**Test Design Document (TDD)**

**Project:** Azure Databricks Workspace Setup & Validation for Sales Data Analytics  
**Prepared By:** [Your Name]  
**Date:** [Date]  
**Version:** 1.0

**1. 🎯 Objective**

To validate the end-to-end deployment, configuration, and data processing capabilities of an Azure Databricks Workspace specifically set up for ingesting, transforming, and analyzing **Sales data** stored in Azure Data Lake Storage.

**2. 🧭 Scope**

This TDD covers:

✅ Workspace provisioning and configuration  
✅ Data Lake Storage access and mount points  
✅ Cluster configuration and policy validation  
✅ Data ingestion from CSV and Parquet sources  
✅ Data transformation using PySpark  
✅ Library installation validation  
✅ Sample analytics output verification  
✅ Security and access controls

**3. 📦 Test Items**

* Azure Databricks Workspace
* Resource Group & Networking
* Azure AD Authentication
* Clusters & Libraries
* Mount Points to ADLS
* PySpark Notebooks
* Secrets from Azure Key Vault
* Sales Data Files (CSV, Parquet)

**4. 🧩 Features to be Tested**

| **Category** | **Features** |
| --- | --- |
| Workspace Provisioning | Deployment via ARM or Terraform; SKU validation |
| Authentication | AAD integration; role assignment |
| Cluster Management | Cluster policies, auto-scaling, auto-termination |
| Library Management | Install PyPI (pandas), Maven libraries |
| Data Mounting | Securely mounting ADLS Gen2 containers |
| Data Ingestion | Loading CSV and Parquet sales data |
| Data Transformation | Aggregating sales totals, cleansing invalid records |
| Data Output | Writing transformed data to Delta Lake tables |
| Security | Secrets access, RBAC enforcement |
| Monitoring | Validating logs and diagnostics in Log Analytics |

**5. 🚫 Features Not to be Tested**

* Azure Cost Management and billing alerts
* Real-time streaming (Kafka) integrations
* Machine Learning model training

**6. 🧪 Test Approach**

Testing will be executed in **phases**:

1. **Provision & Networking Validation**
2. **Authentication & Security Checks**
3. **Cluster Creation & Policy Validation**
4. **Library Installation**
5. **Data Mount Validation**
6. **Data Ingestion and Processing**
7. **Result Verification**
8. **Monitoring Validation**

**7. 🧾 Test Cases**

Here are **detailed test cases with code examples**:

**✅ Test Case 1: Workspace Deployment**

* **Objective:** Validate workspace deployment in target region.
* **Steps:**
  1. Deploy workspace via ARM template.
  2. Verify workspace URL.
  3. Confirm region and SKU.
* **Expected Result:** Workspace is accessible and correct SKU is applied.

**✅ Test Case 2: VNet Injection**

* **Objective:** Validate networking configuration.
* **Steps:**
  1. Check subnet association.
  2. Confirm NSG rules (e.g., allow outbound port 443).
* **Expected Result:** Workspace is correctly injected into VNet.

**✅ Test Case 3: Azure AD Authentication**

* **Objective:** Confirm users can authenticate.
* **Steps:**
  1. Assign test user to workspace.
  2. Login and validate permissions.
* **Expected Result:** User can access workspace.

**✅ Test Case 4: Cluster Creation with Policies**

* **Objective:** Validate cluster configurations.
* **Steps:**
  1. Create cluster with policy.
  2. Attempt policy override (e.g., node type).
* **Expected Result:** Policies enforced.

**Sample Cluster Policy JSON:**

json

CopyEdit

{

"cluster\_type": "all-purpose",

"autoscale": {

"min\_workers": 2,

"max\_workers": 5

},

"spark\_version": "12.2.x-scala2.12",

"node\_type\_id": "Standard\_DS3\_v2",

"autotermination\_minutes": 30

}

**✅ Test Case 5: Library Installation**

* **Objective:** Validate libraries can be installed.
* **Steps:**
  1. Install PyPI package pandas.
  2. Install Maven package com.databricks:spark-xml\_2.12:0.14.0.
* **Sample Code:**

python

CopyEdit

# Install PyPI library via notebook

%pip install pandas

# Validate installation

import pandas as pd

df = pd.DataFrame({'col1':[1,2]})

display(df)

* **Expected Result:** Libraries installed and functional.

**✅ Test Case 6: Mount ADLS Gen2**

* **Objective:** Verify secure mounting of Data Lake.
* **Steps:**
  1. Use Key Vault secret to mount.
  2. List files.
* **Sample Code:**

python

CopyEdit

configs = {

"fs.azure.account.auth.type": "OAuth",

"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"fs.azure.account.oauth2.client.id": dbutils.secrets.get(scope="sales-kv", key="client-id"),

"fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="sales-kv", key="client-secret"),

"fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<tenant-id>/oauth2/token"

}

dbutils.fs.mount(

source = "abfss://salesdata@<storage-account-name>.dfs.core.windows.net/",

mount\_point = "/mnt/sales",

extra\_configs = configs)

* **Expected Result:** Mount point /mnt/sales accessible.

**✅ Test Case 7: Data Ingestion and Transformation**

* **Objective:** Validate reading and processing Sales CSV.
* **Steps:**
  1. Load raw CSV.
  2. Clean and aggregate data.
* **Sample Code:**

python

CopyEdit

# Read CSV

df = spark.read.csv("/mnt/sales/raw/sales\_data.csv", header=True, inferSchema=True)

# Clean: remove nulls

df\_clean = df.dropna(subset=["OrderID", "SalesAmount"])

# Aggregate

df\_agg = df\_clean.groupBy("Region").sum("SalesAmount")

# Display

display(df\_agg)

* **Expected Result:** Aggregation succeeds with correct totals.

**✅ Test Case 8: Write to Delta Lake**

* **Objective:** Validate writing output as Delta Table.
* **Steps:**
  1. Write transformed data.
* **Sample Code:**

python

CopyEdit

df\_agg.write.format("delta").mode("overwrite").save("/mnt/sales/processed/sales\_summary")

* **Expected Result:** Delta files created.

**✅ Test Case 9: Secret Access**

* **Objective:** Confirm secrets are securely retrieved.
* **Steps:**
  1. Access test secret.
* **Sample Code:**

python

CopyEdit

secret = dbutils.secrets.get(scope="sales-kv", key="client-secret")

print("Secret length:", len(secret))

* **Expected Result:** Secret retrieved, no errors.

**✅ Test Case 10: Log Diagnostics**

* **Objective:** Validate logs are emitted to Log Analytics.
* **Steps:**
  1. Run notebook.
  2. Verify logs in Log Analytics workspace.
* **Expected Result:** Job logs appear.

**8. 🧪 Test Data**

| **Data Element** | **Description** |
| --- | --- |
| Raw Sales CSV | /mnt/sales/raw/sales\_data.csv |
| Sample Parquet File | /mnt/sales/raw/sales\_data.parquet |
| Key Vault Secrets | client-id, client-secret |
| Test Users | AAD Users with Contributor roles |
| Libraries | pandas, spark-xml |

**9. ✅ Entry Criteria**

* Azure subscription configured.
* Service principals provisioned.
* ADLS Gen2 storage account created.
* Key Vault with secrets prepared.

**10. 🏁 Exit Criteria**

* All critical test cases passed.
* No high-severity defects open.
* Workspace declared production-ready.

**11. 🗂️ Test Deliverables**

* Completed Test Case Results (Excel / Azure DevOps)
* Defect Reports
* Test Summary Report

**12. 🛠️ Environment**

| **Component** | **Details** |
| --- | --- |
| Azure Region | East US 2 |
| Databricks Runtime | 12.2 LTS |
| Node Type | Standard\_DS3\_v2 |
| Cluster Autoscaling | 2–5 Workers |
| Notebook Language | PySpark |

✅ **Done!** This TDD is ready for **Sales Data Projects on Azure Databricks**.

Next Version

**📝 Test Design Document (TDD)**

**Project:** Azure Databricks Workspace Setup & Sales Data Analytics  
**Prepared By:** [Your Name]  
**Date:** [Date]  
**Version:** 2.0

**1. 🎯 Objective**

Validate and automate the deployment, configuration, and operation of an Azure Databricks workspace for Sales Data ingestion and analytics using:

* **Terraform** for infrastructure provisioning
* **Azure DevOps Pipelines** for CI/CD
* **PySpark Notebooks** for transformations
* **Delta Lake** for output storage

**2. 🧭 Scope**

This version of the TDD includes:  
✅ Workspace provisioning with Terraform  
✅ Networking, Key Vault, and Storage configuration  
✅ Databricks cluster and policies via Terraform  
✅ Azure DevOps pipelines to automate deployment and validation  
✅ Data ingestion and processing workflows  
✅ Monitoring and logging validation

**3. 🧩 Test Items**

* Terraform configurations (\*.tf files)
* Azure DevOps Pipelines (YAML)
* Databricks Workspace and Clusters
* Key Vault secrets
* ADLS Gen2 containers
* PySpark Notebooks
* Libraries and dependencies

**4. 🧷 Terraform Infrastructure Details**

Below are **sample Terraform files** to provision key resources.

**📄 Terraform main.tf**

hcl

CopyEdit

provider "azurerm" {

features = {}

}

resource "azurerm\_resource\_group" "rg" {

name = "rg-sales-analytics"

location = "East US 2"

}

resource "azurerm\_databricks\_workspace" "dbw" {

name = "dbw-sales-analytics"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

sku = "premium"

}

resource "azurerm\_storage\_account" "storage" {

name = "salesdatalake${random\_string.suffix.result}"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

account\_tier = "Standard"

account\_replication\_type = "LRS"

is\_hns\_enabled = true

}

resource "random\_string" "suffix" {

length = 6

special = false

upper = false

}

resource "azurerm\_key\_vault" "kv" {

name = "kv-sales-${random\_string.suffix.result}"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

sku\_name = "standard"

}

resource "azurerm\_key\_vault\_secret" "client\_secret" {

name = "client-secret"

value = "PLACEHOLDER\_SECRET"

key\_vault\_id = azurerm\_key\_vault.kv.id

}

**📄 Terraform outputs.tf**

hcl

CopyEdit

output "databricks\_url" {

value = azurerm\_databricks\_workspace.dbw.workspace\_url

}

**💡 Tip:** Store secrets securely via environment variables or Azure Key Vault references.

**5. 🛠️ Azure DevOps Pipeline (YAML)**

Below is a **sample YAML** for automating Terraform deployment and Databricks jobs:

**📄 azure-pipelines.yml**

yaml

CopyEdit

trigger:

branches:

include:

- main

variables:

- name: environment

value: dev

stages:

- stage: Terraform\_Deploy

displayName: "Terraform Provisioning"

jobs:

- job: terraform

pool:

vmImage: 'ubuntu-latest'

steps:

- checkout: self

- task: TerraformInstaller@0

inputs:

terraformVersion: '1.5.0'

- script: |

terraform init

terraform plan -out=tfplan

displayName: "Terraform Init and Plan"

- script: terraform apply -auto-approve tfplan

displayName: "Terraform Apply"

- stage: Databricks\_Setup

displayName: "Databricks Cluster and Job"

dependsOn: Terraform\_Deploy

jobs:

- job: databricks

pool:

vmImage: 'ubuntu-latest'

steps:

- checkout: self

- task: UsePythonVersion@0

inputs:

versionSpec: '3.x'

- script: pip install databricks-cli

displayName: "Install Databricks CLI"

- script: |

databricks configure --token <<EOF

https://<your-databricks-instance>

$(DATABRICKS\_TOKEN)

EOF

displayName: "Configure Databricks CLI"

- script: |

databricks clusters create --json-file clusters/cluster\_config.json

displayName: "Create Cluster"

- script: |

databricks libraries install --cluster-id $(ClusterId) --pypi-package pandas

displayName: "Install Libraries"

**📄 Sample Cluster Config (clusters/cluster\_config.json)**

json

CopyEdit

{

"cluster\_name": "sales-cluster",

"spark\_version": "12.2.x-scala2.12",

"node\_type\_id": "Standard\_DS3\_v2",

"autoscale": {

"min\_workers": 2,

"max\_workers": 5

},

"autotermination\_minutes": 30

}

**Note:**

* Store DATABRICKS\_TOKEN securely in Azure DevOps library.
* You can add *notebook runs* similarly using databricks runs submit.

**6. 🧾 Test Cases (Extended)**

**All test cases from earlier are included**, with **additional ones** for Terraform and CI/CD:

**✅ Test Case 11: Terraform Plan Validation**

* **Objective:** Ensure Terraform plan shows no destructive changes.
* **Steps:**
  1. Run terraform plan.
  2. Review output.
* **Expected Result:** No unexpected resource deletions.

**✅ Test Case 12: Azure DevOps Pipeline Execution**

* **Objective:** Validate pipeline runs end-to-end.
* **Steps:**
  1. Trigger pipeline from main branch.
  2. Confirm successful stages.
* **Expected Result:** All stages succeed without error.

**✅ Test Case 13: Cluster Job Run**

* **Objective:** Ensure Databricks job can run automatically.
* **Steps:**
  1. Submit sample job.
  2. Validate logs.
* **Expected Result:** Job completes with SUCCESS state.

**7. 🧪 Test Data**

| **Data Element** | **Description** |
| --- | --- |
| CSV Path | /mnt/sales/raw/sales\_data.csv |
| Parquet Path | /mnt/sales/raw/sales\_data.parquet |
| Databricks Token | Azure DevOps secret |
| Cluster Config | clusters/cluster\_config.json |
| Libraries | pandas, spark-xml |

**8. ✅ Entry Criteria**

* Terraform files reviewed and approved.
* Azure DevOps pipeline configured.
* Secrets stored securely in DevOps Library.

**9. 🏁 Exit Criteria**

* Terraform successfully provisions resources.
* Databricks clusters created and functional.
* Sales data ingested and transformed.
* No critical defects open.

**10. 🗂️ Test Deliverables**

* Terraform execution logs.
* Azure DevOps pipeline run history.
* Databricks job run reports.
* Final Test Summary Report.

**11. 🛠️ Environment**

| **Component** | **Details** |
| --- | --- |
| Azure Region | East US 2 |
| Terraform Version | 1.5.0 |
| Databricks Runtime | 12.2 LTS |
| Cluster Node Type | Standard\_DS3\_v2 |
| Azure DevOps Agent | Ubuntu-latest |

**12. 📈 Sample PySpark Notebook Code**

This example aggregates sales data:

python

CopyEdit

# Read Sales CSV

df = spark.read.csv("/mnt/sales/raw/sales\_data.csv", header=True, inferSchema=True)

# Clean Data

df\_clean = df.dropna(subset=["OrderID", "SalesAmount"])

# Convert Amount to Decimal

from pyspark.sql.functions import col

df\_clean = df\_clean.withColumn("SalesAmount", col("SalesAmount").cast("decimal(18,2)"))

# Aggregate Sales by Region

df\_agg = df\_clean.groupBy("Region").sum("SalesAmount")

# Write to Delta

df\_agg.write.format("delta").mode("overwrite").save("/mnt/sales/processed/sales\_summary")

✅ **Done!**

This **TDD with Terraform and Azure DevOps integration** is now **enterprise-ready**.

**📝 Test Design Document (TDD)**

**Project:** Azure Databricks Workspace Setup & Sales Data Analytics  
**Prepared By:** [Your Name]  
**Date:** [Date]  
**Version:** 2.0

**1. 🎯 Objective**

Validate and automate the deployment, configuration, and operation of an Azure Databricks workspace for Sales Data ingestion and analytics using:

* **Terraform** for infrastructure provisioning
* **Azure DevOps Pipelines** for CI/CD
* **PySpark Notebooks** for transformations
* **Delta Lake** for output storage

**2. 🧭 Scope**

This version of the TDD includes:  
✅ Workspace provisioning with Terraform  
✅ Networking, Key Vault, and Storage configuration  
✅ Databricks cluster and policies via Terraform  
✅ Azure DevOps pipelines to automate deployment and validation  
✅ Data ingestion and processing workflows  
✅ Monitoring and logging validation

**3. 🧩 Test Items**

* Terraform configurations (\*.tf files)
* Azure DevOps Pipelines (YAML)
* Databricks Workspace and Clusters
* Key Vault secrets
* ADLS Gen2 containers
* PySpark Notebooks
* Libraries and dependencies

**4. 🧷 Terraform Infrastructure Details**

Below are **sample Terraform files** to provision key resources.

**📄 Terraform main.tf**

hcl

CopyEdit

provider "azurerm" {

features = {}

}

resource "azurerm\_resource\_group" "rg" {

name = "rg-sales-analytics"

location = "East US 2"

}

resource "azurerm\_databricks\_workspace" "dbw" {

name = "dbw-sales-analytics"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

sku = "premium"

}

resource "azurerm\_storage\_account" "storage" {

name = "salesdatalake${random\_string.suffix.result}"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

account\_tier = "Standard"

account\_replication\_type = "LRS"

is\_hns\_enabled = true

}

resource "random\_string" "suffix" {

length = 6

special = false

upper = false

}

resource "azurerm\_key\_vault" "kv" {

name = "kv-sales-${random\_string.suffix.result}"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

sku\_name = "standard"

}

resource "azurerm\_key\_vault\_secret" "client\_secret" {

name = "client-secret"

value = "PLACEHOLDER\_SECRET"

key\_vault\_id = azurerm\_key\_vault.kv.id

}

**📄 Terraform outputs.tf**

hcl

CopyEdit

output "databricks\_url" {

value = azurerm\_databricks\_workspace.dbw.workspace\_url

}

**💡 Tip:** Store secrets securely via environment variables or Azure Key Vault references.

**5. 🛠️ Azure DevOps Pipeline (YAML)**

Below is a **sample YAML** for automating Terraform deployment and Databricks jobs:

**📄 azure-pipelines.yml**

yaml

CopyEdit

trigger:

branches:

include:

- main

variables:

- name: environment

value: dev

stages:

- stage: Terraform\_Deploy

displayName: "Terraform Provisioning"

jobs:

- job: terraform

pool:

vmImage: 'ubuntu-latest'

steps:

- checkout: self

- task: TerraformInstaller@0

inputs:

terraformVersion: '1.5.0'

- script: |

terraform init

terraform plan -out=tfplan

displayName: "Terraform Init and Plan"

- script: terraform apply -auto-approve tfplan

displayName: "Terraform Apply"

- stage: Databricks\_Setup

displayName: "Databricks Cluster and Job"

dependsOn: Terraform\_Deploy

jobs:

- job: databricks

pool:

vmImage: 'ubuntu-latest'

steps:

- checkout: self

- task: UsePythonVersion@0

inputs:

versionSpec: '3.x'

- script: pip install databricks-cli

displayName: "Install Databricks CLI"

- script: |

databricks configure --token <<EOF

https://<your-databricks-instance>

$(DATABRICKS\_TOKEN)

EOF

displayName: "Configure Databricks CLI"

- script: |

databricks clusters create --json-file clusters/cluster\_config.json

displayName: "Create Cluster"

- script: |

databricks libraries install --cluster-id $(ClusterId) --pypi-package pandas

displayName: "Install Libraries"

**📄 Sample Cluster Config (clusters/cluster\_config.json)**

json

CopyEdit

{

"cluster\_name": "sales-cluster",

"spark\_version": "12.2.x-scala2.12",

"node\_type\_id": "Standard\_DS3\_v2",

"autoscale": {

"min\_workers": 2,

"max\_workers": 5

},

"autotermination\_minutes": 30

}

**Note:**

* Store DATABRICKS\_TOKEN securely in Azure DevOps library.
* You can add *notebook runs* similarly using databricks runs submit.

**6. 🧾 Test Cases (Extended)**

**All test cases from earlier are included**, with **additional ones** for Terraform and CI/CD:

**✅ Test Case 11: Terraform Plan Validation**

* **Objective:** Ensure Terraform plan shows no destructive changes.
* **Steps:**
  1. Run terraform plan.
  2. Review output.
* **Expected Result:** No unexpected resource deletions.

**✅ Test Case 12: Azure DevOps Pipeline Execution**

* **Objective:** Validate pipeline runs end-to-end.
* **Steps:**
  1. Trigger pipeline from main branch.
  2. Confirm successful stages.
* **Expected Result:** All stages succeed without error.

**✅ Test Case 13: Cluster Job Run**

* **Objective:** Ensure Databricks job can run automatically.
* **Steps:**
  1. Submit sample job.
  2. Validate logs.
* **Expected Result:** Job completes with SUCCESS state.

**7. 🧪 Test Data**

| **Data Element** | **Description** |
| --- | --- |
| CSV Path | /mnt/sales/raw/sales\_data.csv |
| Parquet Path | /mnt/sales/raw/sales\_data.parquet |
| Databricks Token | Azure DevOps secret |
| Cluster Config | clusters/cluster\_config.json |
| Libraries | pandas, spark-xml |

**8. ✅ Entry Criteria**

* Terraform files reviewed and approved.
* Azure DevOps pipeline configured.
* Secrets stored securely in DevOps Library.

**9. 🏁 Exit Criteria**

* Terraform successfully provisions resources.
* Databricks clusters created and functional.
* Sales data ingested and transformed.
* No critical defects open.

**10. 🗂️ Test Deliverables**

* Terraform execution logs.
* Azure DevOps pipeline run history.
* Databricks job run reports.
* Final Test Summary Report.

**11. 🛠️ Environment**

| **Component** | **Details** |
| --- | --- |
| Azure Region | East US 2 |
| Terraform Version | 1.5.0 |
| Databricks Runtime | 12.2 LTS |
| Cluster Node Type | Standard\_DS3\_v2 |
| Azure DevOps Agent | Ubuntu-latest |

**12. 📈 Sample PySpark Notebook Code**

This example aggregates sales data:

python

CopyEdit

# Read Sales CSV

df = spark.read.csv("/mnt/sales/raw/sales\_data.csv", header=True, inferSchema=True)

# Clean Data

df\_clean = df.dropna(subset=["OrderID", "SalesAmount"])

# Convert Amount to Decimal

from pyspark.sql.functions import col

df\_clean = df\_clean.withColumn("SalesAmount", col("SalesAmount").cast("decimal(18,2)"))

# Aggregate Sales by Region

df\_agg = df\_clean.groupBy("Region").sum("SalesAmount")

# Write to Delta

df\_agg.write.format("delta").mode("overwrite").save("/mnt/sales/processed/sales\_summary")

✅ **Done!**

This **TDD with Terraform and Azure DevOps integration** is now **enterprise-ready**.