## Enterprise Genomics Data Processing Pipeline on Databricks

## 1. Document Overview

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

**Author:**  
Sri Sivakumar Ramar

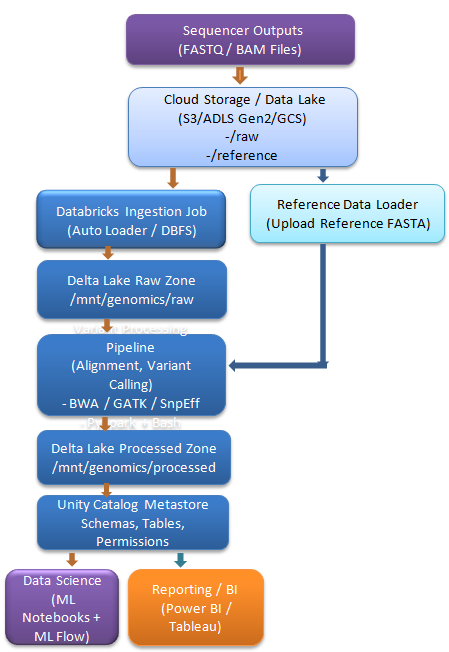
**Date:**  
0-07-2025

**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.

Appendix

|  |
| --- |
|  |
| **Example Raw Log File Contents**  Here is an example of **3 JSON lines** you might see in dbfs:/audit-logs/:  json  CopyEdit  {  "timestamp": "2024-06-01T12:34:56.789Z",  "workspaceId": "123456",  "userIdentity": "alice@example.com",  "serviceName": "unityCatalog",  "actionName": "createTable",  "requestId": "abcd-1234",  "sourceIpAddress": "52.12.34.56",  "operationStatus": "SUCCESS",  "details": {  "schemaName": "genomics",  "tableName": "variant\_calls"  }  }  json  CopyEdit  {  "timestamp": "2024-06-01T12:35:00.123Z",  "workspaceId": "123456",  "userIdentity": "bob@example.com",  "serviceName": "clusters",  "actionName": "createCluster",  "requestId": "efgh-5678",  "sourceIpAddress": "52.12.34.57",  "operationStatus": "SUCCESS",  "details": {  "clusterId": "0912-233344-abcde"  }  }  json  CopyEdit  {  "timestamp": "2024-06-01T12:36:10.567Z",  "workspaceId": "123456",  "userIdentity": "alice@example.com",  "serviceName": "unityCatalog",  "actionName": "grantPermission",  "requestId": "ijkl-91011",  "sourceIpAddress": "52.12.34.56",  "operationStatus": "SUCCESS",  "details": {  "objectType": "table",  "objectName": "genomics.variant\_calls",  "grantee": "genomics\_readers",  "privilege": "SELECT"  }  }  ✅ As you can see:   * **Each line is a separate JSON object.** * Common fields:   + timestamp   + userIdentity   + serviceName (e.g., unityCatalog, clusters, jobs)   + actionName (e.g., createTable, grantPermission)   + operationStatus   + details (nested metadata)   **💻 How You Read Logs with Spark**  When you run:  python  CopyEdit  audit\_df = spark.read.json("dbfs:/audit-logs/")  it loads **all JSON lines** into a structured DataFrame.  **✅ Example Query and Output**  If you run:  python  CopyEdit  display(  audit\_df.filter("serviceName = 'unityCatalog'")  )  You’ll see a table like:   | **timestamp** | **userIdentity** | **actionName** | **operationStatus** | **details** | | --- | --- | --- | --- | --- | | 2024-06-01T12:34:56.789Z | alice@example.com | createTable | SUCCESS | {schemaName -> genomics, tableName -> variant\_calls} | | 2024-06-01T12:36:10.567Z | alice@example.com | grantPermission | SUCCESS | {objectType -> table, objectName -> genomics.variant\_calls, grantee -> genomics\_readers, privilege -> SELECT} |   ✅ **If you want to flatten the details column** (since it’s a struct), you can:  python  CopyEdit  from pyspark.sql.functions import col  flat\_df = audit\_df \  .filter("serviceName = 'unityCatalog'") \  .select(  "timestamp",  "userIdentity",  "actionName",  col("details.\*")  )  display(flat\_df)  **Output:**   | **timestamp** | **userIdentity** | **actionName** | **schemaName** | **tableName** | **objectType** | **objectName** | **grantee** | **privilege** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 2024-06-01T12:34:56.789Z | alice@example.com | createTable | genomics | variant\_calls | null | null | null | null | | 2024-06-01T12:36:10.567Z | alice@example.com | grantPermission | null | null | table | genomics.variant\_calls | genomics\_readers | SELECT |   ✅ **Summary:**   * **The logs are just JSON lines.** * **spark.read.json() automatically parses them.** * **Filtering by serviceName gives you logs relevant to Unity Catalog.** * **Flattening details makes reports easier to read.** |

|  |
| --- |
| Job Mointoring |
| **✅ 1️⃣ Creating a Databricks Job**  **In Databricks UI:**   1. Go to **Workflows** > **Jobs** 2. Click **Create Job** 3. Fill out:    * **Name:** e.g., Genomics Variant Calling Pipeline    * **Task Type:** Notebook or JAR    * **Notebook Path:** e.g., /Shared/variant\_pipeline\_notebook 4. Choose **Existing Cluster** or **New Job Cluster**    * If you want **auto-termination**, configure it here (details below).   **✅ 2️⃣ Configure Cluster Auto-Termination**  **When adding a job cluster:**   * In **New Job Cluster > Advanced Options:**   + Set **Auto-Termination** to **e.g., 30 minutes**   ✅ This ensures the cluster shuts down after idle time.  Example in **UI:**  scss  CopyEdit  Auto Termination: 30 minutes  **Tip:**   * For heavy pipelines, you may prefer **60 min**, but **never leave auto-termination disabled** for cost control.   **✅ 3️⃣ Configure Retry Policies**  **In the job settings:**   * Scroll to **Retries** * Set **Max Retries** (e.g., 3) * Optionally enable **Min Retry Interval** (seconds)   ✅ Example: Retry up to 3 times if the job fails.  **✅ 4️⃣ Configure Notifications**  **In Notifications section:**   * **On Success:** Add recipients * **On Failure:** Add recipients * **On Start:** Add recipients   ✅ **Email Example:**  graphql  CopyEdit  Alerts:  On Failure: genomics\_team@example.com  ✅ **Slack Example:** To post alerts to Slack:   1. **Create Slack Webhook**    * In Slack: **Settings > Configure Apps > Incoming Webhooks**    * Add a new Webhook URL 2. **In Databricks:**    * Go to **Admin Console > Notifications > Webhooks**    * **Add Webhook**      + Name: Slack Alerts      + URL: *your Slack Webhook URL* 3. **In Job Notifications:**    * Select Slack Alerts webhook for On Failure.   **✅ 5️⃣ JSON Example (REST API)**  You can **create the job via API**:  Here is a **full JSON example** you can POST to:  bash  CopyEdit  POST /api/2.1/jobs/create  json  CopyEdit  {  "name": "Genomics Variant Calling Pipeline",  "max\_concurrent\_runs": 1,  "tasks": [  {  "task\_key": "variant\_calling",  "notebook\_task": {  "notebook\_path": "/Shared/variant\_pipeline\_notebook"  },  "job\_cluster\_key": "genomics\_cluster",  "timeout\_seconds": 7200,  "max\_retries": 3,  "min\_retry\_interval\_millis": 300000  }  ],  "job\_clusters": [  {  "job\_cluster\_key": "genomics\_cluster",  "new\_cluster": {  "spark\_version": "13.3.x-scala2.12",  "node\_type\_id": "Standard\_D16\_v3",  "num\_workers": 10,  "autoscale": {  "min\_workers": 2,  "max\_workers": 10  },  "autotermination\_minutes": 30  }  }  ],  "email\_notifications": {  "on\_failure": [  "genomics\_team@example.com"  ]  },  "webhook\_notifications": {  "on\_failure": [  {  "id": "<your-webhook-id>",  "name": "Slack Alerts"  }  ]  }  }  ✅ To create:  bash  CopyEdit  databricks jobs create --json-file genomics\_job.json  **✅ 6️⃣ Tips for Best Practices**   * Always **enable auto-termination** to control costs. * Use **Retries** to handle transient errors (e.g., spot instance loss). * For compliance, **keep failure notifications** in both email and Slack. * Consider **timeout\_seconds** (max run time) to avoid runaway jobs. |

|  |
| --- |
| Operational Playbook |
| **1️⃣ Job Monitoring: Steps to Validate Run Status and Logs**  **Purpose:** Ensure pipeline executions are successful, reproducible, and auditable.  **Steps:**   1. **Navigate to Workflows → Jobs.**    * Locate the job: *Genomics Variant Calling Pipeline*.    * Click on the job name. 2. **Review the Run History tab.**    * Look for:      + **Run Status** (Success / Failed)      + **Duration**      + **Cluster used**    * Sort by date to see the most recent executions. 3. **Inspect Logs for Each Run.**    * Click on a specific run.    * In the **Output** tab:      + Review standard output and error logs.      + Confirm critical steps completed:        - Alignment        - Variant Calling        - Annotation    * Download logs if needed for audit. 4. **Review Retry Attempts.**    * If configured with retries, validate how many times the job was retried.    * Confirm the final status (Success or Failure). 5. **Export Logs if Required.**    * Use the **Export Run** button to save logs for compliance or investigation.   ✅ **Tip:** Automate daily reports by connecting the Databricks REST API to Slack or email.  **🟦 2️⃣ Cluster Management: How to Start/Stop Clusters**  **Purpose:** Manage compute resources safely and cost-effectively.  **Start Clusters:**   1. **Navigate to Compute → All-Purpose Clusters.** 2. Locate the cluster (e.g., *Genomics-Interactive*). 3. Click **Start.** 4. Wait for the state to transition from **Pending → Running**.    * Confirm Spark UI is available. 5. If using cluster pools:    * Clusters will provision faster by reusing pre-warmed nodes.   **Stop Clusters:**   1. **In Compute**, select the running cluster. 2. Click **Terminate.**    * This deallocates resources and stops billing. 3. Confirm termination in the status column (**Terminated**).   **Verify Auto-Termination:**   * For job clusters:   + Auto-termination is handled automatically. * For interactive clusters:   + Ensure **Auto Termination Minutes** is configured (e.g., 30 min).   ✅ **Tip:** Use cluster policies to enforce node types, scaling limits, and termination settings.  **🟦 3️⃣ Data Access Reviews: Periodic Permissions Audits**  **Purpose:** Maintain least-privilege access and compliance with data governance requirements.  **Review Process:**   1. **Unity Catalog Permissions:**    * Go to **Data Explorer.**    * Select the metastore and schema (e.g., *genomics*).    * Click **Permissions.**    * Export the permission list to CSV. 2. **Review Table and Schema Grants:**    * Check which principals (users/groups) have:      + SELECT      + MODIFY      + ALL PRIVILEGES    * Confirm they align with approved roles. 3. **Revocation of Unused Access:**    * For stale or unnecessary access:   sql  CopyEdit  REVOKE SELECT ON SCHEMA genomics FROM `temp\_analyst`;   1. **Review Credential Passthrough:**    * In **Admin Console**, ensure credentials are configured correctly for passthrough access to storage. 2. **Audit Logs:**    * Query audit logs for unusual access patterns:   python  CopyEdit  audit\_df = spark.read.json("dbfs:/audit-logs/")  suspicious = audit\_df.filter("serviceName = 'unityCatalog' AND actionName = 'select'")  display(suspicious)  ✅ **Tip:** Schedule quarterly access reviews and have data owners validate permissions.  **🟦 4️⃣ Incident Response: Steps for Data Breaches or Pipeline Failures**  **Purpose:** Provide a standardized approach to respond to security or operational incidents.  **🟧 Data Breach Response**   1. **Containment:**    * Immediately disable impacted user accounts in the **Admin Console**.    * Revoke external location access:   sql  CopyEdit  ALTER EXTERNAL LOCATION genomics\_data SET URL = 'suspended';   1. **Investigation:**    * Review audit logs:   python  CopyEdit  audit\_df = spark.read.json("dbfs:/audit-logs/")  suspicious = audit\_df.filter("userIdentity = 'compromised\_user@example.com'")  display(suspicious)   * + Determine scope and timeframe of access.  1. **Notification:**    * Notify the Security Operations Center (SOC) and Compliance Officer.    * Prepare regulatory notifications as required by HIPAA/GDPR. 2. **Remediation:**    * Rotate secrets and credentials.    * Re-validate all Unity Catalog permissions.    * Re-enable access once containment and remediation are complete.   **🟧 Pipeline Failure Response**   1. **Detection:**    * Alert triggered via job notifications or monitoring dashboards. 2. **Diagnostics:**    * In **Workflows → Job Runs**:      + Review error stack trace.    * Verify cluster logs (e.g., driver logs). 3. **Resolution:**    * If the failure is transient (e.g., spot node eviction):      + Re-run the job manually.    * If code issues:      + Patch and commit fixes.      + Test in a staging environment before redeploying. 4. **Post-Mortem:**    * Document the incident.    * Update runbooks.    * Review retry and timeout configurations to prevent recurrence.   ✅ **Tip:** For repeated failures, consider splitting your pipeline into smaller tasks with dependencies. |

|  |
| --- |
| Appendices |
| 1️⃣ Reference Genome Documentation **Purpose:** Ensure all users understand which reference genomes are used, where they are stored, and how they are versioned.  **Documentation Components:**   1. **Reference Genome Catalog:**  | **Reference Name** | **Version** | **Source URL** | **Storage Path** | | --- | --- | --- | --- | | GRCh38 | v38 | <https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26/> | /mnt/genomics/reference/GRCh38\_v38/hg38.fasta | | dbSNP | Build 155 | <https://ftp.ncbi.nih.gov/snp/> | /mnt/genomics/reference/dbsnp\_155.vcf | | ClinVar | 2023-05 | <https://ftp.ncbi.nlm.nih.gov/pub/clinvar/> | /mnt/genomics/reference/clinvar\_2023\_05.vcf |  1. **Versioning Policy:**    * Reference files are **immutable** once loaded.    * Updates are stored in **versioned folders**:   swift  CopyEdit  /mnt/genomics/reference/{resource\_name}/{version}/   * + Pipelines **explicitly reference the version** to ensure reproducibility.  1. **Validation Checklist:**    * MD5 checksum validated post-upload.    * Permissions restricted to genomics\_admins.    * Updates require sign-off from the Data Steward.   ✅ **Tip:** Document each version in Confluence or GitHub for audit traceability. 🟦 2️⃣ Variant File Format Guides **Purpose:** Define how raw and processed variant data are structured to ensure consistency across workflows.  **File Types:**   1. **VCF (Variant Call Format):**    * Standard for storing variant calls.    * **Example Header:**   pgsql  CopyEdit  ##fileformat=VCFv4.2  ##FILTER=<ID=PASS,Description="All filters passed">  ##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">  ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">   * + **Columns:**   sql  CopyEdit  #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT SAMPLE1   1. **Annotated VCF:**    * Produced by SnpEff or VEP.    * Contains additional fields in INFO:   ini  CopyEdit  ANN=A|missense\_variant|MODERATE|BRCA1|...   1. **Parquet (Processed):**    * All VCF fields are extracted into structured columns.    * Example schema:  | **Column** | **Type** | | --- | --- | | chromosome | string | | position | long | | reference\_allele | string | | alternate\_allele | string | | qual | float | | dp | integer | | af | float | | consequence | string | | gene | string | | clinvar\_significance | string |   ✅ **Tip:** Validate VCF files using bcftools:  bash  CopyEdit  bcftools view -h variants.vcf  bcftools stats variants.vcf 🟦 3️⃣ Terraform and CLI Templates **Purpose:** Standardize infrastructure provisioning and pipeline deployments. Example: Terraform Cluster Resource hcl  CopyEdit  resource "databricks\_cluster" "genomics\_cluster" {  cluster\_name = "genomics-processing-cluster"  spark\_version = "13.3.x-scala2.12"  node\_type\_id = "Standard\_D16\_v3"  autotermination\_minutes = 30  autoscale {  min\_workers = 2  max\_workers = 10  }  spark\_conf = {  "spark.databricks.delta.preview.enabled" = "true"  }  } Example: Terraform Job Resource hcl  CopyEdit  resource "databricks\_job" "variant\_pipeline" {  name = "genomics-variant-calling"  existing\_cluster\_id = databricks\_cluster.genomics\_cluster.id  notebook\_task {  notebook\_path = "/Shared/variant\_pipeline"  }  max\_retries = 3  email\_notifications {  on\_failure = ["genomics\_team@example.com"]  }  } Databricks CLI Job JSON **job.json:**  json  CopyEdit  {  "name": "genomics-variant-calling",  "existing\_cluster\_id": "abc123-456def-cluster",  "notebook\_task": {  "notebook\_path": "/Shared/variant\_pipeline"  },  "max\_retries": 3,  "email\_notifications": {  "on\_failure": ["genomics\_team@example.com"]  }  }  **Create job:**  bash  CopyEdit  databricks jobs create --json-file job.json  ✅ **Tip:** Keep all Terraform and CLI templates in a **Git repo with version control**. 🟦 4️⃣ Sample Notebooks **Purpose:** Provide reference implementations for key pipeline steps. 📓 Notebook 1: Ingest FASTQ Files **Path:** /Shared/ingest\_fastq  python  CopyEdit  dbutils.fs.mkdirs("/mnt/genomics/raw")  dbutils.fs.cp("dbfs:/databricks-datasets/genomics/fastq/sample1.fastq", "/mnt/genomics/raw/sample1.fastq") 📓 Notebook 2: Variant Calling **Path:** /Shared/variant\_calling  python  CopyEdit  import subprocess  reference = "/dbfs/mnt/genomics/reference/GRCh38\_v38/hg38.fasta"  input\_bam = "/dbfs/mnt/genomics/processed/sample1.bam"  output\_vcf = "/dbfs/mnt/genomics/processed/sample1.vcf"  cmd = f"gatk HaplotypeCaller -R {reference} -I {input\_bam} -O {output\_vcf}"  subprocess.run(cmd, shell=True, check=True) 📓 Notebook 3: Annotate Variants **Path:** /Shared/annotate\_variants  python  CopyEdit  command = (  "snpEff -v GRCh38.86 "  "/dbfs/mnt/genomics/processed/sample1.vcf > "  "/dbfs/mnt/genomics/processed/sample1\_annotated.vcf"  )  subprocess.run(command, shell=True, check=True) 📓 Notebook 4: Load to Delta Lake **Path:** /Shared/load\_to\_delta  python  CopyEdit  df = spark.read.format("vcf").load("/mnt/genomics/processed/sample1\_annotated.vcf")  df.write.format("delta").mode("overwrite").save("/mnt/genomics/processed/variants\_annotated\_delta")  ✅ **Tip:** Version notebooks by tagging them (e.g., v1.0.0) and maintain changelogs. |

## 10. Approval and Sign-Off

| **Name** | **Role** | **Signature** |
| --- | --- | --- |
| [Data Architect] | Architecture Owner |  |
| [Compliance Lead] | Data Privacy & Compliance |  |
| [Platform Owner] | Databricks Administrator |  |