INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Non-Functional Testing Framework for Container-Based Applications

Shadab Alam Siddiqui[1]*, **Tamanna Siddiqui**[2]

1 Research Scholar, Department of Computer Science, Aligarh Muslim University, India
2 Professor, Department of Computer Science, Aligarh Muslim University, India

OPEN ACCESS

Received: 09.11.2021

Accepted: 13.12.2021

Published: 25.12.2021

Citation: Siddiqui SA, Siddiqui T (2021) Non-Functional Testing Framework for Container-Based Applications . Indian Journal of Science and Technology 14(47): 3433-3441. https://doi.org/10.17485/IJST/v14i47.1909

* **Corresponding author**.

siddiqui.shadab@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: To propose a generic framework that would help in defining a process and an eco-system which could be used to thoroughly test the non-functional behavior of any container-based applications. **Methods:** A comprehensive study was conducted to investigate the available frameworks for testing non-functional characteristics. This was followed up by quantitative analysis to know containers, non-functional aspects, generic framework, and its usage requirement. The step-by-step method was used to design the framework, where first the sub-parts were designed and implemented. It was followed up by defining the integration points of all the sub-parts. Finally, a fully matured, functional, and integrated framework came into effect. **Findings:** There are no frameworks available to focus on the non-functional aspect of container-based applications. The proposed framework can be utilized by various entities, viz developers, testers, integrators, and system or application certification authorities. It can be used to determine the nature and behavior of containerized applications under the known and unknown factors of non-functional characteristics, which transcends the mere realm of application behavior but primarily delves into the surrounding actors for the proper functioning of the application. **Novelty:** A unique testing framework for container applications focusing on non-functional characteristics.

Keywords: Framework; Capacity; Scalability; High Availability; NonFunctional Testing

1 Introduction

Applications that are built using a container-based framework are agnostic to the underlying hardware and the Operating system. Another area where containerization of application is happening in the transformation of existing monolithic applications to containerized form using the microservices-based architecture. The positives of creating or transforming an application using Container and microservices-based architecture are immense compared to virtualized architecture. The benefits are spread across a wide variety of parameters. The benefits and detailed comparison of containers versus virtual machines are discussed by Tamanna et al.[1].

Non-Functional testing is an integral part of any software or application. It reflects the application behavior based on the surrounding environment.

Non-functional testing evaluates the application go-live or production readiness as per non-functional characteristics or parameters that are not considered testing criteria in functional testing. In this paper, we have identified four Non-functional characteristics of container-based applications that are of prime importance. A brief description of the four identified Non-functional characteristics follows below:

1.1 Capacity

Capacity testing is conducted to numerically ascertain the functional transaction capacity of the application without jeopardizing and degrading its functional health with respect to the allocated resource. The by-product of capacity testing is that the numeric result or the traffic carrying capacity of an instance of the application is further used to design and define the scalability criteria of the application to be tested. This is general, is quite helpful in dimensioning the application vis-a-vis the projected traffic of the application in production.

1.2 Scalability

Scalability testing is used to quantify the threshold of resources, primarily CPU and memory, which can be used to decide for scaling up/down and in/out. This helps determine that the application can handle spurts of user traffic without affecting its core functionality. The output of capacity testing is very potent to decide Scalability numbers. Although capacity testing points out the full load-bearing capability of an instance, from a Stability point of view, it becomes empirical not to use the full capacity but rather define thresholds of the resources that could trigger the scale function. A sample scaling-out parameter would be 80% utilization of Memory or CPU, and similarly, a sample scale-in parameter would be 5% utilization of Memory or CPU.

1.3 Stability and Robustness

Stability and Robustness testing is carried out to determine that the application performs well within an acceptable time period and with variable traffic loads. The main aim is to identify the application's strength, stability, and robustness when it is exposed to heavy load and stress in different environments. One of the sample benchmarks to test the stability or robustness of the application is by executing an elongated test scenario; for example, if we are testing a calling application, the test case would be to test the application by using 72 hours call. And several such testing scenarios could be utilized to ascertain the stability and robustness of the container-based application. It also should be noted that while doing such type of testing, to be more effective, it should be done multiple times with changing environments like underlying hardware or the traffic scenarios.

1.4 High Availability

High Availability (HA) testing is required to test the application's response and preparedness to cater to the failures of the resources it utilizes for its execution. It is an essential testing that needs special attention when the application in test is critical in nature and cannot support any type of downtime. Application such as remote surgery or self-driving cars find their mention in critical applications. HA testing also includes the testing of Geo-redundant characteristics of the application, and it has been extensively worked upon by Siddiqui et al.(2). For cloud-based applications. In Geo-redundant testing, it is made sure that the backup of the application is geographically distant from the active application. This is done to ensure that the application is up and running and available to its users even when there are natural calamities or uncontrollable factors like power failure (including backup) or a fire within the premises hosting the application. In such scenarios, the geographically remote backup is made active, and the application is available to its users despite the unforeseen circumstances of the location of the original functional application.

2 Container Testing

Testing is an integral part of any application, and testing results are the deciding factors for an application's success. A pictorial depiction of testing any application is illustrated in figure 1 below. A comprehensive review of various software testing approaches for cloud applications has been done by Siddiqui et al.(3,4),but it is limited to cloud-based applications.

The evolution and resurgence of container technology have primarily been driven by industry adaptation and implementation rather than core academic research. Especially on the testing side of applications, the focus has been on having testing approaches and frameworks based on containerized platform rather than for container-based applications. Chuchuen et al.(5) have come up with a Container-based parallel system used for the Automation of software testing. They talk about the parallelization of selenium-robot-framework with four workers to improve performance.

For a container-based application, non-functional testing becomes of paramount importance because the application is unaware of the underlying hardware and the middleware. Patrizio[6], in his blogs, has listed out five primary reasons why testing of containers-based (cloud-native) applications is different from standard on-premise software of applications. It also must be noted that container-based applications are agnostic to the underlying hardware and OS. Thus, to minimize the effect of these unknowns, it becomes imperative to test such applications around their non-characteristics. This would help the application perform flawless execution and service to its users as the testing would have weeded out potential issues and problems which could have taken place when the application goes into production and where the surrounding eco-system of the application's host is unknown and uncontrollable by the application owner. This is also very important because issues faced in production are hard to mimic in the testing environments and thus are more costly to rectify, both in terms of time and money.

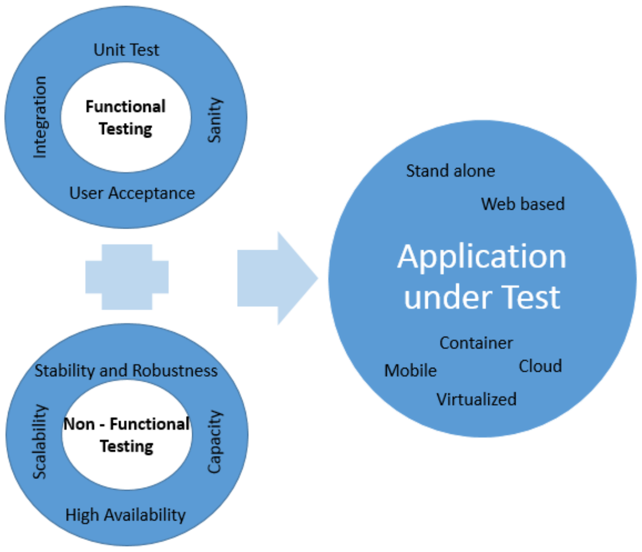


Fig 1. Application Testing

Testing of any software or application, including Container-based applications, involves two important branches of testing:

2.1 Functional Testing

It refers to the functional characterization of the application it intends to address. Any application that is developed is responsible for providing some features or solving a problem. Testing these sets of features that the application aims to provide is categorized as functional testing. This type of testing is carried out thoroughly using the well-known testing cycles during the development, testing, and QA phase.

2.2 Non-Functional Testing

It refers to the application's non-functional characteristics that describe the surrounding environment of the application and its behavior. Siddiqui et al.[7] hRave suggested three prominent non-functional characteristics - Scalability and Performance, Stability and Robustness and High Availability, and Geo-Redundancy. Building upon these proposed non-functional characteristics, we have defined and considered the four defined non-functional characteristics of Capacity, Scalability, Stability, and High Availability.

3 Proposed Framework

A defined structure with guidelines is required to streamline a set of activities and control its waywardness. Similar is the requirement for the Non-Functional Testing of the Container-based applications. A Non-Functional Requirement (NFR) Modelling framework has been proposed by Paradkar[8], which elaborates on key non-functional characteristics in business-critical systems. This section proposes a complete framework to test the container-based application with respect to all its non-functional characteristics. The figure below is an exhaustive diagrammatic representation of the proposed framework.

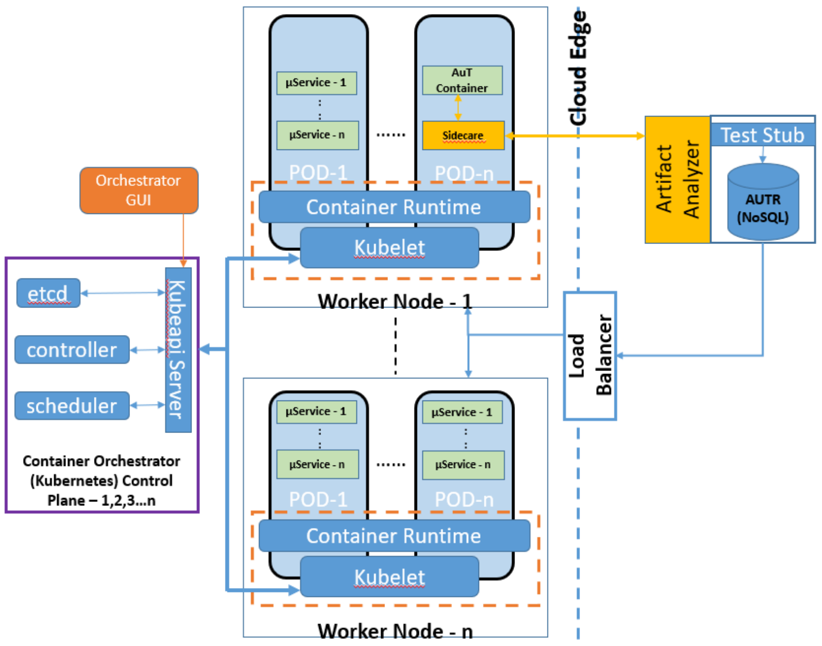


Fig 2. NFT Framework for CBA Applications

The framework is further divided into sub-parts that are integrated to give an end-to-end Container Non-functional testing framework. The subparts of the overall proposed framework are:

1. NF Test Repository creation
2. NF-AUT Setup
3. NF Test Execution
4. NFT Artifact Analysis
5. NF Test Reporting and Certification

3.1 NF Test Repository Creation

This step involves finalizing the test cases for the container application under test. The following list of activities are performed to complete the repository creation:

1) As a first step, a basic test case repository database is prepared, which houses all the test cases related to each Non-Functional characteristic. NoSQL Db is proposed to be used for the repository, and the following table definition is to be used:

Table 1. Repository Table Definition

Case-ID	TC-Summary	TC Definition	NFT-Class	Class Priority	TC-Priority	Expected Outcome	TC-Owner

All these test cases are prioritized, and the initial NFT Class prioritization is as follows:

Table 2. NF Class Prioritization Table

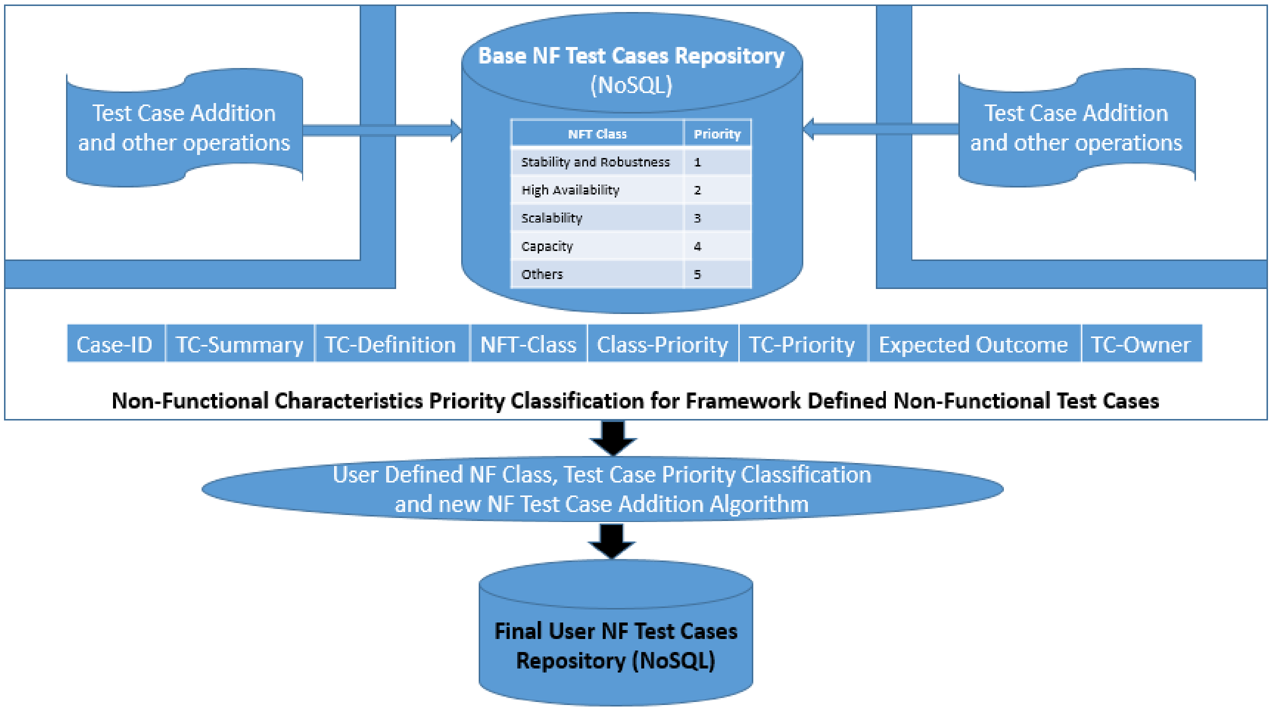
NF Class	Priority
Stability and Robustness	1
High Availability	2
Scalability	3
Capacity	4
Others	5

2) A pre-defined test suite based on all NF characteristics is loaded in the Application Under Test Repository (AUTR) database. This pre-defined test suite is common and inherited to all test repositories which are mapped to each Application Under Test (AUT).

3) Based on the AUT, new test cases are added to the repository to enhance the effectiveness of the overall repository.

4) Once all the desired test cases are captured and stored in the repository, each test case is analyzed and prioritized as per the AUT requirement.

5) As a side requirement, an algorithm is to be proposed to inject new test cases dynamically once the AUTR is finalized.


Base NF Test Cases Repository (NoSQL)
Test Case Addition and other operations
Test Case Addition and other operations
Case-ID
TC-Summary
TC-Definition
NFT-Class
Class-Priority
TC-Priority
Expected Outcome
TC-Owner
Non-Functional Characteristics Priority Classification for Framework Defined Non-Functional Test Cases
User Defined NF Class, Test Case Priority Classification and new NF Test Case Addition Algorithm
Final User NF Test Cases Repository (NoSQL)

NFT Class	Priority
Stability and Robustness	1
High Availability	2
Scalability	3
Capacity	4
Others	5

Fig 3. AUTR Creation Framework

The AUTR creation or the Customized NF Test Repository Creation Framework, as depicted in Figure 3 is a pictorial illustration of the process of creating the AUTR database. Once the test setup is ready for the AUT, this database is fed into the NF Test Execution phase.

3.2 NF AUT Setup

A non-Functional Application under Test (NF-AUT) setup is required to test the non-functional behavior of the application successfully. This can be worked in parallel to the NF Test Repository creation phase. In this activity, all the requirements for testing the application are acquired and integrated to execute the test cases defined in the AUTR. The setup takes into account the following:

1) Functionality provided by the AUT and various means to access the application.

2) Resource and instance requirements for testing all cases of the AUTR.

3) Resources required to test the NF characteristics:

a. Capacity

b. Scalability

c. Stability and Robustness

d. High Availability

The proposed setup in the figure reflects the entities required to test any container-based application for all its non-Functional requirements.

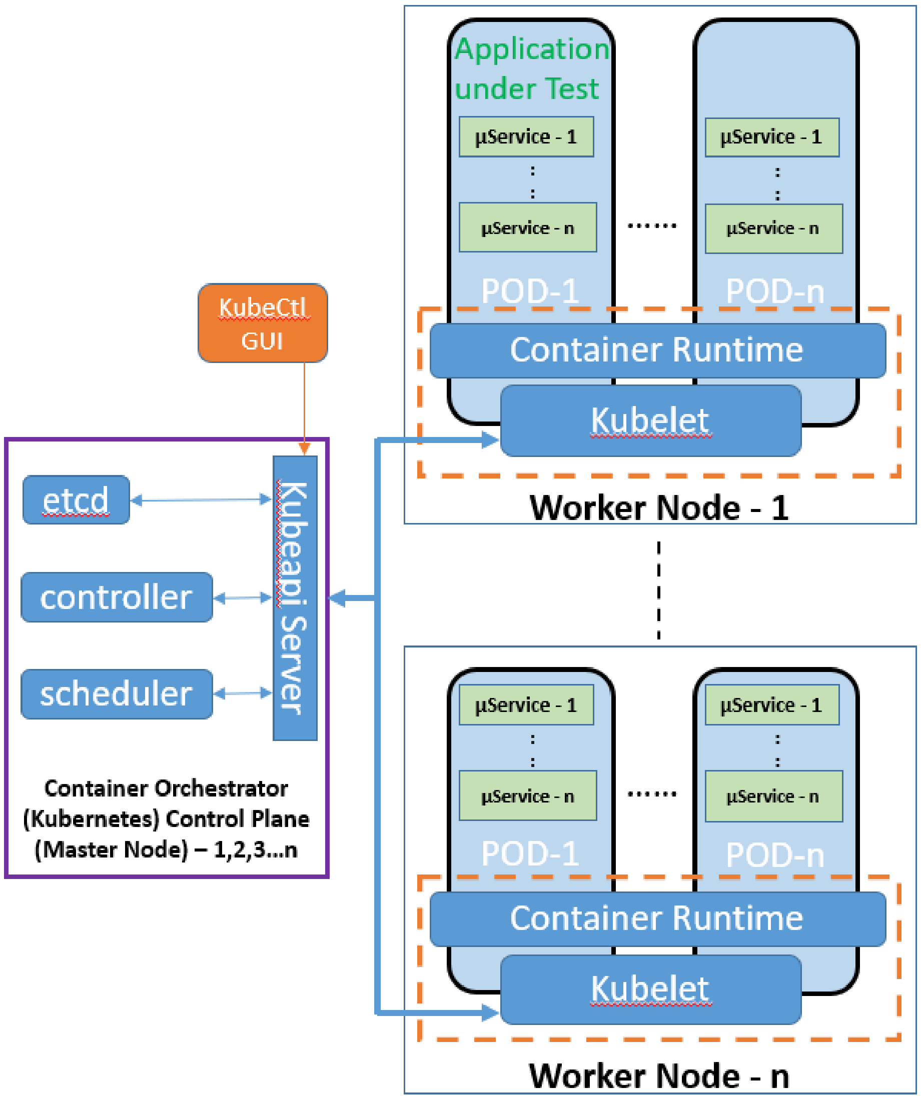


Fig 4. NF AUT Setup

In the above setup, a dedicated set of computes are pooled in to create a container setup. The computes are then divided into master and worker nodes. The master nodes are responsible for the upkeep of the worker nodes and pods running on those workers. The worker nodes are actual powerhouses that host the container-based application for testing. The number of nodes in the master can be one or added up to provide redundancy. But as we aim to test the application rather than the container testing setup, we have limited it to 1 to be cost-effective. The worker nodes can also start from one and be supplemented depending on the tested applications. Here, more than one worker node is required because we must test various non-functional characteristics where the application would need to run on multiple hosts.

3.3 NF Test Execution

In this phase, the prime objective is to integrate the design and deliverables of the previous two stages where we have Test repository creation and NF-AuT Setup. This integration will lead to a robust test execution environment. The proposed integration mechanism for test execution is pictorially defined in the below figure:

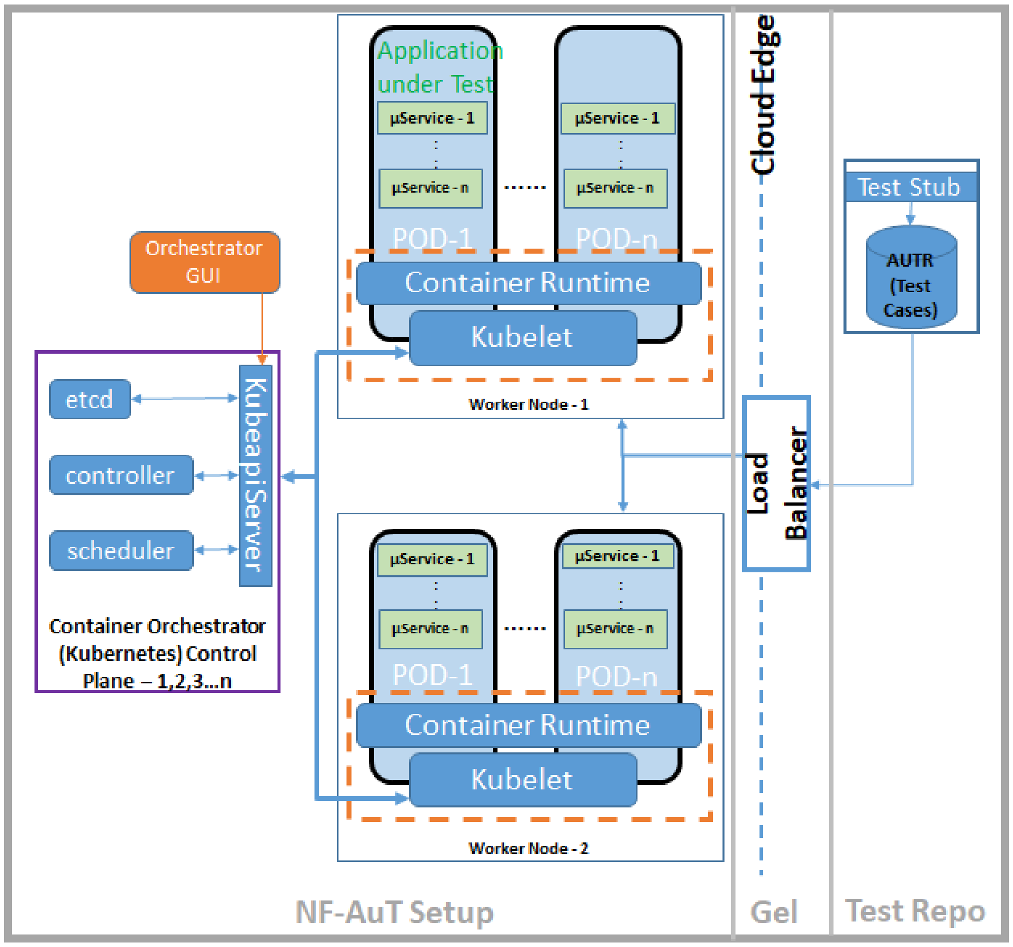


Fig 5. NF Test Execution

The integration of NF-AuT setup and Test Repo is achieved by introducing a connectivity layer between the two setups called a Gel layer. It primarily could be a networking interface helping communicate the test setup, which would be in a private cloud environment with the Test case repository used by the test stub. This test stub could be outside the private cloud and hence needs to be routed to the testing setup. For this, the Gel layer, which could act as the gateway of communication between the two, is placed. This is also an ideal place to bring in a customized Load balancer with policies to distribute the testing load as per the requirement.

3.4 NFT Artifact Analysis

Artifact analysis is envisioned by using the sidecar concept of Kubernetes where in Fault Management (FM) and Performance Management (PM) metrics of the application and the application logs will be collected by the sidecar pods deployed in tandem with the application pod that is under test. These sidecar pods would deliver the relevant artifact collected for any test cases to the NFT Artifact analyzer. The NFT artifact analyzer will extend the test stub where mechanisms will be employed to map the artifacts with its relevant test case definition.

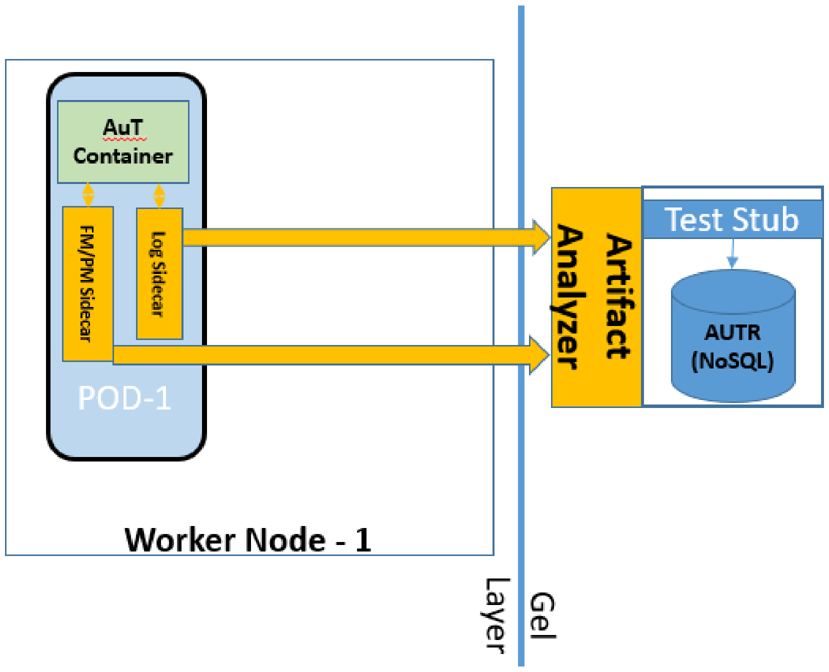


Fig 6. NF Artifact Analysis Setup

3.5 NF Test Reporting and Certification

This is the final step where all test cases have been executed, and their relevant artifacts are back in the artifact analyzer. Based on the result of artifacts, each test case would be categorized according to the following criteria.

Table 3. NFT Result Categorization

Category	Reasoning
Pass	If the artifacts are in sync with the expected result of the Test case
Fail	If the artifacts suggest that the result of the test case in not as expected
NA	The test case is not applicable for the AuT.
NE	The test case is applicable for the AuT but could not get executed because of any reason.

Based on the quantitative analysis of the cases category after its execution as defined in the above table, an algorithm will be devised to certify the application with different levels.

4 Conclusion

The proposed framework is a holistic approach that can be employed to test any Container-Based application. The framework devices a mechanism that starts from defining test cases, defining test setup, and to a proposed mechanism of analyzing the collected artifacts. This approach will help ease the burden faced by developers and testers in the fast-changing application development domain, where they must focus more on the functionality of the application rather than its non-functional aspects. It will further reduce the timelines in testing any application predefined step-by-step guidelines have been proposed to be followed. The framework also has a further scope of evolving as a certification mechanism by designing an effective algorithm on the quantitative data presented by the Artifact analyzer.

References

1) Siddiqui T, Siddiqui SA, Khan NA. Comprehensive Analysis of Container Technology. *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*. 2019;p. 218–223.

2) Siddiqui T, Ahmad R. 2018. Available from: https://www.jardcs.org/backissues/abstract.php?archiveid=3100.
3) Siddiqui T, Ahmad R. A review on software testing approaches for cloud applications. *Perspectives in Science*. 2016;8:689–691. Available from: https://dx.doi.org/10.1016/j.pisc.2016.06.060. doi:10.1016/j.pisc.2016.06.060.
4) Siddiqui T, Ahmad R. Cloud Testing-A Systematic Review". *International Research Journal of Engineering and Technology*. 2015;2(03):397–406.
5) Chuchuen Y, Rattanaopas K. Implementation of Container Based Parallel System for Automation Software Testing. *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*. 2021;p. 193–196.
6) A, Patrizio A. 2020. Available from: https://www.functionize.com/blog/5-ways-cloud-native-application-testing-is-different-from-testing-on-premises-software/.
7) Siddiqui SA, Siddiqui T. . Available from: https://doi.org/10.17577/ijertv9is070290.
8) Paradkar SS. A Framework for Modeling Non-Functional Requirements for Business-Critical Systems. *SSRN Electronic Journal*;2021:2347–5552. Available from: https://dx.doi.org/10.2139/ssrn.3778082. doi:10.2139/ssrn.3778082.