# Hands‑On Lab Guide: Aviation Data Platform Modernization (Azure + Databricks + Delta + Purview)

This lab guide provides an end‑to‑end, step‑by‑step implementation for an Aviation Data Platform on Azure. It covers environment setup, data ingestion (batch & streaming), lakehouse curation (Bronze → Silver → Gold), governance with Purview, orchestration with Azure Data Factory, CI/CD using GitHub Actions, and analytics enablement. All commands are production‑leaning but simplified for hands‑on execution.

## 0) Prerequisites

- Azure subscription with rights to create resource groups and services

- Azure CLI, Databricks workspace access, GitHub account

- Power BI Desktop installed (optional for dashboards)

- Python 3.10+ (for local simulators), Node.js optional

- Service principal or Managed Identities for automation

## 1) Variables (Reused Across Steps)

LOCATION="eastus"  
PREFIX="avi"  
RG\_NAME="${PREFIX}-rg"  
STG\_NAME="${PREFIX}stg$RANDOM" # must be globally unique  
KV\_NAME="${PREFIX}-kv"  
DF\_NAME="${PREFIX}-df"  
EH\_NAMESPACE="${PREFIX}-ehns"  
EH\_NAME="telemetry"  
DBW\_NAME="${PREFIX}-dbw"  
PURVIEW\_NAME="${PREFIX}-purview"  
SQL\_NAME="${PREFIX}-sqlsrv"  
SQL\_DB="${PREFIX}db"  
RAW\_CONT="raw"  
CURATED\_CONT="curated"  
GOLD\_CONT="gold"  
LOG\_CONT="logs"

## 2) Provision Azure Resources (CLI)

# Login and choose subscription  
az login  
# az account set --subscription <SUB\_ID>  
  
# Resource Group  
az group create -n $RG\_NAME -l $LOCATION  
  
# Storage (ADLS Gen2)  
az storage account create -n $STG\_NAME -g $RG\_NAME -l $LOCATION \  
 --sku Standard\_LRS --kind StorageV2 --hierarchical-namespace true  
  
# Containers  
az storage container create --name $RAW\_CONT --account-name $STG\_NAME  
az storage container create --name $CURATED\_CONT --account-name $STG\_NAME  
az storage container create --name $GOLD\_CONT --account-name $STG\_NAME  
az storage container create --name $LOG\_CONT --account-name $STG\_NAME  
  
# Key Vault  
az keyvault create -n $KV\_NAME -g $RG\_NAME -l $LOCATION  
  
# Event Hubs  
az eventhubs namespace create --name $EH\_NAMESPACE -g $RG\_NAME -l $LOCATION --sku Standard  
az eventhubs eventhub create --name $EH\_NAME --namespace-name $EH\_NAMESPACE -g $RG\_NAME \  
 --message-retention 1 --partition-count 4  
  
# Data Factory  
az datafactory create --name $DF\_NAME -g $RG\_NAME -l $LOCATION  
  
# Purview (may require portal if CLI unavailable in tenant)  
az purview account create --name $PURVIEW\_NAME -g $RG\_NAME -l $LOCATION  
  
# Azure SQL (optional for control tables)  
az sql server create -l $LOCATION -g $RG\_NAME -n $SQL\_NAME -u sqladminuser -p "P@ssword123!"  
az sql db create -g $RG\_NAME -s $SQL\_NAME -n $SQL\_DB --service-objective S0

## 3) Identity, Access, and Secrets

Use Managed Identities where possible. Example below shows Service Principal creation and Key Vault storage:

SP=$(az ad sp create-for-rbac -n "${PREFIX}-spn" --role Contributor \  
 --scopes "/subscriptions/$(az account show --query id -o tsv)" -o json)  
APP\_ID=$(echo $SP | jq -r .appId)  
APP\_PWD=$(echo $SP | jq -r .password)  
TENANT\_ID=$(echo $SP | jq -r .tenant)  
  
# Store in Key Vault  
az keyvault secret set --vault-name $KV\_NAME --name sp-app-id --value $APP\_ID  
az keyvault secret set --vault-name $KV\_NAME --name sp-app-secret --value $APP\_PWD  
az keyvault secret set --vault-name $KV\_NAME --name sp-tenant-id --value $TENANT\_ID  
  
# Role assignments (examples)  
STG\_ID=$(az storage account show -n $STG\_NAME -g $RG\_NAME --query id -o tsv)  
EHN\_ID=$(az eventhubs namespace show -n $EH\_NAMESPACE -g $RG\_NAME --query id -o tsv)  
# az role assignment create --assignee <OBJECT\_ID> --role "Storage Blob Data Contributor" --scope $STG\_ID  
# az role assignment create --assignee <OBJECT\_ID> --role "Azure Event Hubs Data Receiver" --scope $EHN\_ID

## 4) Lakehouse Folder Layout

/raw/  
 flight\_operations/  
 aircraft\_telemetry/  
 passenger\_bookings/  
 passenger\_master/  
 crew\_assignments/  
 maintenance\_logs/  
 weather\_data/  
 governance\_logs/  
 data\_quality\_checks/  
/curated/  
 flight/  
 aircraft/  
 crew/  
 passenger/  
 weather/  
/gold/  
 star/  
/logs/

## 5) Control Tables (Azure SQL) – Optional

CREATE TABLE dbo.Control\_Ingestion (  
 control\_id INT IDENTITY(1,1) PRIMARY KEY,  
 dataset\_name NVARCHAR(100) NOT NULL,  
 source\_type NVARCHAR(50) NOT NULL, -- CSV, JSON, API, EVENTHUB  
 source\_location NVARCHAR(400) NOT NULL, -- path or endpoint  
 target\_container NVARCHAR(50) NOT NULL, -- raw  
 target\_path NVARCHAR(400) NOT NULL, -- e.g., raw/flight\_operations  
 file\_pattern NVARCHAR(200) NULL, -- \*.csv  
 delimiter NVARCHAR(10) NULL,  
 header\_rows INT NULL,  
 frequency NVARCHAR(50) NOT NULL, -- daily, hourly, streaming  
 watermark\_col NVARCHAR(100) NULL,  
 last\_success\_ts DATETIME2 NULL,  
 active\_flag BIT NOT NULL DEFAULT(1)  
);  
  
INSERT INTO dbo.Control\_Ingestion(dataset\_name,source\_type,source\_location,target\_container,target\_path,file\_pattern,delimiter,header\_rows,frequency,watermark\_col)  
VALUES  
('flight\_operations','CSV','/landing/flight\_ops/','raw','flight\_operations','\*.csv',',',1,'hourly','scheduled\_departure\_time'),  
('aircraft\_telemetry','EVENTHUB','telemetry','raw','aircraft\_telemetry',NULL,NULL,NULL,'streaming','timestamp\_utc'),  
('passenger\_bookings','CSV','/landing/bookings/','raw','passenger\_bookings','\*.csv',',',1,'daily','booking\_date'),  
('passenger\_master','CSV','/landing/passengers/','raw','passenger\_master','\*.csv',',',1,'weekly',NULL),  
('crew\_assignments','CSV','/landing/crew/','raw','crew\_assignments','\*.csv',',',1,'daily','shift\_start\_time'),  
('maintenance\_logs','CSV','/landing/maintenance/','raw','maintenance\_logs','\*.csv',',',1,'daily','maintenance\_date'),  
('weather\_data','API','https://api.weather.example','raw','weather\_data',NULL,NULL,NULL,'10min','timestamp\_utc');

## 6) Batch Ingestion with ADF (Copy Activity)

Create one pipeline per dataset (or a single metadata-driven pipeline). Example linked service and dataset JSON skeletons:

// Linked Service: AzureBlobStorage (ADLS Gen2 via Managed Identity)  
{  
 "name": "LS\_ADLS",  
 "type": "Microsoft.DataFactory/factories/linkedservices",  
 "properties": {  
 "type": "AzureBlobFS",  
 "typeProperties": {  
 "url": "https://<STG\_NAME>.dfs.core.windows.net"  
 },  
 "connectVia": { "referenceName": "AutoIR", "type": "IntegrationRuntimeReference" }  
 }  
}

// Dataset: DelimitedText for Flight Operations  
{  
 "name": "DS\_FlightOperations\_Raw",  
 "properties": {  
 "linkedServiceName": { "referenceName": "LS\_ADLS", "type": "LinkedServiceReference" },  
 "type": "DelimitedText",  
 "typeProperties": {  
 "location": { "type": "AzureBlobFSLocation", "fileSystem": "raw", "folderPath": "flight\_operations" },  
 "columnDelimiter": ",",  
 "escapeChar": "\\",  
 "firstRowAsHeader": true  
 }  
 }  
}

// Pipeline: Copy from Landing to Raw  
{  
 "name": "PL\_Copy\_FlightOperations",  
 "properties": {  
 "activities": [  
 {  
 "name": "CopyToRaw",  
 "type": "Copy",  
 "typeProperties": {  
 "source": { "type": "DelimitedTextSource" },  
 "sink": { "type": "DelimitedTextSink" }  
 },  
 "inputs": [{ "referenceName": "DS\_FlightOperations\_Landing", "type": "DatasetReference" }],  
 "outputs": [{ "referenceName": "DS\_FlightOperations\_Raw", "type": "DatasetReference" }]  
 }  
 ]  
 }  
}

## 7) Streaming Telemetry (Event Hubs → Databricks)

Producer example (Python):

# pip install azure-eventhub  
import json, time, random  
from datetime import datetime  
from azure.eventhub import EventHubProducerClient, EventData  
  
CONN\_STR = "<EVENT\_HUBS\_CONNECTION\_STRING>"  
EH\_NAME = "telemetry"  
producer = EventHubProducerClient.from\_connection\_string(conn\_str=CONN\_STR, eventhub\_name=EH\_NAME)  
  
aircraft = [f"AC{str(i).zfill(4)}" for i in range(1,201)]  
with producer:  
 for i in range(10000):  
 payload = {  
 "telemetry\_id": f"T{i+1:06d}",  
 "aircraft\_id": random.choice(aircraft),  
 "timestamp\_utc": datetime.utcnow().isoformat(),  
 "altitude\_ft": random.randint(1000,40000),  
 "airspeed\_knots": random.randint(200,600),  
 "engine\_temp\_c": int(random.gauss(500,50)),  
 "fuel\_flow\_kgph": int(random.gauss(2500,300)),  
 "hydraulic\_pressure\_psi": random.randint(2500,3200),  
 "oil\_temp\_c": int(random.gauss(80,5)),  
 "event\_code": random.choices(["Normal","Warning","Alert"],[0.9,0.08,0.02])[0]  
 }  
 batch = producer.create\_batch()  
 batch.add(EventData(json.dumps(payload)))  
 producer.send\_batch(batch)  
 time.sleep(0.01)

Databricks consumer (Structured Streaming to Delta):

from pyspark.sql.functions import col, from\_json, current\_timestamp  
from pyspark.sql.types import \*  
schema = StructType([  
 StructField("telemetry\_id", StringType()),  
 StructField("aircraft\_id", StringType()),  
 StructField("timestamp\_utc", StringType()),  
 StructField("altitude\_ft", IntegerType()),  
 StructField("airspeed\_knots", IntegerType()),  
 StructField("engine\_temp\_c", IntegerType()),  
 StructField("fuel\_flow\_kgph", IntegerType()),  
 StructField("hydraulic\_pressure\_psi", IntegerType()),  
 StructField("oil\_temp\_c", IntegerType()),  
 StructField("event\_code", StringType())  
])  
  
connString = "Endpoint=sb://<namespace>.servicebus.windows.net/;SharedAccessKeyName=<policy>;SharedAccessKey=<key>"  
raw\_path = "abfss://raw@<STG\_NAME>.dfs.core.windows.net/aircraft\_telemetry/"  
chkpt = raw\_path + "\_chkpt/"  
  
eh\_df = (spark.readStream.format("eventhubs")  
 .option("eventhubs.connectionString", connString).load())  
  
json\_df = eh\_df.selectExpr("cast(body as string) as json") \  
 .select(from\_json(col("json"), schema).alias("d")).select("d.\*")  
  
query = (json\_df.withColumn("ingestion\_ts", current\_timestamp())  
 .writeStream.format("delta").outputMode("append")  
 .option("checkpointLocation", chkpt).start(raw\_path))

## 8) Raw → Bronze → Silver Transformations (Databricks)

from pyspark.sql import functions as F  
  
raw = "abfss://raw@<STG\_NAME>.dfs.core.windows.net"  
silver = "abfss://curated@<STG\_NAME>.dfs.core.windows.net"  
  
# Flight Operations  
fl\_bz = (spark.read.format("csv").option("header",True).load(f"{raw}/flight\_operations/\*"))  
fl\_sv = (fl\_bz  
 .withColumn("scheduled\_departure\_time", F.to\_timestamp("scheduled\_departure\_time"))  
 .withColumn("actual\_departure\_time", F.to\_timestamp("actual\_departure\_time"))  
 .withColumn("scheduled\_arrival\_time", F.to\_timestamp("scheduled\_arrival\_time"))  
 .withColumn("actual\_arrival\_time", F.to\_timestamp("actual\_arrival\_time"))  
 .withColumn("delay\_minutes", F.round((F.col("actual\_arrival\_time").cast("long")-F.col("scheduled\_arrival\_time").cast("long"))/60,0))  
 .withColumn("delay\_reason", F.when(F.col("status")=="On-Time","None").otherwise(F.col("delay\_reason")))  
 .dropDuplicates(["flight\_id"]))  
fl\_sv.write.mode("overwrite").format("delta").save(f"{silver}/flight/")  
  
# Passenger Bookings + Master (PII)  
bk\_bz = spark.read.format("csv").option("header",True).load(f"{raw}/passenger\_bookings/\*")  
ps\_bz = spark.read.format("csv").option("header",True).load(f"{raw}/passenger\_master/\*")  
bk\_sv = bk\_bz.withColumn("booking\_date", F.to\_date("booking\_date")).withColumn("ticket\_price\_usd", F.col("ticket\_price\_usd").cast("double"))  
joined = (bk\_sv.alias("b").join(ps\_bz.alias("p"), F.col("b.passenger\_id")==F.col("p.passenger\_id"), "left"))  
joined.write.mode("overwrite").format("delta").save(f"{silver}/passenger/")  
  
# Telemetry  
te\_bz = spark.read.format("delta").load(f"{raw}/aircraft\_telemetry/")  
te\_sv = te\_bz.withColumn("timestamp\_utc", F.to\_timestamp("timestamp\_utc"))  
te\_sv.write.mode("overwrite").format("delta").save(f"{silver}/aircraft/")

## 9) Silver → Gold (Star Schema)

from pyspark.sql import functions as F  
silver = "abfss://curated@<STG\_NAME>.dfs.core.windows.net"  
gold = "abfss://gold@<STG\_NAME>.dfs.core.windows.net"  
  
fl = spark.read.format("delta").load(f"{silver}/flight/")  
ps = spark.read.format("delta").load(f"{silver}/passenger/")  
  
# Dim Date  
dim\_date = (fl.select(F.to\_date("scheduled\_departure\_time").alias("date")).distinct()  
 .withColumn("year", F.year("date")).withColumn("month", F.month("date")).withColumn("day", F.dayofmonth("date")))  
dim\_date.write.mode("overwrite").format("delta").save(f"{gold}/star/dim\_date/")  
  
# Dim Route  
dim\_route = fl.select("route\_id","departure\_airport","arrival\_airport").dropDuplicates()  
dim\_route.write.mode("overwrite").format("delta").save(f"{gold}/star/dim\_route/")  
  
# Fact Flight Performance  
fact\_perf = (fl.withColumn("date", F.to\_date("scheduled\_departure\_time"))  
 .groupBy("route\_id","departure\_airport","arrival\_airport","date")  
 .agg(F.avg("delay\_minutes").alias("avg\_delay\_minutes"),  
 F.sum(F.when(F.col("status")=="On-Time",1).otherwise(0)).alias("on\_time\_flights"),  
 F.count("\*").alias("total\_flights"))  
 .withColumn("on\_time\_arrival\_rate", F.col("on\_time\_flights")/F.col("total\_flights")))  
fact\_perf.write.mode("overwrite").format("delta").save(f"{gold}/star/fact\_flight\_performance/")

## 10) Purview Setup (Catalog, Classification, Lineage)

- Register sources: ADLS Gen2 account, Databricks (via scan), Azure SQL.

- Create scans on containers raw/curated/gold.

- Enable PII classification (Email, Passport Number).

- Add business glossary terms: Flight, Route, Telemetry, PII.

- Validate lineage: ADF → ADLS (raw) → Databricks (silver/gold) → Power BI.

## 11) Data Quality (Great Expectations – optional quick start)

# In Databricks or a Python env:  
# pip install great\_expectations  
  
import great\_expectations as gx  
import pandas as pd  
  
df = spark.read.format("delta").load("abfss://curated@<STG\_NAME>.dfs.core.windows.net/passenger/").toPandas()  
context = gx.get\_context()  
suite = context.add\_or\_update\_expectation\_suite("passenger\_basic\_checks")  
  
# Sample expectations  
from great\_expectations.expectations.core import ExpectColumnValuesToNotBeNull  
suite.add\_expectation(ExpectColumnValuesToNotBeNull(column="passenger\_id"))  
suite.add\_expectation(ExpectColumnValuesToNotBeNull(column="flight\_id"))  
  
# Validate  
r = context.get\_validator(batch=pd.DataFrame(df), expectation\_suite=suite).validate()  
print(r.success)

## 12) CI/CD (GitHub Actions – Databricks Deploy Example)

name: deploy-databricks-jobs  
on:  
 push:  
 branches: [ main ]  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - name: Set up Python  
 uses: actions/setup-python@v5  
 with:  
 python-version: '3.10'  
 - name: Install databricks-cli  
 run: pip install databricks-cli  
 - name: Configure CLI  
 run: |  
 databricks configure --token <<EOF  
 ${{ secrets.DATABRICKS\_HOST }}  
 ${{ secrets.DATABRICKS\_TOKEN }}  
 EOF  
 - name: Deploy Notebooks  
 run: databricks workspace import\_dir ./notebooks /Shared/aviation --overwrite

## 13) Power BI (Consumption)

Connect to Delta tables in Gold via Azure Synapse / SQL Serverless or use the Databricks Connector. Build dashboards:

- Flight Operations: On-time rate, avg delay by route, delay reasons.

- Fleet Health: Alerts count, component issues, downtime hours.

- Passenger Insights: Load factor, cancellations, top routes.

## Appendix A: Dataset Schemas (Raw)

Flight Operations: flight\_id, aircraft\_id, route\_id, departure\_airport, arrival\_airport, scheduled\_departure\_time, actual\_departure\_time, scheduled\_arrival\_time, actual\_arrival\_time, status, delay\_reason

Aircraft Telemetry: telemetry\_id, aircraft\_id, timestamp\_utc, altitude\_ft, airspeed\_knots, engine\_temp\_c, fuel\_flow\_kgph, hydraulic\_pressure\_psi, oil\_temp\_c, event\_code

Passenger Bookings: booking\_id, passenger\_id, flight\_id, seat\_no, booking\_date, fare\_class, ticket\_price\_usd, payment\_method, status

Passenger Master (PII): passenger\_id, first\_name, last\_name, dob, passport\_number, email, phone, frequent\_flyer\_id

Crew Assignments: crew\_id, crew\_name, crew\_role, assigned\_flight\_id, shift\_start\_time, shift\_end\_time, duty\_hours, compliance\_status

Maintenance Logs: maintenance\_id, aircraft\_id, maintenance\_date, component\_checked, issue\_detected, repair\_action, downtime\_hours, next\_due\_check\_date

Weather Data: weather\_id, airport\_code, timestamp\_utc, temperature\_c, visibility\_km, wind\_speed\_knots, precipitation\_mm, weather\_condition

Governance Logs: access\_id, user\_id, dataset\_name, access\_timestamp, access\_type, pii\_mask\_applied, audit\_status

Data Quality Checks: check\_id, dataset\_name, run\_timestamp, row\_count\_expected, row\_count\_actual, missing\_values, validation\_passed, alert\_triggered