Qualification of Apache2.2 Agent Made Easy with Automation Testing Framework
Sushma J.S\(^1\), Dr. H.D. Phaneendra\(^2\)
PG Student\(^1\), Professor and Head\(^2\)
Department of ISE\(^1\), Department of CSE\(^2\)
National Institute of Engineering, Mysuru, India

Abstract:
Qualification is the main phase in the lifecycle development of a product. But in this ever growing information technology field, time is a constraint on the project. This paper mainly focuses on the reduction of the take taken for the qualification phase with the usage of automation testing framework. Along with that it also proposes how automation framework helps in the parallel execution of the jobs and efficiently carry on the testing process.

Keywords: Lifecycle development, Automation testing, Access management, Web server, Data Store, Authentication schemes, Access Manager (AxM)

I. INTRODUCTION

Software development life cycle is a process of designing, coding, building, verifying and validating software. It is a process followed even after development of the product. There are different models of life cycle that are adopted for the software development. One of the major phases of the software development life cycle is ‘Testing’; it is the process of checking whether or not the software or the piece of code we have written is meeting the user’s requirement. Qualification is also the phase of software testing where several use-cases are derived according to the requirements and verifying that the software is meeting the user demands.

High accuracy is the most expected feature of any life cycle and the testability is the key to achieve that feature. “Change is the only constant” in this booming era of Information Technology, so all the features developed must be up-to-date, and it should be able to meet all the use cases, at the same time it should be cost effective and Quality of service based.

The developed product must be able to support the requirements of all types of platforms and formats. So along with the development of the features it is very important to test them accordingly. Various tools are available for testing software. More over the reports generated by automation testing, logs are very helpful to track the execution of the test cases. Even when something goes wrong, with the help of these we can identify the problem and find the solution in no time.

The main demand of the automation testing is the tester is expected to write “reusable methods” which save the considerable time of the automation developers. Manual testing can provide very less scope for the testers to code. Integration with different vendor tools sets in terms of requirements management, test management, test script execution, defect management are easily managed by the team which give more flexibility in choosing the appropriate vendors/tools.

II. MOTIVATION

User access enables users to assume a specific digital identity across applications, which enables access controls to be assigned and evaluated against this identity. The use of a single identity for a given user across multiple systems eases tasks for administrators and users. It simplifies access monitoring and verification and allows the organization to minimize excessive privileges granted to one user. User access can be tracked from initiation to termination of user access. When organizations deploy an identity management process or system, their motivation is normally not primarily to manage a set of identities, but rather to grant appropriate access rights to those entities via their identities. In other words, access management is normally the motivation for identity management and the two sets of processes are consequently closely related.

III. PROBLEM STATEMENT

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Test Automation software is the best way to increase the effectiveness, efficiency and coverage of your software testing. Manual software testing is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations. An automated testing tool is able to playback pre-recorded and predefined actions, compare the results to the
expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. Because of this, savvy managers have found that automated software testing is an essential component of successful development projects. Automated software testing has long been considered critical for big software development organizations but is often thought to be too expensive or difficult for smaller companies to implement. SmartBear’s Tools are affordable enough for single developer shops and yet powerful enough that our customer list includes some of the largest and most respected companies in the world.

IV. OBJECTIVE OF THE AGENT TESTING

The main objective of this project is to understand the life cycle of the agent and verify that Apache 2.2 agent works properly on RHEL 7.2 platform fulfilling all the requirements of the users. Shared automated tests can be used by developers to catch problems quickly before sending to QA. Tests can run automatically whenever source code changes are checked in and notify the team or the developer if they fail. Features like these save developers time and increase their confidence.

V. TYPES OF AUTHENTICATION USED

The use of multiple authentication factors to prove one's identity is based on the premise that an unauthorized actor is unlikely to be able to supply the factors required for access. If, in an authentication attempt, at least one of the components is missing or supplied incorrectly, the user's identity is not established with sufficient certainty and access to the asset (e.g., a building, or data) being protected by multi-factor authentication then remains blocked. The authentication factors of a multi-factor authentication scheme may include: some physical object in the possession of the user, such as a USB stick with a secret token, a bank card, a key, etc. some secret known to the user, such as a password, PIN, TAN, etc. some physical characteristic of the user (biometrics), such as a fingerprint, eye iris, voice, typing speed, pattern in key press intervals, etc.

Knowledge factors

Knowledge factors are the most commonly used form of authentication. In this form, the user is required to prove knowledge of a secret in order to authenticate. A password is a secret word or string of characters that is used for user authentication. This is the most commonly used mechanism of authentication. Many multi-factor authentication techniques rely on password as one factor of authentication. Variations include both longer ones formed from multiple words (a passphrase) and the shorter, purely numeric, personal identification number (PIN) commonly used for ATM access. Traditionally, passwords are expected to be memorized. Many secret questions such as "Where were you born?" are poor examples of a knowledge factor because they may be known to a wide group of people, or be able to be researched.

Possession factors

Possession factors ("something only the user has") have been used for authentication for centuries, in the form of a key to a lock. The basic principle is that the key embodies a secret which is shared between the lock and the key, and the same principle underlies possession factor authentication in computer systems. A security token is an example of a possession factor.

Connected tokens

Connected tokens are devices that are physically connected to the computer to be used, and transmit data automatically. There are a number of different types, including card readers, wireless tags and USB tokens.

Inherence factors

These are factors associated with the user, and are usually biometric methods, including fingerprint readers, retina scanners or voice recognition.

The concept of single sign-on

SSO is user authentication service that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications. The service authenticates the end user for all the applications the user has been given rights to and eliminates further prompts when the user switches applications during the same session. On the back end, SSO is helpful for logging user activities as well as monitoring user accounts. In a basic web SSO service, an agent module on the application server retrieves the specific authentication credentials for an individual user from a dedicated SSO policy server, while authenticating the user against a user repository such as a lightweight directory access protocol (LDAP) directory. Some SSO services use protocols such as Kerberos and the security assertion markup language (SAML). SAML is an XML standard that facilitates the exchange of user authentication and authorization data across secure domains. SAML-based SSO services involve communications between the user, an identity provider that maintains a user directory, and a service provider. When a user attempts to access an application from the service provider, the service provider will send a request to the identity provider for authentication. The service provider will then verify the authentication and log the user in. The user will not have to log in again for the rest of his session. In a Kerberos-based setup, once the user credentials are provided, a ticket-granting ticket (TGT) is issued.

The TGT fetches service tickets for other applications the user wishes to access, without asking the user to re-enter credentials. Although single sign-on is a convenience to users, it presents risks to enterprise security. An attacker who gains control over a user's SSO credentials will be granted access to every application the user has rights to, increasing the amount of potential damage. In order to avoid malicious access, it's essential that every aspect of SSO implementation be coupled with identity governance. Organizations can also use two factor authentication (2FA) or multifactor authentication (MFA) with SSO to improve security.

Sooner or later web development teams face one problem: you have developed an application at domain X and now you want your new deployment at domain Y to use the same login information as the other domain. In fact, you want more: you want users who are already logged-in at domain X to be already logged-in at domain Y. This is what SSO is all about.
V. DEPENDENCIES USED FOR AUTOMATION PROJECT

Maven is a build automation tool used primarily for Java projects. The word maven means "accumulator of knowledge" in Yiddish. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with predefined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project. Maven is built using a plugin-based architecture that allows it to make use of any application controllable through standard input. Theoretically, this would allow anyone to write plugins to interface with build tools (compilers, unit test tools, etc.) for any other language. In reality, support and use for languages other than Java has been minimal. Currently a plugin for the .NET framework exists and is maintained, and a C/C++ native plugin is maintained for Maven 2. Alternative technologies like Gradle and sbt as build tools do not rely on XML, but keep the key concepts Maven introduced. With Apache Ivy, a dedicated dependency manager was developed as well that also supports Maven repositories.

VI. AUTOMATION FRAMEWORK

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionality that make it more powerful and easier to use. It is an open source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit but it is much more powerful than JUnit but still it’s inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Pay special thanks to Cedric Beust who is the creator of TestNG. TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, sequencing, parametrizing.

TestNG's main features include:

1. Annotation support.
2. Support for parameterized and data-driven testing (with @DataProvider and/or XML with @DataProvider and/or XML Configuration).
3. Support for multiple instances of the same test class (with @Factory).
4. Flexible execution model. TestNG can be run either by Ant via build.xml (with or without a test suite defined), or by an IDE plugin with visual results. There isn't a TestSuite class, while test suites, groups and tests selected to run are defined and configured by XML files.
5. Concurrent testing: run tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc.), and test whether the code is multithread safe.
6. Embeds BeanShell for further flexibility.
7. Default JDK functions for runtime and logging (no dependency).
8. Distributed methods for application server testing. 10. Distributed testing: allows distribution of tests on slave machines.

VII. SYSTEM IMPLEMENTATION

Implementation is the process that actually yields the lowest-level system elements in the system hierarchy (system breakdown structure). System elements are made, bought, or reused. Production involves the hardware fabrication processes of forming, removing, joining, and finishing, the software realization processes of coding and testing, or the operational procedures development processes for operators' roles.

If implementation involves a production process, a manufacturing system which uses the established technical and management processes may be required. The purpose of the implementation process is to design and create (or fabricate) a system element conforming to that element’s design properties and/or requirements.

The element is constructed employing appropriate technologies and industry practices. This process bridges the system definition processes and the integration process. Figure portrays how the outputs of system definition relate to system implementation, which produces the implemented (system) elements required to produce aggregates and the SoI. During the implementation process, engineers apply the design properties and/or requirements allocated to a system element to design and produce a detailed description.

They then fabricate, code, or build each individual element using specified materials, processes, physical or logical arrangements, standards, technologies, and/or information flows outlined in detailed descriptions (drawings or other design documentation). A system element will be verified against the detailed description of properties and validated against its requirements. If subsequent verification and validation (V&V) actions or configuration audits reveal discrepancies, recursive interactions occur, which includes predecessor activities or processes, as required, to mitigate those discrepancies and to modify, repair, or correct the system element in question.
The following major activities and tasks are performed during this process:

- **Define the implementation strategy** - Implementation process activities begin with detailed design and include developing an implementation strategy that defines fabrication and coding procedures, tools and equipment to be used, implementation tolerances, and the means and criteria for auditing configuration of resulting elements to the detailed design documentation. In the case of repeated system element implementations (such as for mass manufacturing or replacement elements), the implementation strategy is defined and refined to achieve consistent and repeatable element production; it is retained in the project decision database for future use. The implementation strategy contains the arrangements for packing, storing, and supplying the implemented element.

- **Realize the system element** - Realize or adapt and produce the concerned system element using the implementation strategy items as defined above. Realization or adaptation is conducted with regard to standards that govern applicable safety, security, privacy, and environmental guidelines or legislation and the practices of the relevant implementation technology. This requires the fabrication of hardware elements, development of software elements, definition of training capabilities, drafting of training documentation, and the training of initial operators and maintainers.

- **Provide evidence of compliance** - Record evidence that the system element meets its requirements and the associated verification and validation criteria as well as the legislation policy. This requires the conduction of peer reviews and unit testing, as well as inspection of operation and maintenance manuals. Acquire measured properties that characterize the implemented element (weight, capacities, effectiveness, level of performance, reliability, availability, etc.).

- **Package, store, and supply the implemented element** - This should be defined in the implementation strategy.

**Performance Analysis:**
- Time saving
- Parallel test case execution
- Less human intervention
- Faster report and insight generation
- Easy to enhance the design
- Cost effective

These are the main analysis report of agent qualification when it is compared with the manual testing.

**VIII. CONCLUSION**

This paper explains the easiness of automation testing to qualify an apache2.2 agent on RHEL 7.2 platform. The best effort is made to explain the various components involved in the product architecture and the framework used to automate them using TsetNG. Through this work an effort is made to compare the performance of automation testing and automation testing.

**IX. REFERENCES**

[1]. Seema Vargheese, Infosys, white paper-2016,”Moving Fragmented Test data towards centralized approach”

[2]. Infosys, white paper-2015,”end-to-end test automation, a behavior and agnostic approach”


[14]. www.rasuniversity.com

[15]. www.stackoverflow.com

[16]. www.emc.com

[17]. www.apache.org

**X. ACKNOWLEDGMENTS**

This paper is result of understanding and the approach made to undertake a project which is a part of lifecycle development of the product. I would like to thank my guide, all the faculties of my college and my mentors in the company for their constant support and suggestions to improve my technical knowledge. I would all thank all the authors whose papers have been used as a reference while writing my paper.