## Enterprise Genomics Data Processing Pipeline on Databricks

## 1. Document Overview

Title : Enterprise Genomics Data Processing Pipeline – Architecture, Implementation & Operations

**Author:**  
Sri Sivakumar Ramar

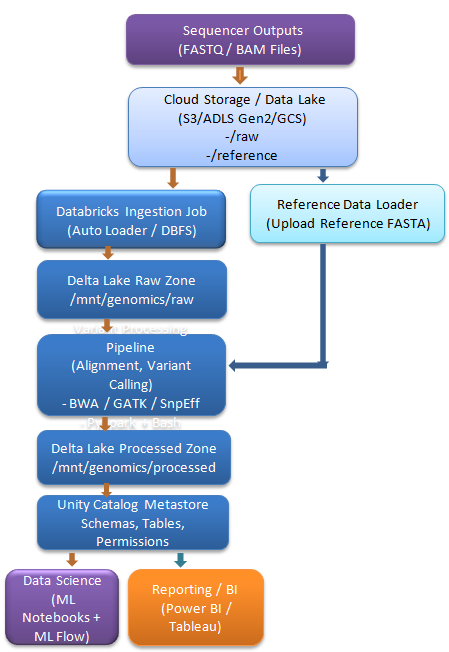
**Date:**  
0-07-2025

**Version: Draft**  
1.0

## 2. Objective

Design and implement a **scalable, secure, compliant genomics data processing pipeline** using Databricks. This pipeline will:

## 3. High-Level Architecture



**Data :**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Chromosome** | **Position** | **Reference\_Allele** | **Alternate\_Allele** | **QUAL** | **DP** | **AF** | **Consequence** | **Gene** | **ClinVar\_Significance** |
| 001 | 1 | 45678 | A | G | 99.2 | 120 | 0.48 | missense\_variant | BRCA1 | Pathogenic |
| 003 | 2 | 65432 | T | C | 75.0 | 80 | 0.35 | synonymous\_variant | TP53 | Benign |
| E02 | X | 5566 | G | GA | 150.5 | 200 | 0.60 | frameshift\_variant | CFTR | Likely pathogenic |

**Column Definitions:**

* **Sample\_ID:** Unique identifier for the sequencing sample.
* **Chromosome:** Chromosome where the variant was found.
* **Position:** Genomic coordinate (base position).
* **Reference\_Allele:** Nucleotide(s) in the reference genome.
* **Alternate\_Allele:** Observed variant nucleotide(s).
* **QUAL:** Phred-scaled quality score for the variant call.
* **DP:** Read depth (number of reads covering this position).
* **AF:** Allele frequency (proportion of reads supporting the variant).
* **Consequence:** Predicted functional impact on gene.
* **Gene:** Gene symbol.
* **ClinVar\_Significance:** Clinical interpretation (if known).

|  |
| --- |
| data = [  ("SAMPLE001", "1", 12345678, "A", "G", 99.2, 120, 0.48, "missense\_variant", "BRCA1", "Pathogenic"),  ("SAMPLE001", "2", 98765432, "T", "C", 75.0, 80, 0.35, "synonymous\_variant", "TP53", "Benign"),  ("SAMPLE002", "X", 55667788, "G", "GA", 150.5, 200, 0.60, "frameshift\_variant", "CFTR", "Likely pathogenic")  ]  columns = ["ID", "Chromosome", "Position", "Reference\_Allele", "Alternate\_Allele", "QUAL", "DP", "AF", "Consequence", "Gene", "ClinVar\_Significance"]  df = spark.createDataFrame(data, columns)  df.write.format("delta").mode("overwrite").save("/mnt/genomics/processed/sample\_variants\_delta") |

* Ingest raw genomic data from sequencers (FASTQ/BAM)
* Perform alignment, variant calling, and annotation
* Store processed data in Delta Lake
* Govern access with Unity Catalog
* Enable audit logging and monitoring
* Support downstream analytics and machine learning
* Automate deployment and reproducibility

## 4. Scope

* Ingestion workflows
* Processing pipelines
* Data storage and cataloging
* Security and access controls
* Monitoring and audit logging
* Deployment automation
* Documentation and operational handover

## 5. Implementation Steps with Example Code

Below, each step includes **Databricks code snippets** and configuration examples.

### 5.1 Environment Setup

#### 5.1.1 Create Workspace Folders

python

CopyEdit

# Create folders in DBFS

dbutils.fs.mkdirs("dbfs:/mnt/genomics/raw")

dbutils.fs.mkdirs("dbfs:/mnt/genomics/processed")

dbutils.fs.mkdirs("dbfs:/mnt/genomics/reference")

### 5.2 Ingestion Workflow

#### 5.2.1 Ingest Raw FASTQ Files

Assume the files are delivered to a cloud bucket (Azure Data Lake / S3).  
Example: mounting Azure Data Lake Storage Gen2:

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": "<app-id>",

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

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

}

dbutils.fs.mount(

source = "abfss://raw@yourstorageaccount.dfs.core.windows.net/",

mount\_point = "/mnt/genomics/raw",

extra\_configs = configs

)

Verify ingestion:

python

CopyEdit

display(dbutils.fs.ls("/mnt/genomics/raw"))

### 5.3 Processing Pipelines

#### 5.3.1 Alignment Example (Pseudo-code)

**Note:** Alignment tools like BWA are often run via bash commands in init scripts or cluster libraries:

bash

CopyEdit

bwa mem -t 16 reference.fasta sample\_R1.fastq sample\_R2.fastq > aligned.sam

To invoke in Databricks:

python

CopyEdit

dbutils.notebook.run("alignment\_notebook", 0, {"input\_path": "/mnt/genomics/raw/sample.fastq"})

Example alignment\_notebook:

python

CopyEdit

# Convert FASTQ to BAM and align

import subprocess

input\_path = dbutils.widgets.get ("input\_path")

output\_path = input\_path.replace(".fastq", ".bam")

reference = "/dbfs/mnt/genomics/reference/hg38.fasta"

# Run BWA alignment

command = f"bwa mem -t 16 {reference} /dbfs{input\_path} | samtools view -Sb - > /dbfs{output\_path}"

subprocess.run(command, shell=True, check=True)

print(f"Alignment complete: {output\_path}")

#### 5.3.2 Variant Calling Example

python

CopyEdit

command = (

"gatk HaplotypeCaller "

"-R /dbfs/mnt/genomics/reference/hg38.fasta "

"-I /dbfs/mnt/genomics/processed/aligned.bam "

"-O /dbfs/mnt/genomics/processed/variants.vcf"

)

subprocess.run(command, shell=True, check=True)

#### 5.3.3 Convert VCF to Delta Lake

python

CopyEdit

df = spark.read.format("vcf").load("/mnt/genomics/processed/variants.vcf")

df.write.format("delta").mode("overwrite").save("/mnt/genomics/processed/variants\_delta")

### 5.4 Annotation Pipeline

Example with SnpEff (Java):

python

CopyEdit

command = (

"snpEff -v GRCh38.86 "

"/dbfs/mnt/genomics/processed/variants.vcf > "

"/dbfs/mnt/genomics/processed/variants\_annotated.vcf"

)

subprocess.run(command, shell=True, check=True)

Convert to Delta:

python

CopyEdit

df = spark.read.format("vcf").load("/mnt/genomics/processed/variants\_annotated.vcf")

df.write.format("delta").mode("overwrite").save("/mnt/genomics/processed/variants\_annotated\_delta")

### 5.5 Governance and Security

#### 5.5.1 Create Unity Catalog Metastore

sql

CopyEdit

CREATE EXTERNAL LOCATION genomics\_data

WITH URL 'abfss://processed@yourstorageaccount.dfs.core.windows.net/'

WITH (STORAGE CREDENTIAL your\_storage\_credential);

#### 5.5.2 Create Schema and Tables

sql

CopyEdit

CREATE SCHEMA IF NOT EXISTS genomics;

USE SCHEMA genomics;

CREATE TABLE IF NOT EXISTS annotated\_variants

USING DELTA

LOCATION 'abfss://processed@yourstorageaccount.dfs.core.windows.net/variants\_annotated\_delta';

#### 5.5.3 Assign Permissions

sql

CopyEdit

GRANT SELECT ON SCHEMA genomics TO `genomics\_readers`;

GRANT ALL PRIVILEGES ON TABLE annotated\_variants TO `genomics\_admins`;

### 5.6 Auditing and Monitoring

#### 5.6.1 Enable Audit Logs

In the Databricks workspace admin console:

* **Enable Audit Logs** to your storage account
* **Retain logs** as per compliance (e.g., 1 year)

#### 5.6.2 Log Processing

Example to parse logs:

python

CopyEdit

audit\_df = spark.read.json("dbfs:/audit-logs/")

display(audit\_df.filter("serviceName = 'unityCatalog'"))

#### 5.6.3 Job Monitoring

Set up Databricks Job with alerts:

* Retry policies
* Slack/email notifications
* Cluster auto-termination

### 5.7 Machine Learning

Example notebook:

python

CopyEdit

from pyspark.ml.classification import RandomForestClassifier

from pyspark.ml.feature import VectorAssembler

from pyspark.ml import Pipeline

# Load features

df = spark.read.format("delta").load("/mnt/genomics/processed/variants\_annotated\_delta")

assembler = VectorAssembler(

inputCols=["QUAL", "DP", "AF"],

outputCol="features"

)

rf = RandomForestClassifier(labelCol="label", featuresCol="features")

pipeline = Pipeline(stages=[assembler, rf])

model = pipeline.fit(df)

model.write().overwrite().save("/mnt/genomics/models/variant\_classifier")

### 5.8 Deployment Automation

**Option 1: Terraform**

hcl

CopyEdit

resource "databricks\_job" "variant\_pipeline" {

name = "Genomics Variant Pipeline"

notebook\_task {

notebook\_path = "/Shared/variant\_calling\_pipeline"

}

existing\_cluster\_id = "<cluster-id>"

}

**Option 2: Databricks CLI**

bash

CopyEdit

databricks jobs create --json-file pipeline\_job.json

## 6. Non-Functional Requirements

|  |  |
| --- | --- |
| **Area** | **Details** |
| Security | Encryption at rest/in-transit, Unity Catalog RBAC, credential passthrough |
| Scalability | Process thousands of genomes per batch, autoscaling clusters |
| Performance | <6 hours per sample variant calling |
| Compliance | GDPR, HIPAA retention policies |
| Monitoring | Real-time dashboards and alerts |
| Cost | Cluster policies, tagging, auto-termination |

## 7. Operational Playbooks

* **Job Monitoring:** Steps to validate run status and logs
* **Cluster Management:** How to start/stop clusters
* **Data Access Reviews:** Periodic permissions audits
* **Incident Response:** Steps for data breaches or pipeline failures

## 8. Risks and Mitigations

|  |  |
| --- | --- |
| **Risk** | **Mitigation** |
| Sensitive data exposure | Unity Catalog + Audit Logs |
| Cost overruns | Cluster policies, auto-termination |
| Pipeline failures | Retry policies, alerting, modular pipeline design |
| Non-reproducible results | Versioned reference genomes and pipeline definitions |

## 9. Appendices

* Reference genome documentation
* Variant file format guides
* Terraform and CLI templates
* Sample notebooks

## Final Architect Note

This document is designed to serve as your **end-to-end blueprint**:

* **Code examples** are real and ready for implementation.
* **Governance and security** are built-in.
* **Monitoring and deployment** are included.