

# Leveraging Docker to accelerate application life cycle for a major technology company

Our client is a leading IT company catering to organizations across multiple industries. Their wide range of technology products empowers businesses to boost their revenue by growing sales and marketing. One of their product offerings is a fully automated platform used by their clients belonging to a particular industry. This product mines data, identifies and attracts business opportunities, manages inventory, and does more from an easy-to-use interface.


[DOWNLOAD PDF](#)

## How Did It All Start

The client keeps building and releasing regular feature upgrades to enhance their product. They perform manual as well as automation tests before releasing new versions. The client used **C#, Selenium Grid, MS Test, MS Azure** to run their automated tests in the Azure pipeline. They have followed Page Object Model with Page factory design in their testing framework and maintained the codebase in the Azure repository.

A nightly build was triggered automatically and ran all the automated tests in the QA environment in separate virtual machines (VM). These VMs let individuals emulate computers and run tests from different operating systems on a single physical machine.

The client's automated tests often failed due to the inefficiency of the VMs. Furthermore, the VMs also gave way to memory issues, high operating costs, and difficulty in maintaining and upgrading the Selenium Grid. The flaky nature of the client's existing test framework caused more and more automated test failures every time. These issues impacted the nightly test runs, and the QA team had to restart it manually every time.

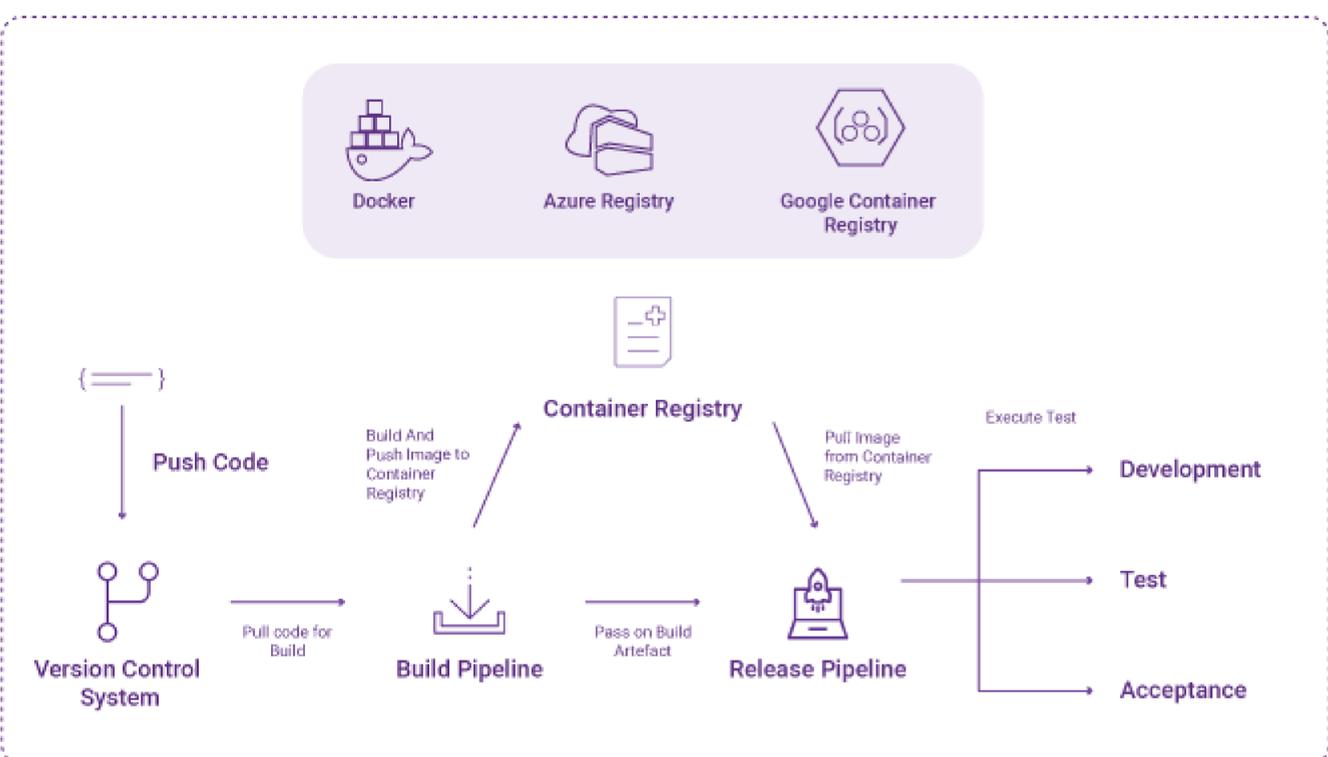
The overall test execution time increased and led to poor performance. It also slowed down the testing team's productivity as they were no longer able to differentiate between a test that had a bug and one that they were wasting their time on because of an invalid fail. The framework was also devoid of wrapper classes, and that led to a lack of code reusability.

## What Did We Do

When the client came to TVS Next with all the issues mentioned above, we decided to investigate the tests that were going flaky. To identify the problem, we familiarized ourselves with the common reasons for flaky tests in this environment.

We quarantined the flaky tests, fixed them, and got them to run again. We investigated the daily QA run in Azure, performed root cause analysis, and flagged the reason for the issues early on. To overcome the problems caused by VMs, we started running the automated tests in docker containers.

## Architecture



The Docker platform enabled the client to isolate and securely run multiple containers on the same host machine. Due to the lightweight of the containers, the client was able to share them easily. Because the containers had everything needed, the client could run tests independent of what was installed on the host machine. The client can now run wholly isolated tests in Dev, Test, Acceptance, and Production environments.

## The Business Outcome

**30%** reduction in test timing



Elimination of VM issues



Parallel test execution through containers



Ability to scale testing to infinity



Exploratory testing to aid the development team



Broke-free from OS dependencies in testing