**Analysis of EV Impact on National Grid Project**

**Java Documentation**

By Tejas Aiyar

**Introduction:**

The Java program serves a critical role within a larger project, functioning as an integral part of a data processing pipeline. This program is designed to address the challenges of handling diverse data formats and ensuring the integrity of the data before further processing. The program's primary focus lies in converting various input file formats into CSV files, which are pivotal for downstream data cleansing and analysis through FluentD.

**Capabilities and Supported Input Formats:**

The Java program offers the capability to perform multiple tasks vital to the data processing pipeline. It not only converts input files into CSV format, a common and versatile data format, but also conducts thorough malware scans on these files prior to conversion. By supporting various input file formats such as Excel (both XLS and XLSX), JSON, Avro, and Parquet, the program can handle a wide range of data sources, ensuring flexibility and adaptability to varying data origins.

**Significance of CSV Output Files:**

The generated CSV output files serve as a crucial bridge between the disparate input file formats and the downstream data cleansing and analysis stages, powered by FluentD. CSV, known for its simplicity and compatibility, acts as the common ground upon which diverse data structures converge, making it ideal for subsequent data processing.

**Context and Integration:**

This Java program functions within a broader ecosystem, residing alongside FluentD on the same AWS instance. The overall architecture involves the seamless flow of data from various sources, including data about petrol, diesel, and electric cars. The larger project aims to employ predictive modelling to assess the UK's national grid's capacity to accommodate an increase in electric cars. However, the focus of this documentation is on the Java program's role within this system, emphasising its preprocessing capabilities and data format conversion tasks.

**Data Quality and Security:**

The program ensures the quality and security of the data through two key mechanisms: malware scanning and format standardisation. Before any conversion occurs, the program meticulously scans incoming files for malware, mitigating potential risks. Subsequently, it converts the files into the universally accepted CSV format, which not only simplifies data handling but also minimises data discrepancies and inconsistencies.

**Summary:**

In summary, the Java program is a pivotal component of a comprehensive data processing pipeline. It transforms diverse input file formats into a standardised CSV format, promoting seamless data handling and analysis through FluentD. Moreover, its incorporation of malware scanning underscores its commitment to data integrity and security. As a crucial enabler of the broader project's goals, this program's capabilities are integral to achieving accurate insights into the UK's national grid readiness for an influx of electric cars.

**Installation and Setup:**

This section provides comprehensive step-by-step instructions for downloading, installing, and configuring the Java program on an AWS instance. Additionally, it guides you through the process of setting up ClamAV, a vital component for malware scanning.

**Installing and Configuring the Java Program:**

1. **Download the Java Program:**

* Download the Java program's executable JAR file from the provided source.

1. **Upload JAR File to AWS Instance:**

* Use a tool like `scp` to upload the JAR file to your AWS instance. Run the following command in your local terminal:

     scp path/to/java-program.jar ec2-user@your-instance-ip:/path/on/instance/

1. **Configure the Java Program:**

* Connect to your AWS instance using SSH.
* Navigate to the directory where the JAR file is located.
* Edit the Java program's source code to set the correct file paths for source and output folders.

1. **Run the Java Program:**

* Execute the Java program by running the following command:

     java -jar java-program.jar

**Installing and Configuring ClamAV:**

1. **Installing ClamAV:**

* Connect to your AWS instance using SSH.
* Run the following commands to install ClamAV:

     sudo yum install -y clamav clamd

1. **Update ClamAV Definitions:**

* Run the following commands to update ClamAV's malware definitions:

     sudo freshclam

1. **Start ClamAV Service:**

* Start the ClamAV service using the following commands:

     sudo systemctl start clamd@scan

     sudo systemctl enable clamd@scan

1. **Configure ClamAV .conf Files:**

* Upload the provided `.conf` files to your AWS instance. You can use `scp` as shown earlier.
* Place the `.conf` files in the appropriate directory, such as `/etc/clamd.d/`.

1. **Run ClamAV at Boot:**

* Configure ClamAV to run at boot by executing:

     sudo systemctl enable clamd@scan

**Final Steps:**

1. **Run the Java Program and ClamAV:**

* Ensure that ClamAV is running: `sudo systemctl start clamd@scan`
* Execute the Java program as needed.

1. **Verify Configuration:**

* Monitor logs and outputs to ensure that both the Java program and ClamAV are functioning correctly.

By following these steps, you'll successfully install and configure both the Java program and ClamAV on your AWS instance. The provided `.conf` files for ClamAV ensure that users can seamlessly set up the program without manual configuration. This integrated setup facilitates effective data processing, malware scanning, and file conversion for your larger project.

**Usage Instructions:**

This section provides a detailed guide on how to use the Java program to convert various file types to CSV format. The program is designed for simplicity and efficiency, requiring minimal user input.

1. **Run the Java Program:**

* Connect to your AWS instance using SSH.
* Navigate to the directory where the JAR file is located.
* Run the Java program by executing the following command:

     java -jar java-program.jar

1. **Conversion Process:**

* Upon execution, the program will automatically convert files within the specified source folder to CSV format.
* All supported file types (Excel, JSON, Avro, Parquet) present in the source folder will be converted to corresponding CSV files.

1. **Input and Output Folders:**

* The program requires two essential parameters: the source folder and the output folder.
* By default, the program will use the current directory as both the source and output folders.
* To specify custom folders, modify the source and output paths in the Java program's source code.

1. **Specifying Custom Folders:**

* Open the Java program's source code (InterfaceMain class).
* Locate the lines defining the source and output folder paths.
* Modify the paths to match your desired folders, ensuring they exist on your AWS instance.

1. **Running the Conversion:**

* After configuring the source and output folders, save the Java program's source code.
* Run the program again using the command:

     java -jar java-program.jar

* All files within the source folder will be converted to CSV format and saved in the specified output folder.

1. **Monitoring and Verifying:**

* Monitor the console for any conversion errors or notifications.
* Once the program finishes execution, navigate to the output folder to find the generated CSV files.

1. **Automated Malware Scanning:**

* Before the conversion process begins, the program automatically performs malware scanning using ClamAV.
* This ensures that all input files are safe and secure for processing.

**Example Usage:**

Suppose you have the Java program JAR file named `java-program.jar` and you want to convert files located in `/home/ec2-user/input` to CSV format, saving the CSV files in `/home/ec2-user/output`. Here's how you would run the program:

1. Connect to the AWS instance using SSH.

1. Navigate to the directory containing `java-program.jar`.

1. Run the Java program with the specified input and output folders:

   java -jar java-program.jar

1. The program will automatically convert all supported file types in the input folder and save the CSV files in the output folder.

By following these steps, you can effectively use the Java program to convert various file types to CSV format, making them ready for further data analysis and processing.

**Supported File Types:**

The Java program supports the conversion of various file formats into CSV format. Below is a list of the supported file types along with their corresponding file extensions:

|  |  |
| --- | --- |
| **File Extension** | **File Type** |
| .xls, .xlsx | Excel |
| .json | JSON |
| .avro | Avro |
| .parquet | Parquet |

The program is capable of seamlessly converting these file formats into CSV files, allowing for standardised data processing and analysis through FluentD. This versatility ensures that a wide range of data sources can be utilised within the data processing pipeline.

**Configuration Options:**

The Java program provides flexibility through configuration options that can be customised to suit specific needs. These options are managed within the program's source code. Below are the available configuration options and instructions on how to modify them:

1. **File Paths and Folders:**

* **Source Folder:** This option defines the folder where the input files are located. By default, the program scans the source folder for files to convert.
* **Output Folder:** This option specifies the folder where the generated CSV files will be saved. The converted CSV files will be stored in this directory.

To modify these folder paths, navigate to the relevant utility classes within the program and locate the appropriate variables. Update the values to the desired folder paths.

1. **ClamAV Configuration:**

* **ClamAV Host:** The hostname or IP address of the ClamAV server for malware scanning.
* **ClamAV Port:** The port number to connect to the ClamAV server.
* **Connection and Read Timeouts:** Configure the timeouts for connecting to and reading from the ClamAV server.

To configure ClamAV settings, access the `ClamAVService` class in the program. Modify the corresponding variables with the desired values.

Please note that the program's configuration options are primarily managed through direct modifications in the source code. After making any changes, compile the program to apply the updated configurations.

It's recommended to have a backup of the original source code before making modifications to ensure that any unintended changes can be reverted easily.

Additionally, the provided `.conf` files for ClamAV should be uploaded and used without modification, as they are designed to work seamlessly with the Java program.

Please exercise caution while modifying the configuration options and ensure that you have a good understanding of the program's structure and functionality before making changes.

**Troubleshooting:**

During the usage of the Java program for file conversion, users may encounter various issues. Below are some common problems users might face and their corresponding troubleshooting tips and solutions:

1. **Issue: ClamAV Service Not Running**

* **Symptoms:** The program might fail to run or produce errors if the ClamAV service is not running.
* **Solution:** Ensure that ClamAV is up and running on the AWS instance. To make sure ClamAV starts on boot, follow the provided steps to configure it to run as a service. Use the command line to start ClamAV manually if needed.

1. **Issue: Incorrect File Paths**

* **Symptoms:** The program may not locate the source files or output folder, leading to errors or incomplete conversions.
* **Solution:** Double-check the specified file paths for the source folder and output folder in the program's source code. Ensure they point to valid and accessible directories.

1. **Issue: Unsupported File Formats**

* **Symptoms:** The program might encounter errors or unexpected behaviour if it attempts to convert unsupported file formats.
* **Solution:** Refer to the list of supported file formats in the program's documentation. Make sure you're providing input files with the correct extensions that match the supported formats.

1. **Issue: Malware Detection Failures**

* **Symptoms:** The program might not be able to perform malware scans correctly, resulting in unreliable results.
* **Solution:** Ensure that ClamAV is properly configured and running. Verify the host and port settings in the program match the ClamAV server's details. Also, make sure that the ClamAV database is up to date.

1. **Issue: Program Crashes or Errors**

* **Symptoms:** The program might crash or display errors during execution.
* **Solution:** Check the program's logs for error messages or stack traces. If possible, isolate the problematic scenario and try to replicate it on a smaller scale. If the issue persists, consider reviewing the program's source code or seeking assistance from technical support.

1. **Issue: Performance and Memory Problems**

* **Symptoms:** Large files or excessive memory usage might lead to performance issues or crashes.
* **Solution:** Optimise the AWS instance's resources (CPU, memory) based on the size and complexity of the input files. Monitor system resources during execution and adjust as needed.

1. **Issue: Dependency or Library Errors**

* **Symptoms:** Dependency-related errors might occur if required libraries are missing or not properly configured.
* **Solution:** Make sure all necessary libraries and dependencies are correctly included and configured. Check the program's documentation for any prerequisites or required packages.

If you encounter issues that are not covered here, refer to the program's logs and error messages for more specific information. If needed, consult the program's documentation or seek assistance from technical support or online communities.

**Performance and Limitations:**

While the Java program offers valuable functionality for file conversion, it's important to be aware of its performance considerations and limitations, especially when hosted on an AWS micro instance:

1. **Resource Constraints:**

* AWS Micro instances have limited CPU and memory resources. Processing large files or a high number of files concurrently may lead to performance degradation or even instance unresponsiveness.
* **Recommendation:** Monitor CPU and memory usage during program execution. Adjust the workload to match the instance's capabilities, and consider using larger instances for resource-intensive tasks.

1. **File Size:**

* Large input files may strain the AWS micro instance's memory and processing power, leading to slower execution or out-of-memory errors.
* **Recommendation:** Avoid processing extremely large files on a micro instance. If required, consider segmenting or preprocessing large files before conversion.

1. **Concurrency:**

* AWS micro instances have limited concurrency capacity. Running multiple instances of the program simultaneously might cause contention for resources.
* **Recommendation:** Limit the number of concurrent program instances running on a micro instance to prevent resource contention.

1. **Network Latency:**

* AWS micro instances might experience network latency, especially if hosted in certain regions or availability zones.
* **Recommendation:** Be prepared for potential network delays when accessing external resources, such as ClamAV services or data storage.

1. **Malware Scanning Overhead:**

* Malware scanning adds an additional layer of processing during file conversion, impacting overall execution time.
* **Recommendation:** Optimise malware scanning by maintaining up-to-date ClamAV databases and avoiding scanning unnecessary files.

1. **Disk Space:**

* Micro instances come with limited storage capacity, which might be a concern when dealing with a significant number of large files.
* **Recommendation:** Regularly monitor disk usage to ensure sufficient space for input, output, and temporary files.

1. **Java Heap Size:**

* The Java program's heap size affects memory usage. A smaller heap size may lead to memory-related errors, while a larger heap size might exhaust the instance's available memory.
* **Recommendation:** Adjust the Java heap size based on the available memory. Experiment with different heap sizes to find a balance between performance and resource usage.

1. **File Format Complexity:**

* Complex file formats, such as Parquet and Avro, might require more resources to convert compared to simpler formats like Excel or JSON.
* **Recommendation:** Evaluate the complexity of the input files and consider the instance's capabilities when choosing the appropriate file format for intermediate conversions.

In summary, the Java program's performance on an AWS micro instance is subject to several limitations related to CPU, memory, and network resources. It's crucial to optimise your workflows and manage resources effectively to ensure smooth execution and avoid performance bottlenecks. If your requirements exceed the micro instance's capabilities, consider using more powerful instances or distributed processing solutions.

**Logging and Error Handling:**

The Java program employs a comprehensive logging mechanism to track its execution and handle errors and exceptions gracefully. The program uses the built-in Java logging framework to provide detailed insights into its behaviour and to assist in diagnosing issues. Here's how the logging and error handling mechanisms work:

1. **Logging Levels:**

* The program utilises different logging levels (e.g., INFO, WARNING, SEVERE) to categorise log messages based on their significance.
* **Example from the Code:**

A screen shot of a computer program

Description automatically generated

1. **Log Output:**

* Log messages are written to the console by default, but they can also be configured to write to files or other destinations.
* **Example from the Code:**

A screen shot of a computer code

Description automatically generated

1. **Error Handling:**

* The program employs try-catch blocks to catch exceptions that may occur during the conversion process.
* When an exception is caught, relevant error messages are logged, providing context and details about the error.
* **Example from the Code:**

A computer screen with text on it

Description automatically generated

1. **Custom Exception Handling:**

* The program may define custom exceptions to handle specific scenarios and provide clear error messages to users.
* **Example from the Code:**

A computer screen shot of a program code

Description automatically generated

1. **Contextual Information:**

* Log messages may include contextual information, such as the current step of the conversion process or the name of the file being processed.
* **Example from the Code:**

A black background with white text

Description automatically generated

1. **Detailed Stack Traces:**

* When exceptions are caught, their stack traces are logged to provide developers with precise information about the origin of the error.
* **Example from the Code:**

A screen shot of a computer

Description automatically generated

In summary, the Java program's logging and error handling mechanisms work together to ensure that any issues or exceptions encountered during the conversion process are properly documented and communicated. The use of logging levels, custom exceptions, contextual information, and detailed stack traces contributes to effective debugging and troubleshooting.

**Dependencies and Licences:**

The Java program utilises several third-party libraries to enhance its functionality. Below is a list of these libraries along with their respective licences:

1. **JUnit Jupiter**

   - Version: 5.10.0

   - Licence: Eclipse Public License - v 2.0

   - Website: https://junit.org/junit5/

2. **Apache POI - Core**

   - Version: 5.2.3

   - Licence: Apache License 2.0

   - Website: https://poi.apache.org/

3. **Apache POI - OOXML**

   - Version: 5.2.3

   - Licence: Apache License 2.0

   - Website: https://poi.apache.org/

4. **LOG4j Core**

   - Version: 2.20.0

   - Licence: Apache License 2.0

   - Website: https://logging.apache.org/log4j/2.x/

5. **Jackson JSON Library**

   - Version: 2.15.0

   - Licence: Apache License 2.0

   - Website: https://github.com/FasterXML/jackson-databind

6. **OpenCSV Library**

   - Version: 5.7.1

   - Licence: Apache License 2.0

   - Website: http://opencsv.sourceforge.net/

7. **Apache Avro**

   - Version: 1.11.2

   - Licence: Apache License 2.0

   - Website: https://avro.apache.org/

8. **SLF4J API**

   - Version: 2.0.7

   - Licence: MIT License

   - Website: http://www.slf4j.org/

9. **SLF4J Binding with Logback Implementation**

   - Version: 1.4.8

   - Licence: Eclipse Public License - v 1.0 / LGPL 2.1

   - Website: http://logback.qos.ch/

10. **Apache Parquet**

    - Version: 1.13.1

    - Licence: Apache License 2.0

    - Website: https://parquet.apache.org/

11. **Apache Hadoop**

    - Version: 3.3.6

    - Licence: Apache License 2.0

    - Website: https://hadoop.apache.org/

12. **jdom2**

    - Version: 2.0.6.1

    - Licence: Apache License 2.0

    - Website: https://github.com/hunterhacker/jdom

13. **ODF Toolkit**

    - Version: 0.9.0

    - Licence: Apache License 2.0

    - Website: https://incubator.apache.org/odftoolkit/

14. **iText Core**

    - Version: 5.5.13.3

    - Licence: Affero General Public License (AGPL)

    - Website: https://github.com/itext/itextpdf

15. **HTTP Components**

    - Version: 4.5.14

    - Licence: Apache License 2.0

    - Website: https://hc.apache.org/