International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Designing Jenkins Pipelines for Nightly Regression on Cross-Browser CI Testing with Maven-Selenium Stack

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
In this work, we describe practices for configuring Jenkins pipelines to automate cross-browser regression testing within a Continuous Integration (CI) system for nightly automated test runs, using a Maven-Selenium toolset. Automated testing presents a challenge of homogeneous UI testing across various browsers, operating systems and settings. The above integration process by Charter Communications illustrates how automation by using Selenium WebDriver, Maven as a dependency management tool as well as Jenkins as an Orchestration tool can provide an automated, scaled and robust test execution. The system is meant to provide test execution with optimal resources such as test execution in parallel processes, and inclusive and wide-range reporting to provide timely feedback. Proactive maintenance of test cases is managed by test-case modularity, test-browser compatibility monitoring and version-controlled pipeline-as-code maintenance strategies. The QA team benefits through a reduction in manual work, test-driven release processes, quick regression stability, and quicker detection without impacting the release cadence.

Keywords: Jenkins Pipeline, Selenium Grid, Parallel Execution.

I.Introduction

Regression testing is a necessary part of a modern software delivery pipeline that can be used to ensure that new code being added does not affect the existing functionality. Regression testing is also essential where a large organization like Charter Communications exists and should be able to maintain the application functionality round-the-clock on a very large number of browsers and operating systems. Any divergence is a risk of customer dissatisfaction, lack of credence and inefficiency of operation. The manual regression testing is not effective; it is also time-consuming and resource intensive and it does not entirely cover delivery time. Automation can deal with the problems by providing better coverage, testing speed and repeatability. In this respect, an overview was created of Jenkins to coordinate, Maven to coordinate dependencies, and Selenium WebDriver to automate cross-browsers. All these tools enable a scalable maintainable system to perform nightly regression tests in parallel with many environments and thereby provide speed, reliability and improved user experience.

II.Background and Rationale

Scalable software systems continue to evolve, and code is often modified regularly to provide new functionality, security updates and performance improvements. Every update however comes with the risk of introducing regressions which may be detrimental to the stability of the system [5]. In the case of Charter Communications, which relies on digital platforms to engage the millions of users every day, it is a matter of mission critical importance that new deployments do not undermine service. Historically common, manual regression testing cannot sustainably perform in the speed of the modern CI-CD pipelines. It has low test coverage, inconsistencies based on human intervention, and delays that do not

promote quick release cycles. These shortcomings make automated solutions very urgent. To overcome these issues, the Jenkins-Maven-Selenium stack was chosen due to it being a mature, open source and broadly supported stack. Jenkins provides uninterrupted integration and orchestration, Maven provides consistent dependency management, and Selenium WebDriver provides cross-browser validation. The combination guarantees rapid implementation, increased coverage, and scalable testing system that meets enterprise delivery objectives.

III.System Architecture & Pipeline Design

Currently the code-based, pipeline-based nightly regression is based on a declarative Jenkinsfile. The pipeline also provides consistency among builds as the stages are reusable and modular and extend to future improvements. Stages have been represented in a manner that indicates the presence of such stages as checking out code, environment comparison, taking tests and reporting. This modularity improves the maintenance since it can be upgraded at the conclusion of stages without affecting the entire pipeline.

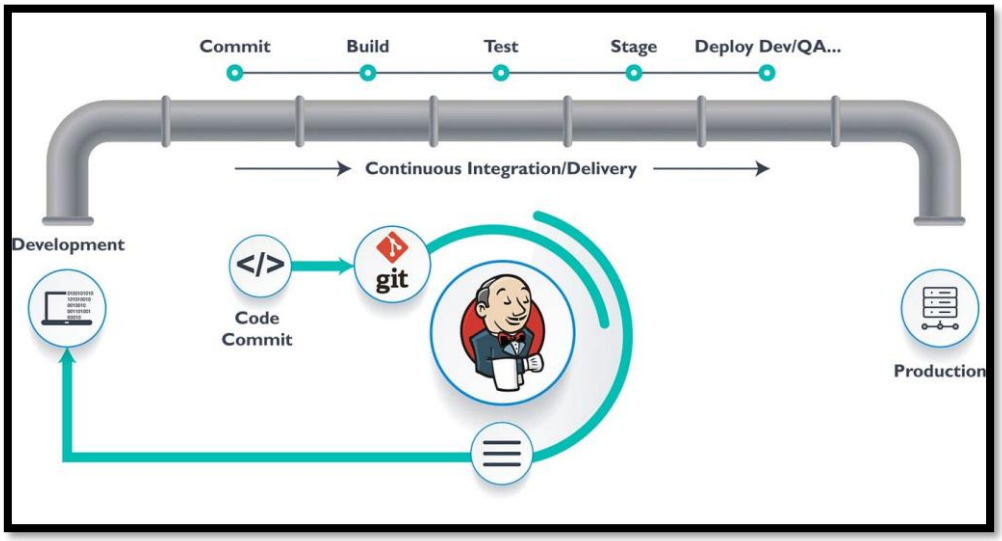

Fig 1: Jenkins Pipeline

Also, declarative syntax is intuitive and not that difficult to read because it lacks complexity and it makes the pipeline approachable to a developer and QA engineer [3]. The principle of parallel processes is also applied to the systems creating the needs to satisfy the cross-browser validation to minimum windows of operation during a given night. It is distributed using Selenium Grid which can conduct distributed testing on a combination of various browsers and operating systems. Elastic execution environment where extra scalability is needed can be configured to cloud based services like BrowserStack or Sauce Labs. The design minimises the time of execution; it provides wide coverage and believable test responses in a period that is readily accommodative.

Containerization induces conformity of test environments. A docker container also contains a browser driver, Selenium nodes, and other devices to provide coherent performance through a large number of machines. This can be achieved by keeping a version-managed artifact to define the environment which makes the test runs traceable and reproducible. The pipeline-as-code paradigm results in a regulated implementation, code of the environment, tracked and tested. They cancel machine variations and streamline automated testing on a scaled footing. Integration with Git based source control can be used to activate pipelines in response to code commitments, mergers or to scheduled night jobs [2]. The Jenkinsfile is committed to the repository, as pipeline definitions are versioned to the application code. This conformity of the pipeline and the code environments provides transparency, rollback where necessary and CI/CD government.

IV.Integration & Maintenance Strategy

Regression framework consistency will be based on the harmonious interoperability of tools and proactive maintenance to ensure long-term effectiveness. Maven is a dependency management key that ensures that all libraries, browsers, and testing systems are always updated and resolved across platforms. One pom.xml will make sure that developers and testers will not have dependency conflicts and generally, replicate the builds in various machines and containers [1]. Jenkins extends this with bespoke fit selenium execution plug-ins. A good example of a plug-in that makes configuring test jobs easier, allows results to be structured and minimizes manual work is the selenium grid and JUnit Reporter. This allows easy execution pipelines with each step of the execution pipeline - setting up of the environment to test orchestration - automated in Jenkins.

The other common issue is compatibility with browser-drives where the browsers will be updated and likely to fail the test automation. This can be addressed by setting up automated driver management software e.g. WebDriverManager that will map drivers to appropriate browser versions on-the-fly. Inspections will also contribute to ensuring that the minimum level of inconveniences related to version conflicts is ensured. The test cases also have the problem of modularity. This eases the maintenance through creation of tests on terms of reusing elements thereby reducing the chances of cascading errors. The configuration itself is scaled and robust and meets the changing requirements of the CI/CD through proactive maintenance as such a routine verification of the code, refactoring and compatibility-testing.

V.Scalability & Optimization

Scalability Nightly testing should be rapid and reproducible, across several browsers. In a homogeneous environment we can get this scalability by Containerization using dockerization. The dependencies (browser drivers and Selenium nodes) are found in each container which removes the mystery of having variability in machines and allows the tests to behave the same in the development, staging and production pipelines. At the same time, parallel performance is also more efficient.

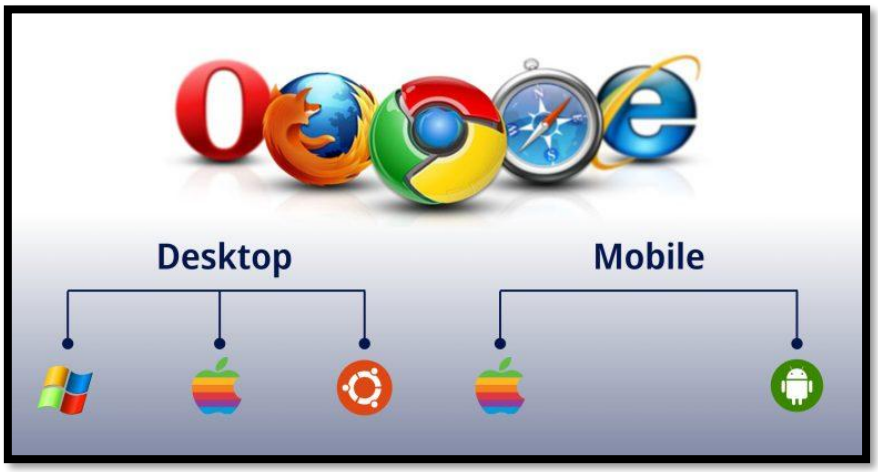

Fig 2: Cross Browser Testing

The system can run hundreds of test cases at a given time by spreading tests to several containers and using the power of Selenium Grid. This considerably decreases the overall runtime of night regressions, meaning that detailed feedback is provided prior to the commencement of the following development cycle [6]. The dynamic scaling of containers about workload demand leads to optimized resource usage. Unutilized resources are released during idle time, and peak demand initiates new ones, which are cost-effective without compromising the volume of tests or the speed of their implementation.

VI.Reporting & Monitoring

Reporting and monitoring are essential in converting the raw test execution process into information that can be acted on. Jenkins dashboards offer real-time visibility and past trend analysis, allowing teams to monitor regression results as time passes. Test results are compiled and presented with intuitive indicators of pass rates, execution time, and browser specific failures, facilitating quick reporting of system health. To ensure that issues are flagged in their initial stage, automated defect reporting is integrated into the pipeline.

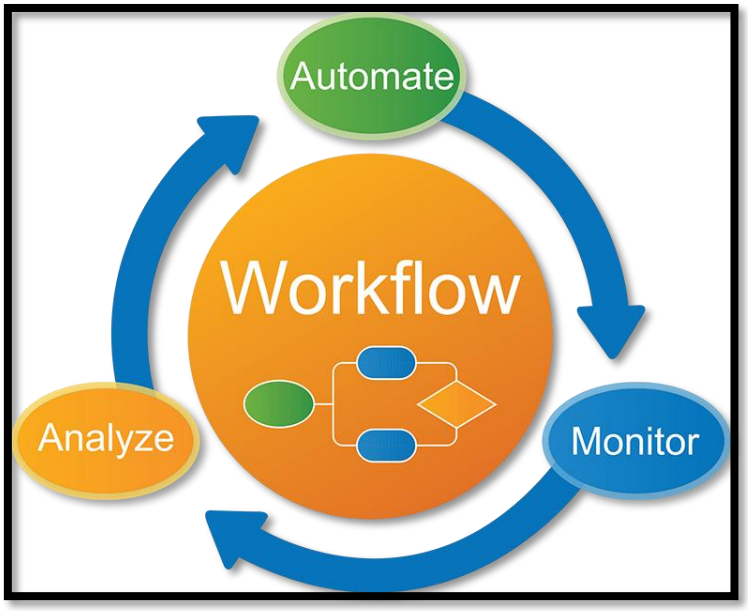

Fig 3: Automation Workflow

Failures generate alerts via email or built-in messaging services like Slack, which give real-time visibility to developers and QA engineers. This quick negative feedback loop minimizes the average time to identify and fix defects, aiding in sustaining the velocity of releases. The combination of reporting plugins improves transparency and traceability. JUnit reports structure raw results, while Allure and Extent Reports provide rich, interactive visualizations with logs, screenshots, and test histories. Collectively, these tools form an entirely reporting ecosystem that can aid in informed decision-making.

VII.Conclusion

Jenkins-Maven-Selenium regression framework has been able to enhance stability and the rate in the software delivery pipeline of Charter Communications. The automated regression process is being used daily to identify cross browser problems early and therefore the manual work required is minimized and the release prepared. Modular pipeline development, container, and parallel processing offers scalability and efficiency, whereas proactive maintenance offers long-term stability. The project improved the maturity of the CI/CD by integrating quality checks in development. Automated defect notification and real-time reporting allowed quick feedback, allowing the making of go/no-go decisions with confidence without sales being withheld. Moving forward, AI can increase the capabilities in the framework in terms of test case maintenance to minimise flakiness and cloud-native scaling to meet increased workloads.

REFERENCES:

[1] Arachchi, S.A.I.B.S. and Perera, I., 2018, May. Continuous integration and continuous delivery pipeline automation for agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)* (pp. 156-161). IEEE

[2] Elazhary, O., Werner, C., Li, Z.S., Lowlind, D., Ernst, N.A. and Storey, M.A., 2021. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, *48*(7), pp.2570-2583.

[3] Long, Z., Wu, G., Zhang, Y., Chen, W. and Wei, J., 2021, April. Poster: Repair cross browser layout issues by combining learning and search-based technique. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)* (pp. 470-473). IEEE.

[4] Vucnik, M., Solc, T., Gregorc, U., Hrovat, A., Bregar, K., Smolnikar, M., Mohorcic, M. and Fortuna, C., 2018. Continuous integration in wireless technology development. *IEEE Communications Magazine*, *56*(12), pp.74-81.

[5] Wu, G., He, M., Chen, W., Wei, J. and Zhong, H., 2018. X-Check: Improving effectiveness and efficiency of cross-browser issues detection for JavaScript-based Web applications. *IEEE Transactions on Services Computing*, *14*(4), pp.1123-1137.

[6] Xu, S., Zhou, C., Gu, Z., Wu, G., Chen, W. and Wei, J., 2018, July. X-diag: Automated debugging cross-browser issues in web applications. In *2018 IEEE International Conference on Web Services (ICWS)* (pp. 66-73). IEEE.