

Especialización de Data Analytics

Módulo de DataOps

Alumno. Paul Efrén Santos Andrade / DNI: 47062815

Examen parcial: Implementación de un Pipeline en Jenkins

1. Problemática a Resolver

En entornos de desarrollo y análisis de datos, la ejecución manual de scripts puede generar problemas como inconsistencias en los datos, errores humanos y dificultades en el mantenimiento. Se requiere un proceso automatizado que garantice la correcta ejecución de los scripts en cada etapa del flujo de datos.

2. Beneficios de Jenkins ante el Problema

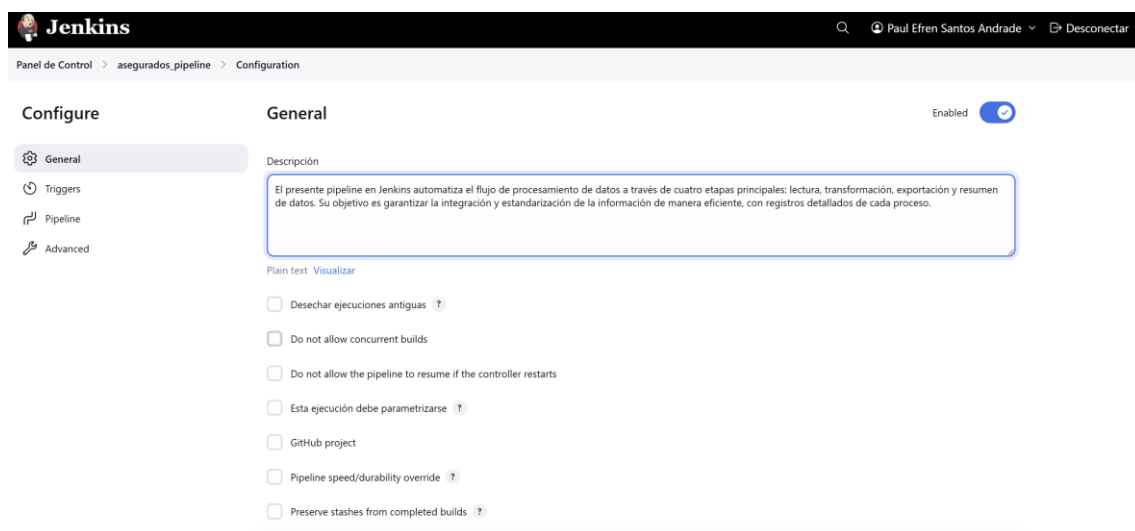
Jenkins es una herramienta de integración continua que permite:

- Automatizar la ejecución de scripts en diferentes etapas del pipeline.
- Generar registros (logs) detallados de cada proceso, fundamentales para el monitoreo, depuración y mantenimiento de sistemas informáticos.
- Monitorear el estado del pipeline en tiempo real.
- Integrarse con sistemas de notificación por correo en caso de fallos o éxito, los cuales pueden ser complementados con información básica de los procesos.

3. Despliegue del Pipeline en Jenkins

a) Creación del Pipeline

1. Acceder a Jenkins y crear un nuevo Pipeline.



2. En la sección *Pipeline*, seleccionar *Pipeline Script* y pegar el código.

Panel de Control > asegurados_pipeline > Configuration

Configure

- General
- Triggers
- Pipeline**
- Advanced

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script

```
Script ?
6  SCRIPT_PATH = 'D:\\dataops_certus\\'
7  LOG_PATH = 'D:\\dataops_certus\\logs'
8  SUMMARY_FILE = 'D:\\dataops_certus\\logs\\summary.txt' // Archivo de resumen
9
10
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
```

try sample Pipeline...

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Advanced

3. Guardar el pipeline.

Panel de Control > asegurados_pipeline >

asegurados_pipeline

[editar descripción](#)

El presente pipeline en Jenkins automatiza el flujo de procesamiento de datos a través de cuatro etapas principales: lectura, transformación, exportación y resumen de datos. Su objetivo es garantizar la integración y estandarización de la información de manera eficiente, con registros detallados de cada proceso.

Stage View

No data available. This Pipeline has not yet run.

Enlaces permanentes

Builds

No builds

4. Ejecutar el pipeline.

Panel de Control > asegurados_pipeline >

Status

asegurados_pipeline

editar descripción

El presente pipeline en Jenkins automatiza el flujo de procesamiento de datos a través de cuatro etapas principales: lectura, transformación, exportación y resumen de datos. Su objetivo es garantizar la integración y estandarización de la información de manera eficiente, con registros detallados de cada proceso.

Stage View

	Preparar	READ	TRANSFORM	EXPORT
Average stage times:	1s	1s	942ms	5s
#1 22:01 No Changes	1s	1s	942ms	5s

Enlaces permanentes

Builds

Today

#1 22:01

REST API Jenkins 2.492.2

La ejecución completa del pipeline

Panel de Control > asegurados_pipeline >

Status

asegurados_pipeline

editar descripción

El presente pipeline en Jenkins automatiza el flujo de procesamiento de datos a través de cuatro etapas principales: lectura, transformación, exportación y resumen de datos. Su objetivo es garantizar la integración y estandarización de la información de manera eficiente, con registros detallados de cada proceso.

Stage View

	Preparar	READ	TRANSFORM	EXPORT	SUMMARY	Declarative: Post Actions
Average stage times: (full run time: ~1min 20s)	1s	1s	942ms	47s	22s	5s
#1 22:01 No Changes	1s	1s	942ms	47s	22s	5s

Enlaces permanentes

Builds

Today

#1 22:01

REST API Jenkins 2.492.2

4. Código del Pipeline

El siguiente código de Jenkins define un pipeline para la ejecución automática de scripts de procesamiento de datos en Python, con las siguientes etapas:

Etapas del Pipeline

a) READ (Lectura de Datos)

- Se ejecuta el script `data_read.py`, el cual lee un archivo `txt` y lo almacena en un temporal `csv`.

b) TRANSFORM (Transformación de Datos)

- Se ejecuta el script `data_transform.py`, que limpia y procesa la información.

c) EXPORT (Exportación de Datos)

- Se ejecuta el script `data_export.py`, el cual almacena los datos procesados en el directorio de salida, en el archivo `clientes_ordenados.xlsx`

d) SUMMARY (Resumen del Proceso)

- Se ejecuta `data_summary.py`, que genera un resumen que se almacena en el archivo `summary.txt` con estadísticas clave (cantidad de registros procesados, errores, etc.).

```

pipeline {
    agent any

    environment {
        PYTHON_EXEC = 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python312\\python.exe'
        SCRIPT_PATH = 'D:\\dataops_certus'
        LOG_PATH = 'D:\\dataops_certus\\logs'
        SUMMARY_FILE = 'D:\\dataops_certus\\logs\\summary.txt' // Archivo de resumen
    }

    stages {
        stage('Preparar') {
            steps {
                script {
                    bat "if not exist \"%{LOG_PATH}%\" mkdir \"%{LOG_PATH}%\""
                    bat "\"${PYTHON_EXEC}\" --version > \"%{LOG_PATH}\\python_version.log\" 2>&1"
                    echo "Versión de Python verificada. Logs guardados en
                    %{LOG_PATH}\\python_version.log"
                }
            }
        }

        stage('READ') {
            steps {
                script {
                    def logFile = "${LOG_PATH}\\read_stage.log"
                    if (fileExists("${SCRIPT_PATH}\\data_read.py")) {
                        bat "\"${PYTHON_EXEC}\" \"%{SCRIPT_PATH}\\data_read.py\" > \"%{logFile}%\" 2>&1"
                    } else {
                        error "El archivo data_read.py no existe en %{SCRIPT_PATH}"
                    }
                }
            }
        }

        stage('TRANSFORM') {
            steps {
                script {
                    def logFile = "${LOG_PATH}\\transform_stage.log"
                    if (fileExists("${SCRIPT_PATH}\\data_transform.py")) {
                        bat "\"${PYTHON_EXEC}\" \"%{SCRIPT_PATH}\\data_transform.py\" > \"%{logFile}%\"
                        2>&1"
                    } else {
                        error "El archivo data_transform.py no existe en %{SCRIPT_PATH}"
                    }
                }
            }
        }

        stage('EXPORT') {
            steps {
                script {
                    def logFile = "${LOG_PATH}\\export_stage.log"
                    if (fileExists("${SCRIPT_PATH}\\data_export.py")) {
                        bat "\"${PYTHON_EXEC}\" \"%{SCRIPT_PATH}\\data_export.py\" > \"%{logFile}%\" 2>&1"
                    } else {
                        error "El archivo data_export.py no existe en %{SCRIPT_PATH}"
                    }
                }
            }
        }

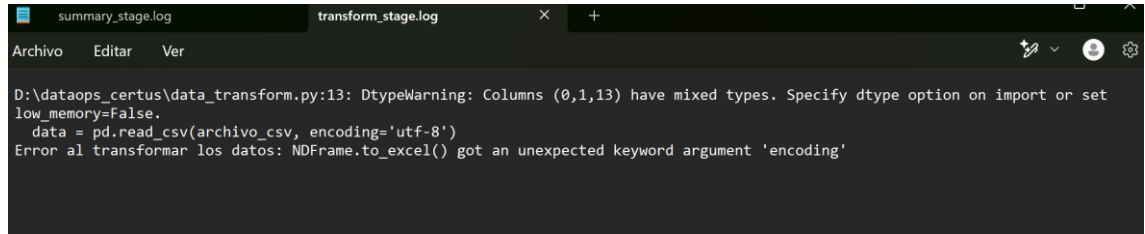
        stage('SUMMARY') {
            steps {
                script {
                    def logFile = "${LOG_PATH}\\summary_stage.log"
                    if (fileExists("${SCRIPT_PATH}\\data_summary.py")) {
                        bat "\"${PYTHON_EXEC}\" \"%{SCRIPT_PATH}\\data_summary.py\" > \"%{logFile}%\"
                        2>&1"
                    } else {
                        if (!fileExists(SUMMARY_FILE)) {
                            error "El archivo de resumen no se generó correctamente."
                        } else {
                            error "El archivo data_summary.py no existe en %{SCRIPT_PATH}"
                        }
                    }
                }
            }
        }

        post {
            always {
                script {
                    def summaryContent = "No summary available."
                    if (fileExists(SUMMARY_FILE)) {
                        summaryContent = readFile(SUMMARY_FILE)
                    }
                    mail to: 'paulefren16@gmail.com',
                        subject: "Build ${currentBuild.fullDisplayName}",
                        body: """
                        <p>The build was <b>${currentBuild.result}</b></p>
                        <p>Check console output at <a href=\"%{env.BUILD_URL}%\">${env.BUILD_URL}</a> to
                        view the results.</p>
                        <h3>Dataset Summary:</h3>
                        <pre>${summaryContent}</pre>
                        """,
                        mimeType: 'text/html'
                }
            }
            failure {
                mail to: 'paulefren16@gmail.com',
                    subject: "Failed Build ${currentBuild.fullDisplayName}",
                    body: "The build FAILED: Check console output at ${env.BUILD_URL} to view the results."
            }
        }
    }
}

```

5. Monitoreo del Pipeline

- Se puede visualizar en la interfaz gráfica de Jenkins.
- Revisar los logs generados en la carpeta D:\...\logs.



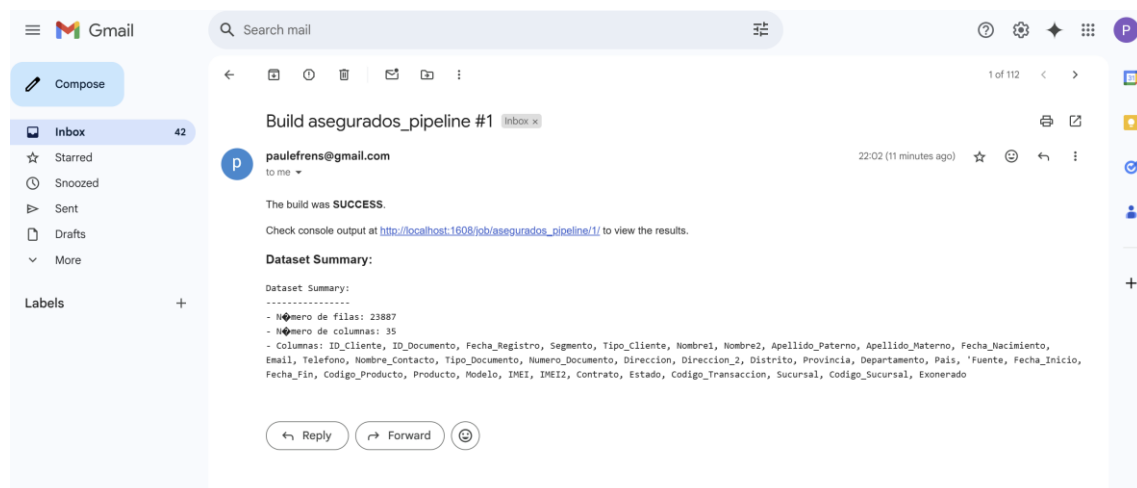
```
summary_stage.log  transform_stage.log
Archivo  Editar  Ver

D:\dataops_certus\data_transform.py:13: DtypeWarning: Columns (0,1,13) have mixed types. Specify dtype option on import or set
low_memory=False.
  data = pd.read_csv(archivo_csv, encoding='utf-8')
Error al transformar los datos: NDFrame.to_excel() got an unexpected keyword argument 'encoding'
```

- Validar la existencia del archivo summary.txt como resultado final.

6. Envío de Notificaciones Mediante Mail

La configuración de Jenkins para el envío de correos de notificación de la ejecución del pipeline.



7. Conclusiones y Recomendaciones

- **Automatización eficiente:** Jenkins permite ejecutar los scripts de manera automática y organizada.
- **Monitoreo y Logs:** Se pueden revisar logs en tiempo real y recibir notificaciones de errores.
- **Facilidad de Integración:** Es posible conectar Jenkins con sistemas de almacenamiento y bases de datos para una mejor gestión.
- **Recomendación:** Se sugiere validar que todos los scripts y rutas estén configurados correctamente antes de ejecutar el pipeline.

Este pipeline facilita la ejecución de procesos de transformación de datos en un entorno controlado y seguro, mejorando la trazabilidad y eficiencia del flujo de trabajo.

8. Secuencia de código

```
# data_read.py
import pandas as pd
import os

# Definir ruta base
ruta_base = r"D:\dataops_certus"

# Ruta del archivo de entrada y salida
archivo_txt = os.path.join(ruta_base, "Asegurados.txt")
archivo_csv = os.path.join(ruta_base, "clientes.csv")

try:
    # Leer el archivo TXT
    data = pd.read_csv(archivo_txt, delimiter='|', encoding='utf-8')

    # Guardar los datos en un archivo CSV
    data.to_csv(archivo_csv, index=False, encoding='utf-8')

    print(f"Datos leídos y guardados en {archivo_csv}")
except Exception as e:
    print(f"Error al leer el archivo TXT: {e}")

# data_transform.py
import pandas as pd
import os

# Definir ruta base
ruta_base = r"D:\dataops_certus"

# Ruta de archivos
archivo_csv = os.path.join(ruta_base, "clientes.csv")
archivo_excel = os.path.join(ruta_base, "clientes_ordenados.xlsx")

try:
    # Leer el archivo CSV
    data = pd.read_csv(archivo_csv, encoding='utf-8')

    # Verificar si la columna "Nombre1" existe
    if "Nombre1" not in data.columns:
        raise ValueError("La columna 'Nombre1' no existe en el archivo CSV.")

    # Ordenar los datos por "Nombre1"
    data_ordenada = data.sort_values(by="Nombre1")

    # Exportar a Excel
    data_ordenada.to_excel(archivo_excel, index=False, encoding='utf-8')

    print(f"Datos ordenados y guardados en {archivo_excel}")
except Exception as e:
    print(f"Error al transformar los datos: {e}")

# data_export.py
import pandas as pd
import os

# Definir ruta base
ruta_base = r"D:\dataops_certus"

# Ruta de archivos
archivo_csv = os.path.join(ruta_base, "clientes.csv")
archivo_excel = os.path.join(ruta_base, "clientes_ordenados.xlsx")

try:
    # Leer el archivo CSV con especificación de tipos de datos
    data = pd.read_csv(archivo_csv, encoding='utf-8', dtype={
        0: str, # Columna 0 como string
        1: str, # Columna 1 como string
        13: str # Columna 13 como string
    }, low_memory=False)

    # Verificar si la columna "Nombre1" existe
    if "Nombre1" not in data.columns:
        raise ValueError("La columna 'Nombre1' no existe en el archivo CSV.")

    # Ordenar los datos por "Nombre1"
    data_ordenada = data.sort_values(by="Nombre1")

    # Exportar a Excel (eliminar encoding y usar openpyxl como motor)
    data_ordenada.to_excel(archivo_excel, index=False, engine="openpyxl")

    print(f"Datos ordenados y guardados en {archivo_excel}")
except Exception as e:
    print(f"Error al transformar los datos: {e}")

# data_summary.py
import os
import pandas as pd

# Definir ruta base
ruta_base = r"D:\dataops_certus"

# Ruta de archivos
archivo_excel = os.path.join(ruta_base, "clientes_ordenados.xlsx")

# Cargar el dataset final
df = pd.read_excel(archivo_excel) # Cambiado de read_csv a read_excel

# Generar un resumen
summary = f"""
Dataset Summary:
-----
- Numero de filas: {df.shape[0]}
- Numero de columnas: {df.shape[1]}
- Columnas: {', '.join(df.columns)}
"""

# Definir la ruta del archivo de resumen
ruta_summary = os.path.join(ruta_base, "summary.txt")

# Crear la carpeta de logs si no existe
os.makedirs(os.path.dirname(ruta_summary), exist_ok=True)

# Guardar el resumen en un archivo
with open(ruta_summary, "w") as f:
    f.write(summary)

print("Resumen generado exitosamente.")
```