MetaCDP: Metamorphic Testing For Quality Assurance of Containerized Data Pipelines

Faqeer ur Rehman, Sidrah Umbreen, Mudasser Rehman

MetaCDP: Metamorphic Testing For Quality Assurance of Containerized Data Pipelines

Faqeer ur Rehman
Montana State University
Bozeman, USA
faqeer.rehman@student.montana.edu

Sidrah Umbreen
Montana State University
Bozeman, USA
sidrah.umbreen@student.montana.edu

Mudasser Rehman
Uzair Technology
Kohat, Pakistan
mudasseruzr@gmail.com

***Abstract*—In the ever-evolving world of technology, companies are investing heavily in building and deploying state-of-the-art Machine Learning (ML) based systems. However, such systems heavily rely on the availability of high-quality data, which is often prepared/generated by the Extract Transform Load (ETL) data pipelines; thus, they are critical components of an end-to-end ML system. A low-performing model (trained on buggy data) running in a production environment can cause both financial and reputational losses for the organization. Therefore, it is of paramount significance to perform the quality assurance of underlying data pipelines from multiple perspectives. However, the computational complexity, continuous change in data, and the integration of multiple components make it challenging to test them effectively, ultimately causing such solutions to suffer from the Oracle problem. In this research paper, we propose MetaCDP, a Metamorphic Testing approach that can be used by both researchers and practitioners for quality assurance of modern Containerized Data Pipelines. We propose 10 Metamorphic Relations (MRs) that target the robustness and correctness of the data pipeline under test, which plays a crucial role in providing high-quality data for developing a clustering-based anomaly detection model. To show the applicability of the proposed approach, we tested a data pipeline (from the E-commerce domain) and uncovered several erroneous behaviors. We also present the nature of issues identified by the proposed MRs, which can better help/guide software engineers and researchers to use best coding practices for maintaining and improving the quality of their data pipelines.**

***Index Terms*—Testing ETL pipelines, Metamorphic Testing, Quality Assurance of Docker Containers, Oracle problem in ETL pipelines, MiniKube, Testing Data Engineering Applications, Testing Data Pipelines**

I. Introduction

Extract Transform Load (ETL) is a process (as shown in Figure 1) used for *extracting* the data from different data sources (i.e., relational/non-relational databases, XML/CSV/text files, and blob storage such as Amazon S3 object storage, Azure blob storage, Google storage etc.), performing the needed *transformation* to convert the data into the desirable format, and then *loading* it to its final destination (i.e., database, data warehouse, and data lake). The data in the destination system can be used by business analytical solutions and for training Machine Learning (ML) models to help organizations make well-informed decisions and drive business value.

In today's world, data is as valuable as oil, and the ETL data pipelines are the factories/refineries that process/refine such a valuable asset. Just like a data integrity issue in an oil refinery can lead to significant losses for the company, a small bug in the data pipeline can generate both financial and reputational loss for an organization. According to Gartner, organizations believe that almost $15 million (on average) per year in losses can be attributed to using low-quality data [1]. Imagine feeding incorrect/buggy data to a system that can cause i) a machine learning model to wrongly forecast a high demand for a less popular product, which ultimately can lead to significant financial loss for the company, ii) a deep learning model to wrongly predict that an incoming network request is not malicious, whereas, in reality, it is; thus, may cause the company to lose all its top secret business data, and iii) a data analytical solution to report incorrect annual financial results of the company, which can seriously hamper its reputation across the globe. All these buggy behaviors in a critical system can not only generate a million-dollar loss for the company but can also expose it to legal action(s).

In software testing, software's 'correctness' attribute (i.e., its ability to meet specification) is verified by comparing its output to an expected output, also known as *test oracle*. The *Oracle problem* is encountered when either the test oracle is not known beforehand or is theoretically possible to identify but practically too difficult/expensive to apply [2]. Considering the critical nature of ETL pipelines for generating high-quality data to build reliable and intelligent solutions, performing their quality assurance from multiple perspectives is of paramount significance. However, testing of data pipelines suffers from the oracle problem. This is because the integration of multiple components, complex computations performed in them, and difficulty in assessing their outputs (which can involve large volumes of continuously changing data) make it difficult for software testers to verify whether the data extracted, processed, and loaded by them is accurate or not.

Metamorphic Testing (MT) has been widely used in addressing the oracle problem in a diverse range of applications, i.e., code obfuscators [3], navigational software (i.e., Google Maps) [4], machine translation software [5], anomaly detection [6], and self-driving vehicles [7]. The MT technique verifies the program's correctness over multiple executions and checks whether the program under test adheres to the necessary properties/characteristics (known as *Metamorphic Relations*) expected from the program under test [8]. A violation of

2024 IEEE Cloud Summit | 979-8-3503-7006-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CLOUD-SUMMIT61220.2024.00029

979-8-3503-7006-5/24/$31.00 ©2024 IEEE
DOI 10.1109/Cloud-Summit61220.2024.00029

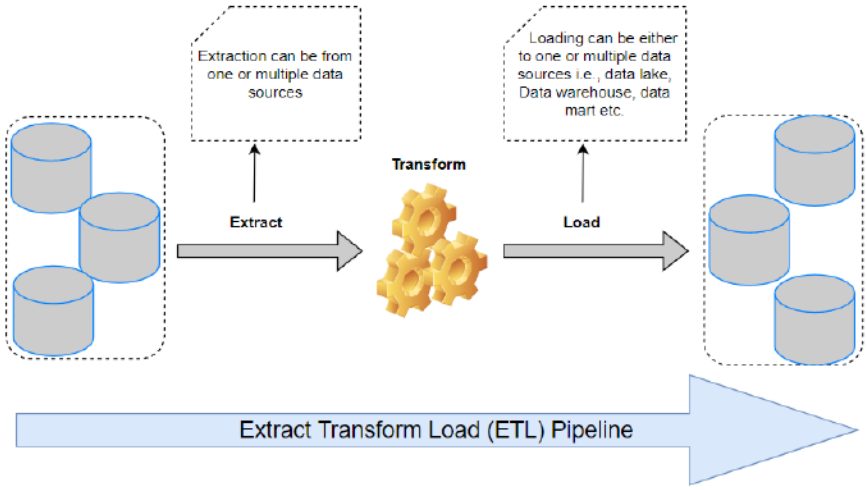

Fig. 1. ETL Process/Pipeline

metamorphic relation is an indication of a potential bug in the program under test.

In this paper, we propose MetaCDP, a Metamorphic Testing (MT) approach that can be used by both researchers and practitioners for quality assurance of Containerized Data Pipelines. We further evaluate our proposed approach to test a containerized ETL data pipeline that processes and generates data within the E-commerce business. The data generated by this pipeline is subsequently used to train a clustering-based anomaly detection model that identifies customers with abnormal sales and purchase patterns. However, in this research study, our focus is only on the testing aspect of the containerized data pipeline, not on the machine learning model.

In summary, we make the following contributions in this paper:

- To the best of our knowledge, this is the first research that focuses on identifying implementation bugs in containerized ETL jobs that suffer from the oracle problem.
- We propose an MT-based approach for performing the quality assurance of data pipelines.
- We propose 10 Metamorphic Relations (MRs) that can be used by both researchers and software testers to verify and validate the correctness of their data pipelines.
- We further demonstrate the effectiveness of the proposed approach in testing an ETL data pipeline in the E-commerce domain. The results obtained show that the proposed MRs not only uncover several implementation bugs in the data pipeline under test but also highlight scalability issues and suggest avoiding poor coding practices.

It is important to highlight that the proposed testing strategy is not just limited to testing ETL pipelines within the E-commerce domain; instead, it can be used for testing ETL pipelines in multiple domains, i.e., education, finance, supply chain, manufacturing, marketing, etc., provided that they are dealing with structured data because some of the proposed MRs uses this characteristic of data to target the necessary characteristics of the data pipeline under test. We don't claim the generalization of its applicability to unstructured data (which is not focused in this study) that is equally a threat to its external validity, also mentioned in the 'threats to validity' section.

The rest of the paper is organized as follows. Section II presents a literature survey, whereas Section III highlights the difficulties encountered in testing ETL data pipelines. Section IV presents some background knowledge along with the proposed MT-based approach for testing the ETL pipeline under test. Section V discusses the results obtained. Section VI mentions threats to validity, and finally, Section VI concludes with potential future work.

II. Literature Survey

Using a container-based virtualization tool like Docker [15] has several benefits. First, it allows all the dependencies of the application to be packaged in a single container. Second, it allows scaling an application to handle increasing workloads and better utilization of hardware resources. Third, it provides better security by separating each container running in an isolated environment. Fourth, it makes an application portable and can run on any platform of your choice. Last but not least, it can make applications available by running multiple replicas; if one fails, the other replica(s) can handle the incoming requests/load.

When the application is deployed as a container, it behaves consistently in any environment as long as the Docker daemon is running on the host machine. This helps to avoid operational issues in the production environment, such as when the application works fine on the developer's machine but crashes in the production environment due to differences in the environment. Due to the widespread usage of dockerized containers in the

cloud, it is crucial to have effective testing strategies that can better test containerized jobs from multiple perspectives.

This work proposes an MT-based approach for testing containerized ETL data pipelines used in the E-commerce domain. In our case, the quality of data generated by the data pipeline under test is essential for building a robust clustering-based anomaly detection model. The data pipeline under test consists of multiple components (deals with sales, purchase, inventory, and payment data) that work together to generate quality data for building an efficient ML model that can cluster customers based on their sales and purchase behavior. It is important to note that this study focuses solely on testing the containerized data pipeline, not the clustering-based anomaly detection ML model. For readers interested in knowing more about using MT for testing clustering-based models, we recommend reading [12] [13] .

Preeth et al. [18], performed a performance evaluation of docker containers (using different benchmarking tools) based on various factors i.e., disk usage, CPU and memory utilization, network I/O statistics, boot time, etc., and suggested that docker containers performance is comparable to the OS running on bare-metal. Kavitha et al. [21] performed a performance comparison between a Virtual Machine (VM) and a docker container on an AWS cloud service provider and concluded that the docker container outperforms the VM on the metrics used, i.e., average response time and throughput. Sergeev et al. [20] conducted stress testing to observe container behavior under increased load. They also performed volume testing to evaluate application response to significantly increased data input. Xu et al. [22] work is focused on evaluating whether docker containers have any impact on the performance of deep learning-based models. They containerized three deep neural network-based models, i.e., Fully Connected Networks, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), and observed very little overhead when performing compute-intensive tasks, thus suggesting that containerizing deep learning-based models is a feasible solution. Similarly, Di Tommaso et al. [23] analyzed the performance overhead of docker containers on genomic pipelines and concluded that their impact on executing mid/long-running computational jobs is negligible and is offset by the large benefits provided by the containerized environment.

To the best of our knowledge, we haven't found any research study that addresses the oracle problem in testing containerized data pipelines using the MT technique, which is equally a motivator for this research work.

III. Challenges In Testing ETL Pipelines

Due to the critical nature of ETL pipelines, it is very important to test them to verify their correctness. However, there are certain challenges faced by software developers and testers to test such solutions effectively. First, the integration of multiple components in the ETL pipeline and the complexity involved in the computation performed by each component makes it a hard task to perform their automated testing. Second, continuous changes in input data and the large volume of data itself are another reason that hinders their effective testing with limited resources. Last but not least, the lack of availability of expected output to verify the correctness for a large volume of input data put these systems into the category of non-testable systems [19].

Suppose an ETL pipeline is comprised of multiple components to perform the needed task. These components coordinate with each other (in which the output of one component is used as an input to the next component) to collect network security logs (in raw form) from multiple network nodes, process them, and then load them into the destination data source. So, from a quality assurance perspective, a pivotal question arises: *how to verify if the data extracted, processed, and loaded into the final destination source is correct, especially when the data/logs generated on network nodes continuously change*. In such environments, knowing the expected output beforehand becomes difficult, which ultimately causes such systems to suffer from the oracle problem and puts them into the category of non-testable systems.

Several research studies have shown that this problem is very common in testing a range of applications i.e., code obfuscators [3], navigational software [4], machine translation software [5], self-driving vehicles [7] and restful web APIs [17], and showed that how MT can be used in testing and addressing oracle problem in them.

It is crucial to note that while documentation and design documents may be available for software testers, testing complex data pipelines from various perspectives can be a daunting task, especially when input data continually changes, resulting in different outputs. In this research, we demonstrate how MT can be effectively utilized to test the behavior of data pipelines from multiple angles.

IV. Proposed Metamorphic Testing (MT) Approach

The MT technique, a property-based testing technique [8], is a well-known software testing technique used for addressing the oracle problem in a range of applications [9] [10] [11] [14]. In this testing technique, the relations are captured, known as *Metamorphic Relations (MRs)* that test the necessary characteristics/behavior expected from the program under test. Each MR verifies the program's output over multiple executions, even if the output for the individual input is unknown. As shown in Figure 2, each MR is comprised of the source input (original test case) and the follow-up input (transformed test case) that targets the necessary properties of the program under test. Both the source and follow-up inputs are fed to the program to obtain the outputs. The outputs are then compared and checked whether the relation (that captures the expected behavior of the program under test) is satisfied or not. The violation of MR is an indication of a potential bug in the program under test.

To better understand the concept, consider a program used to find a distance between any two input data points i,e, (a,b) in a given space. To test it, an MR can be proposed that says, 'If the order of input data points is changed, i.e., (b,a), the program output should still remain the same.' In other words,

the program output should not depend on the order of input data points. If the output for the source input, i.e., (a,b), differs from the output obtained for the follow-input, i.e., (b,a), it will indicate a potential bug in the program under test.

In this paper, we propose an MT-based approach, 'MetaCDP' to perform the quality assurance of the Containerized Data Pipeline that deals with generating and processing E-commerce data. After performing a close analysis, we identify that the two primary data sources consumed by its containerized components are CSV files and a relational database (Microsoft SQL Server). Considering these two data sources and the containerized nature of ETL components, we propose 10 MRs to target the necessary characteristics expected from the pipeline under test, which are as follows.

A. MRModifySequence

This MR is named as *MRModifySequence*. Its purpose is to ensure that the data pipeline being tested is not affected by the position of columns in the data. If the pipeline is sensitive to column position, it could result in higher maintenance costs in the future, as the position of columns can change without modifying the actual data.

MRModifySequence 1.1: If the sequence of columns in a CSV data file is changed, this transformation should not impact changing the program output.

More specifically, let $f(a, b, c)$ be the function/component in the data pipeline that performs the required transformation on the columns a, b, c and generates some output. *MRModifySequence* states that if we change the order of columns and provide to the same function/component, i.e., $f(b, a, c)$, it should not cause the program to produce some different output.

MRModifySequence 1.2: If the sequence of columns in a SQL query (inside a stored procedure, function, or view) used to insert/fetch data from the database is changed, the output of the program should remain consistent.

B. MRRemoval

This MR is named as *MRRemoval*. It aims to ensure that only the needed data for a particular job or task in the data pipeline is provided to it. This is important because providing unnecessary data can lead to security issues and waste network bandwidth resources, especially in an environment with limited computing resources.

MRRemoval 2.1: If a particular column in a CSV data file is deleted, it is expected to either modify the program output or result in a program crash. This is because, according to this MR, all the columns in CSV data files are expected to be used by the containerized job for processing. A column should not exist in the file if it is deemed irrelevant. More specifically, *MRRemoval* states that

$f(a, b, c) \neq f_R(a, b, c)$

where $f_R(a, b, c)$ is a function that randomly removes a column from a CSV data source.

MRRemoval 2.2: Similar to MRRemoval 2.1, if a specific column from a SQL query (inside a stored procedure, function, or view) used to insert/fetch data is deleted, this change should either change the program output or cause the program to crash.

MRRemoval can be used to assess the program's ability to either crash or generate unexpected output when a feature is removed from its data source. If the program generates the same output for the follow-up input, it will suggest that the feature is irrelevant and can be safely removed to prevent the wastage of network bandwidth resources.

C. MRAddition

This MR is named as *MRAddition*. The purpose of this MR is to check the data pipeline's robustness when noise/irrelevant information is added to the underlying data. In a real-world scenario, when many system components intercommunicate, such noise is inevitable, and the system should have proper guardrails configured to handle such scenarios.

MRAddition 3.1: If a new (dummy) column is added to the CSV file, it should not cause the program to produce inconsistent output. Apart from that, a reliable ETL pipeline must be capable of identifying any unexpected/unconfigured newly added column(s) and issuing appropriate warnings or alerts.

MRAddition 3.2: If a new column (with some dummy value) is added to the SQL query (inside a stored procedure, function, or view), it should not cause the program to result in an inconsistent output.

D. MRNetworkFailure

This MR is named as *MRNetworkFailure*. One of the characteristics of containerized applications is that they must be designed to be resilient and fault-tolerant to intermittent network failures. In a cloud environment, it is common to experience intermittent network failures, which may cause a temporary service disruption. To handle such unexpected situations, a reliable data pipeline should have a retry mechanism in place, which can retry after a few seconds or minutes to make the system resilient. This MR 'MRNetworkFailure' is proposed to simulate and test such scenarios/characteristics. When a containerized job is executed (which fetches data from an API), this MR is used to take the API down for a few seconds. The system is expected to be flexible enough to handle such intermittent failures and produce consistent output.

E. MRDBConnectivity

This MR is named as *MRDBConnectivity*. The purpose of this MR is to ensure that the Database (DB) connectivity feature (in the containerized jobs) is functioning correctly. It is necessary to verify that the correct database is connected to execute the required SQL statements. In the tested ETL data pipeline, the containerized jobs/tasks require SQL Server database credentials to be provided as environment variables. This approach helps to keep sensitive information, such as database credentials, away from malicious users.

The proposed MR suggests that if incorrect DB information is provided to the containerized job, it should not be able to

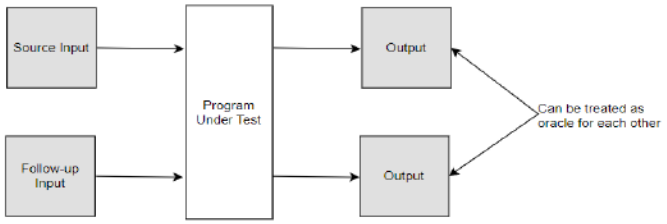

Fig. 2. Metamorphic Testing Technique

connect with it. Instead, it should fail at all points where it attempts to execute the SQL queries on the given data source. This MR can also be useful in an environment where the ETL job is required to access the database through the firewall. If the job is executed outside the firewall settings and it still connects to the database successfully, it would violate the proposed MR, indicating that the pipeline under test does not adhere to the intended security characteristics. Since we do not have such a requirement in the data pipeline under test, we have used this MR only to test the DB connectivity feature by providing an incorrect DB server name and port to the containerized jobs.

F. MRReplicate

This MR is named as *MRReplicate*, which aims to test the scalability aspect of ETL jobs under test. This MR states that no matter how many replicas (within some certain threshold) of the containerized job are created, they should all produce the same output for a given input.

G. MRUpgradeMountPath

This MR is named as *MRUpgradeMountPath*. This MR is designed based on the premise that data pipeline components often mount network-shared directories into the containers to store and share data among them. However, in software engineering, frequent code refactoring cycles are common and can lead to changes in these mount paths. A well-designed data pipeline should be flexible, easy to maintain, and should adapt to these changes, i.e., the target mount path (a location in the container) may change from `/app/InputFiles` to `/app/RawFiles`. To test whether the pipeline under test adheres to such characteristics, we propose *'MRUpgradeMountPath'* which states that 'if we mount the shared network drive to a different target path in the containerized job, it should not affect the pipeline to produce inconsistent output'.

V. Experimental Results

The data pipeline under test belongs to a retail business domain and is composed of three containerized jobs that process sales, purchases, and inventory data. The first job in the pipeline loads sales, purchase, and inventory data from CSV files and the Microsoft SQL Server database. The next job in the pipeline performs further processing to transform the data into the desired format, i.e., performing data cleaning, filtering the required features, data validation, data aggregation, and transformations such as monthly, quarterly, and yearly sale/purchase averages. Finally, the last containerized job loads this processed/transformed data into the data warehouse, which is then consumed by data analytics applications and building a clustering-based anomaly detection model. The containerized jobs are tested locally on MiniKube [16] (running on Windows 11), a tool that enables running a Kubernetes cluster on a single node and can be used for development and testing purposes. We highlight the issues/faults identified by each MR in the data pipeline under test as follows.

```
SELECT cust.customerID,
 CASE
     WHEN cust.isActive = 1 THEN 'Active'
  ELSE
     'Not Active'
 END As Status,
pd.productID, pd. productName,
sl.quantity, sl.soldDate
FROM tblCustomers cust
INNER JOIN SALES sl
ON cust.customerID = sl.customerID
INNER JOIN PRODUCTS pd
ON sl.productID = pd.productID
WHERE cust.isActive=1
AND pd.isActive=1
```

Fig. 3. SQL Query To Pull Sales Data

A. Issues Identified By MRModifySequence

Figure 3 presents an example of SQL query used for pulling sales data from the database. When this MR is used to change the sequence of columns (both in CSV files and the SQL queries), it causes the data pipeline to crash in two places. Upon further debugging, it was discovered that the issue was in a code block that was extracting data features based on their position in the dataset. Such coding practice is not recommended as it assumes that the order of columns in the dataset will never change. As a result, the data pipeline is less robust to future changes, which is not ideal given the ever-evolving nature of software. A visual representation of the issue can be seen in Figure 4, in which the code block uses an item index to extract the data.

```
def replaceNullWithZeros(self):
    for index, row in self.unprocessedSalesDF.iterrows():
        storeID = row.iloc[0]  # storeID
        productID = row.iloc[1]  # productID
        date = row.iloc[2]  # date
        inventory = row.iloc[3]  # inventory

        # replace inventory with 0 if it is null
        if pd.isnull(inventory) or inventory<0:
            self.unprocessedSalesDF.iloc[index, 3] = 0
```

Fig. 4. MRModifySequence

```
SELECT
 cust.customerID,cust.Name,
 cust.Address,sl.ID, sl.quantity,
 sl.quantity*sl.unitprice as total,
 sl.billFile, sl.isLoan
 sl.soldDate
FROM tblCustomers cust
INNER JOIN SALES sl
ON cust.customerID = sl.customerID
ON sl.productID = pd.productID
WHERE cust.isActive=1
AND sl.soldDate > getdate()-30
```

Fig. 5. MRRemoval

B. Issues Identified By MRRemoval

This MR did not identify any implementation bugs in the data pipeline under test, but it did reveal some performance issues. As shown in Figure. 5, we expect the program to produce a different output after removing the 'billFile' column (which stores a bill of sale document as a binary object) from the SQL query. However, it didn't, indicating that the code doesn't use this column to perform the required business logic. Therefore, this column can be safely removed, preventing the waste of resources, such as network bandwidth and memory, and further enhancing the system's performance.

We conducted a small experiment to quantify the impact of using this MR in such scenarios. Based on our analysis, we found that the original query (in Figure 5) took about 5 seconds (on average) to retrieve sales data for the last 30 days. However, after identifying and removing 'billFile', the query performance improved significantly, and the time was reduced to about 3 seconds on average. The difference may look small, but such minor improvements can significantly impact overall system performance, especially when such queries are run on a large scale daily, thus preventing the system resources from being wasted.

C. Issues Identified By MRAddition

After adding a new column to both the CSV files and the SQL queries, we executed 'MRAddition' but found no unintended program behavior. The output remained consistent for both the source and follow-up input executions. It is important to note that while this MR did not reveal any implementation bugs in the data pipeline under test, it should not be treated useless. Instead, we recommend using this MR to potentially uncover implementation bugs in code sections that are sensitive to such changes in data sources.

D. Issues Identified By MRNetworkFailure

As mentioned earlier, during the execution of containerized jobs in the data pipeline being tested, we intentionally bring down the API for a few seconds to simulate intermittent network failures. This approach helps us identify any inconsistencies or bugs in the pipeline that may arise when encountered by such scenarios. On applying this MR, we found that none of the data pipeline jobs could handle such intermittent network failures and started crashing. The current testing revealed that the pipeline lacks a retry mechanism and, if present, could have gracefully recovered from these intermittent network failures.

E. Issues Identified By MRDBConnectivity

As per the relation captured in this MR, we provided incorrect server name and port information (as environment variables) to the data pipeline jobs. This was done to ensure that none of the SQL queries would execute successfully. However, we noticed that one of the queries successfully fetched the data. Upon further investigation, we discovered that a certain code section used hard-coded server information instead of using the class variables (defined to store the database server and port information injected through environment variables). After further communication with the developer, it was discovered that hard-coded information had been used for internal testing and was mistakenly left uncommented before the code was released. It is important to note that such issues might not be detected without using this type of MR, highlighting the significance of using MT to uncover such critical bugs in the program under test.

F. Issues Identified By MRReplicate

To ensure that the containerized jobs running in a cloud environment can handle peak loads, they must be scalable. To test such capability of the data pipeline jobs, we scale up the deployment by updating the '.spec.replicas' configuration in the deployment.yaml file. Our expectation was that the replicated jobs would generate the same output regardless of the number of replicas created to process the given sample data. However, we noticed that when we increased the replicas from 1 to 5, the replicated jobs started throwing errors. This failure occurred because all the replicas were writing logs (to be used for monitoring and debugging purposes) to the same file in a shared location. This caused the replicated jobs to fail when they tried accessing the same log file simultaneously. The violation of this MR suggests that the scalability factor must be taken into then consideration when designing cloud-native applications for handling load at peak times.

G. Issues Identified By MRUpgradeMountPath

The aim of this MR is to detect similar issues as uncovered by the 'MRDBConnectivity', where a software engineer/data

scientist might hard-code the target mount path (a location in the container) in the code instead of utilizing the provided information (through environment variable) stored in class variables. Although this MR did not reveal any unexpected behavior in the data pipeline being tested, we still recommend using this MR in combination with other MRs to further enhance trust in the data pipelines before deploying them to the production environment.

VI. Threats To Validity

Internal Validity: Someone may argue that the MRs that didn't detect any issues in the data pipeline being tested are not useful, thus can be excluded from the MRs repository. However, this is not necessarily true. We argue that these MRs are still useful for testing containerized data pipelines for two reasons. First, they are applicable to testing ETL data pipelines. Second, just because they did not detect any inconsistencies does not necessarily mean that all programs will have a similar implementation. Some programs may be sensitive to the transformations captured in those MRs, and thus will detect inconsistencies.

External Validity: The proposed MT-based approach for testing data pipeline under test (from the E-commerce domain) can be extended to test ETL pipelines in other domains as well, i.e., finance, marketing, manufacturing, cybersecurity, etc., as long as they deal with the structured data and are containerized (because the proposed method uses such knowledge to test the data pipeline under test). Therefore, we don't claim the generalization of the proposed approach to test the ETL pipeline dealing with unstructured data. Similarly, we don't claim that the proposed list of MRs is exhaustive and captures every aspect of testing data pipelines. Both of these observations require further empirical evidence that we aim to investigate in our future work, which is under progress.

Construct Validity: Construct validity may occur due to the measurements used to capture the results. In our case, we have used the fault detection ability of the MRs, a well-established measurement for assessing the effectiveness of testing strategies.

VII. Conclusion

The quality of data processed/generated by ETL data pipelines plays a significant role in developing high-performance machine learning models and making critical business decisions, so their testing is paramount to ensure that they are robust and reliable. However, the integration of multiple components, the large volume of data, and the continuous change in input data put these systems to suffer from the oracle problem. To address this problem, we have proposed a Metamorphic Testing approach for testing containerized data pipelines. We have proposed 10 MRs that software engineers/testers can use to perform the quality assurance of ETL pipelines. Furthermore, we conducted a case study to demonstrate the effectiveness of the suggested MRs in testing a data pipeline that is used in the e-commerce industry. The nature of the issues they identify can help researchers and software engineering teams set code quality standards that can further help better maintain the data pipelines in the long run. Our future plans include two main objectives. Firstly, we aim to propose new MRs to capture the necessary characteristics expected from data pipelines in diverse domains. Secondly, we plan to demonstrate the usefulness of the proposed approach in testing containerized data pipelines that deal with unstructured data.

References

[1] https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement

[2] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2014). The oracle problem in software testing: A survey. IEEE transactions on software engineering, 41(5), 507-525.

[3] Chen, T. Y., Kuo, F. C., Ma, W., Susilo, W., Towey, D., Voas, J., & Zhou, Z. Q. (2016). Metamorphic testing for cybersecurity. Computer, 49(6), 48-55.

[4] Brown, J., Zhou, Z. Q., & Chow, Y. W. (2018). Metamorphic testing of navigation software: A pilot study with Google Maps.

[5] Sun, L., & Zhou, Z. Q. (2018, November). Metamorphic testing for machine translations: MT4MT. In 2018 25th Australasian Software Engineering Conference (ASWEC) (pp. 96-100). IEEE.

[6] Rehman, F. U., & Izurieta, C. (2022, August). An Approach For Verifying And Validating Clustering Based Anomaly Detection Systems Using Metamorphic Testing. In 2022 IEEE International Conference On Artificial Intelligence Testing (AITest) (pp. 12-18). IEEE.

[7] Zhou, Z. Q., & Sun, L. (2019). Metamorphic testing of driverless cars. Communications of the ACM, 62(3), 61-67.

[8] T. Y. Chen, S. C. Cheung, & S. M. Yiu (1998), Metamorphic testing: A new approach for generating next test cases, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01.

[9] Xiao, D., Liu, Z., Yuan, Y., Pang, Q., & Wang, S. (2022). Metamorphic testing of deep learning compilers. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 6(1), 1-28.

[10] Jiang, M., Chen, T. Y., & Wang, S. (2022). On the effectiveness of testing sentiment analysis systems with metamorphic testing. Information and Software Technology, 150, 106966.

[11] Chaleshtari, N. B., Pastore, F., Goknil, A., & Briand, L. C. (2023). Metamorphic Testing for Web System Security. IEEE Transactions on Software Engineering.

[12] Xie, X., Zhang, Z., Chen, T. Y., Liu, Y., Poon, P. L., Xu, B. (2020). METTLE: a METamorphic testing approach to assessing and validating unsupervised machine LEarning systems. IEEE Transactions on Reliability, 69(4), 1293-1322.

[13] Rehman, F. U., & Izurieta, C. (2022, June). MT4UML: Metamorphic Testing for Unsupervised Machine Learning. In 2022 9th Swiss Conference on Data Science (SDS) (pp. 26-32). IEEE.

[14] Tizpaz-Niari, S., Monjezi, V., Wagner, M., Darian, S., Reed, K., & Trivedi, A. (2023, May). Metamorphic testing and debugging of tax preparation software. In 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS) (pp. 138-149). IEEE.

[15] Docker (Webpage), [online] Available: https://docs.docker.com/.

[16] https://kubernetes.io/docs/tutorials/hello-minikube/

[17] Segura, S., Parejo, J. A., Troya, J., & Ruiz-Cortés, A. (2018, May). Metamorphic testing of RESTful web APIs. In Proceedings of the 40th International Conference on Software Engineering (pp. 882-882).

[18] Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In 2015 international conference on control communication & computing India (ICCC) (pp. 697-700). IEEE.

[19] Weyuker, E. J. (1982). On testing non-testable programs. The Computer Journal, 25(4), 465-470.

[20] Sergeev, A., Rezedinova, E., & Khakhina, A. (2022, September). Stress testing of Docker containers running on a Windows operating system. In Journal of Physics: Conference Series (Vol. 2339, No. 1, p. 012010). IOP Publishing.

[21] Kavitha, B., & Varalakshmi, P. (2018). Performance analysis of virtual machines and docker containers. In Data Science Analytics and Applications: First International Conference, DaSAA 2017, Chennai, India, January 4-6, 2017, Revised Selected Papers 1 (pp. 99-113). Springer Singapore.
[22] Xu, P., Shi, S., & Chu, X. (2017, August). Performance evaluation of deep learning tools in docker containers. In 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM) (pp. 395-403). IEEE.
[23] Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M. L., & Notredame, C. (2015). The impact of Docker containers on the performance of genomic pipelines. PeerJ, 3, e1273.