm)
GGn	Gas Generator revolutions per minute (rpm)
Ts	Starboard Propeller Torque (kN)

Tp	Port Propeller Torque (kN)
T48	HP Turbine exit temperature (°C)
T1	Compressor inlet air temperature (°C)
T2	Compressor outlet air temperature (°C)
P48	HP Turbine exit pressure (bar)
P1	Compressor inlet pressure (bar)
P2	Compressor outlet pressure (bar)
Pexh	Exhaust gas pressure (bar)
TIC	Turbine Injection Control (%)
mf	Fuel flow rate (kg/s)
decay_coeff_comp	Compressor decay coefficient
decay_coeff_turbine	Turbine decay coefficient

Capstone Phase with learner Tasks & Deliverables

Phase 1: Documentation & Architecture (SDLC, SRS, HLD, UML)

Tasks:

- Draft **Software Requirements Specification (SRS)** for the turbine health platform.
- Define **functional and non-functional requirements**.
- Create **High-Level Design (HLD)** to represent modules (Ingestion, ETL, Monitoring, API).
- Generate **UML diagrams**:
    - **Use Case Diagram** – turbine data ingestion, anomaly detection, API access.
    - **Deployment Diagram** – Azure VMs, ADLS, ADF, FastAPI integration.
    - **Component Diagram** – ingestion, transformation, visualization.

Phase 2: Data Ingestion & Transformation (Python, Pandas, NumPy)

Tasks:

- Write Python code to **ingest CSV sensor logs**.
- Perform **data cleaning** (nulls, outliers, smoothing).
- Compute **derived features** (e.g., temperature-pressure ratios, torque differentials).
- Use **Pandas and NumPy** for initial EDA.

Phase 3: SQL & Exploratory Analysis

Tasks:

- Create SQL tables: `sensor_readings`, `alerts`, `turbine_metadata`.
- Use **SQL for transformations**: joins, filters, aggregations.
- Analyze:
    - Fuel usage patterns over time
    - Turbine efficiency comparisons
- Visualize results using **Matplotlib / Seaborn**.

Phase 4: ETL and Azure Integration (ADF, ADLS)

Tasks:

- Simulate **ETL pipeline** using Python scripts and Azure Data Factory:
    - Extract: Load CSVs from local storage to **Azure Data Lake Gen2**.
    - Transform: Cleaned sensor logs processed into **silver layer**.
    - Load: Store curated data into structured Azure storage (simulated with CSV or SQL).
- Integrate sensor data into **hierarchical ADLS folders**.

Phase 5: REST APIs using FastAPI

Tasks:

- Build **FastAPI-based endpoints**:
    - `GET /health-summary` – aggregate fuel usage & decay status.
    - `GET /sensor-metrics` – recent telemetry for a turbine.
    - `POST /anomaly-alerts` – log new anomaly detection.
- Use **Pydantic** for schema validation.
- Expose Swagger documentation and enable file uploads.

Phase 6: Testing & Release Readiness

Tasks:

- Write **unit tests** using `pytest` for:
  - Data ingestion
  - API endpoints
  - KPI calculations
- Prepare **Test Summary Report**:
  - Pass/Fail table
  - Defect logs
- Define **Acceptance Criteria**:
  - Data load time thresholds
  - API uptime
  - Alert trigger latency