Cloud-Based Automated Testing Essentials

Scaling the Benefits of Low-Code Test Automation

JUSTIN ALBANO
SOFTWARE ENGINEER, IBM

CONTENTS

As software engineering has advanced over the past few decades, automated testing has taken hold as the primary mechanism for validating applications. With the introduction of Continuous Integration (CI), Continuous Delivery (CD), and testing pipelines, developers can now automatically test and deploy their applications by simply committing a change to a repository. While lower-level tests can be easier to automate in the same language as the code itself, system and acceptance tests have been more difficult to automate due to their wider scope and abstraction.

Low-code and no-code tools have greatly reduced the burden on testers, but even these tools can require extensive understanding of the deployment environment and infrastructure. Cloud-based automated testing tools, on the other hand, provide a valuable addition to our toolbox and allow us to create and access our tests from anywhere without stressing about infrastructure.

In this Refcard, we will explore the basis of automated testing and delve into the details of how cloud-based automated testing solutions can dramatically improve our test suites and reduce the burden on both developers and testers.

AUTOMATED TESTING FUNDAMENTALS

The goal of software testing — whether automated or manual — is to ensure that our application or system adheres to its **specifications**. A specification can be as simple as a component producing a specific or discrete result, or it can be as complex as a component completing its execution within a statistically bound period of time.

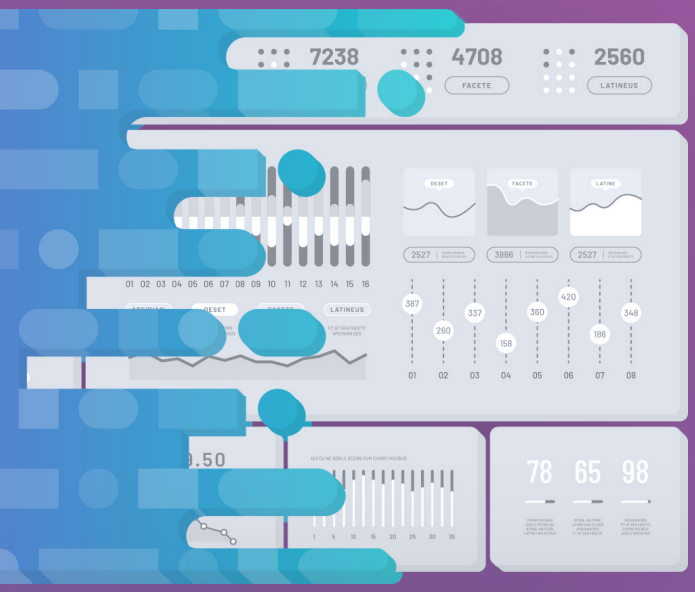
In most cases, our systems will have multiple levels of specifications, with each successively lower level focusing on a smaller subsection of the system. For example, our highest-level specifications will likely focus on the system as a whole, executing in a scaled-down clone of the production environment, while the lowest-level specifications may focus on a single class or method. The tests that we create must, therefore, reflect the granularity of the specifications being tested.

V-MODEL

Generally, our development will have four **phases** — or levels of specifications and corresponding tests:

1. **Unit** – Focuses on individual classes and methods.
2. **Integration** – Focuses on interfaces and interactions between units.
3. **System** – Focuses on the entire system while mocking external interactions.
4. **Acceptance** – Focuses on the entire system executing in an environment that mimics the production environment.

Scaling Testing with Intelligent Test Automation



Test automation is a key enabler - or inhibitor to - realizing the potential of DevOps. Automation is critical to innovating with speed and quality, and we've spent innumerable hours of engineering effort on sophisticated test automation suites. But, lengthy test creation times and hours spent on maintenance prevent us from scaling test coverage.

In order to achieve quality goals, we need testing that is intelligent and accessible to everyone. Built for high-velocity teams, mabl's low-code, intelligent test automation solution helps teams scale their quality efforts.

- Contribute to quality and velocity with easy test creation
- Reduce maintenance with reliable execution and insights
- Eliminate infrastructure management with a native cloud solution
- Increase test coverage across devices and browsers
- Integrate testing directly into your development pipeline
- Democratize testing with an accessible low-code interface

90%
Average increase in test coverage

3x
Faster test creation speed

40%
Fewer bugs in production

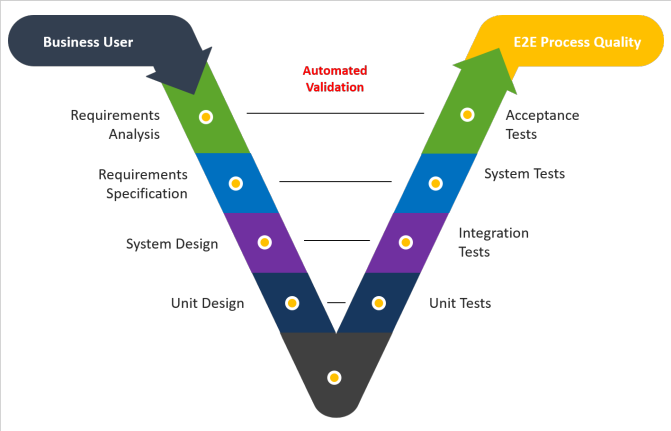
50%
Reduction in time to deployment

TRY MABL FOR FREE:

mabl.com/trial

We can visualize these phases using the V-Model:

Figure 1



Tests can be executed either manually or automatically. While manual tests have their place — often as in visual or User Interface (UI) testing — they have been largely phased out, and for good reason. Manual tests are usually:

- **Slow** – Tests can only be executed as fast as a human can complete a test and record the results.
- **Inconsistent** – It is difficult to repeat a test the same way twice, which introduces variance and irregularities.
- **Tedious** – Following the same steps each time a test is executed can become monotonous over time and cause testers to lose concentration.

In contrast, ***automated testing*** — creating repeatable tests that can be executed on demand — has significant advantages over manual testing, including:

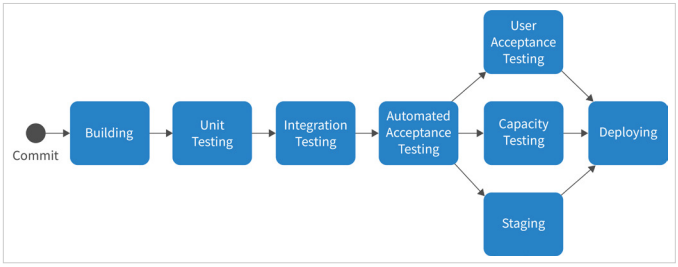
- **Speed** – Tests execute like code.
- **Consistency** – Tests are executed in the same environment every time.
- **Repeatability** – Tests are executed the same way every time.
- **Integration** – Tests and results can be easily integrated with other tools.

PIPELINES

When a significant number of tests, called a ***suite***, are accumulated for each phase, we can further automate our testing by creating a ***pipeline***. A pipeline is a set of steps that we can automatically execute to exercise our system. Each ***stage*** in the pipeline generally corresponds to a phase of testing. For example, the pipeline (or a variation of it) in Figure 2 is common when testing larger systems.

SEE FIGURE 2 IN NEXT COLUMN

Figure 2



At its genesis, a single change to our system — encapsulated as a commit to our version control system — results in a new execution of our pipeline. Each stage, from left to right, walks higher up the right side of the V-Model. In the case of the pipeline above, we may even include performance or accessibility tests (e.g., the user acceptance testing, capacity testing, and staging stages), as well as a deploying stage that ultimately deploys our system into its production environment.

Regardless of the specifics of each stage and the combination of stages we utilize, when one stage of the pipeline successfully completes, the next stage executes. Once the entire pipeline completes, we know that the changes we made in our commit meet all of our specifications. The faster the pipeline executes, the faster we get feedback about our commit. Therefore, the more stages that we automate, the quicker we can get feedback about our change.

AUTOMATING TESTS

Lower-level stages, such as unit and integration testing, are usually easy to automate. At these points in the V-Model, tests are typically written by the development team in the same programming language as the components being exercised. As we work our way further up the V-Model toward system and acceptance testing, this is no longer the case. As the tests become more abstract, more often than not, non-developers will write the tests.

For example, it is common that the stakeholders in a product may write the acceptance tests in a natural language, such as English. Since these tests are being written by non-developers, automating them can be difficult. To solve this problem, we can use automated testing tools that translate natural language into automated actions, or even allow a tester to drag and drop actions to create tests. One of the best tooling options available is cloud-based test automation software.

KEY CONCEPTS AND FEATURES OF CLOUD-BASED AUTOMATED TESTING

Automated testing tools can largely be divided into three categories:

1. **Code-based** – Uses a programming language to represent the actions executed when a test is run. Examples include Cucumber and Selenium. The advantage of these tools is that test specifications can be written in natural language, while the test itself can be executed in code, allowing for seamless integration with the product under test.

2. **Low-code** – Uses a minimal amount of programming to create tests. This category has the advantage of requiring no code — specifically, a lack of extensive programming experience — while also providing the granularity of the code-based approach. Examples include mabl and AutonomIQ.
3. **No-code** – Does not utilize code to execute tests. Examples include Katalon Studio and Perfecto. No-code tools have the advantage of requiring no programming experience to create tests, but usually at the cost of granularity and specificity (i.e., results in more abstract tests).

A large portion of automated tests are written using the code-based approach since this is a natural extension of existing automated testing processing. For example, a stakeholder will create a specification in a format derived from natural language, such as Gherkin, and then the development team will implement the steps in this specification in code. While this approach allows the development team to continue using the programming languages they are familiar with to implement the tests, it requires that the development team be involved in the process of creating high-level tests.

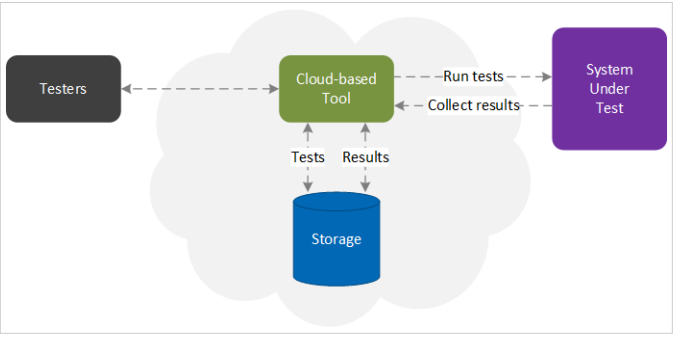
Ideally, stakeholders (without programming experience) should be able to create tests without needing to write any low-level code or calling on the development team to write low-level code on their behalf. While no-code tools do exist, low-code tools offer the benefit of more natural integration, while requiring much less technical input from stakeholders.

In particular, cloud-based low-code testing tools provided a unique and valuable mechanism for creating low-code tests.

CLOUD-BASED AUTOMATED TESTING FUNDAMENTALS

Unlike many other automated testing tools, cloud-based tools do not reside within the infrastructure of our company. Instead, these tools live on the internet and interact with our systems remotely, as illustrated below:

Figure 3



To create a test, a tester uses the tool's UI to create a **workflow** — or a set of actions — and states the expected results. This could mean dragging and dropping a set of clicks and actions, or detailing the endpoint and payload of an Application Programming Interface (API) call. Once this workflow is established, the tester then creates a set of expected results, against which the actual results of the workflow can be verified. This test case is then stored in the cloud.

When executing a test, the tester can either run the entire suite of tests or some subset of tests. The tool then interacts with our deployed product and executes the workflows. Once the workflows are completed, the results are stored in the cloud. The tool can then compare the results against the expected results and show us which test cases passed and which ones failed.

In many cases, the tool can even drill down into which specific parts of a workflow failed and provide us greater insight into why our test cases failed. For example, some tools will provide a difference comparison (diff) between the actual and expected results of an API call. This allows us to focus on the output that differed from the expected results, rather than focus on the entirety of the output (which can be very large for some API calls). Many of these cloud-based tools will also include integration and support repositories such as GitHub and GitLab, as well as CI/CD tools like Jenkins.

Cloud-based automated testing tools have several significant advantages over their native peers, such as scalability and ease of access. However, it is also important to consider other aspects (e.g., security, integration) with existing infrastructure before committing to a cloud-based solution.

BENEFITS OF CLOUD-BASED AUTOMATED TESTING

The benefits of cloud-based automated testing include:

- **Scalability** – The tool automatically grows as the product grows. Testers do not need to increase the resource pool at their organization to support testing as the product begins to grow in size and scope.
- **Availability** – Testers benefit from the vast resource pool of the provider, which minimizes the number of interruptions and downtime of the tool.
- **Manageability** – Compute and storage resources are automatically managed by the provider. This reduces the need to stand up vast resources to support automated testing. This is particularly beneficial for smaller or mid-sized organizations that do not have the infrastructure already established to support large-scale testing solutions.
- **Maintainability** – Tools are automatically updated and maintained by the provider. This reduces the maintenance and upkeep burden for an organization.
- **Accessibility** – Cloud-based solutions can be accessed from anywhere the internet is available. This creates a shared space that all members of the team can access, which promotes visibility and reusability.

In addition, many cloud-based automated testing tools include advanced capabilities, such as Artificial Intelligence (AI) and Machine Learning (ML). Since these cloud providers support a large number of products (not just our own), their ML and AI models incorporate a larger volume of data and can, therefore, be more accurate and provide greater insights.

For example, if a cloud-based tool supports thousands of projects, the ML model for this tool would be based on a more comprehensive set of data compared to a company that supports only a handful of projects in house.

CONSIDERATIONS FOR CLOUD-BASED SOLUTIONS

While cloud-based tools have a significant number of advantages, there are also a few points that should be considered before transitioning:

- **Interacting with infrastructure** – Some teams may host their testing environments on-site or on-premises. This allows the team to use their own compute and storage resources, or to closely mimic their production environment. These sites or premises may not be easily accessible from the internet, precluding the cloud-based tool from accessing the system under test.
- **Security** – Some systems may be hosted in highly secure environments that cannot be accessed from the internet or that require storage (even of tests and test results) to be maintained on-site. For example, proprietary information may be maintained in on-site storage, or a product may be classified, where an internet-based tool may not have access to the network that the product is hosted on.
- **Pricing** – Teams that already have the infrastructure and capability to support large-scale automated testing may incur added costs when using a cloud-based solution (i.e., may pay for a service that could be hosted in-house and integrated into existing infrastructure for minimal cost).

Cloud-based tools can be valuable in a wide array of situations, but it is still important to consider the specific needs of our product and ensure that these tools integrate well into our environments and provide maximal benefit to our testing infrastructure.

PROMINENT TOOLS

The cloud-based automated testing tools space is relatively young, but a few products have already earned their popularity:

- **mabl** – A functional, cloud-based testing solution that allows testers to create End-to-End (E2E) tests through a web-based dashboard. These tests can be created using a browser tool that can record actions or through API testing support. mabl includes integration support for widely used tools such as Jira, GitHub, GitLab, Jenkins, and CircleCI, as well as many others.
- **Katalon TestCloud** – A unified tool that allows testers to create, orchestrate, and execute tests within the Katalon ecosystem. This product integrates seamlessly with Katalon Studio and Katalon TestOps, and also includes integration support for Jira, GitHub, GitLab, Jenkins, and a host of other tools.
- **Micro Focus UFT One** – An E2E testing tool that focuses on AI and aids developers in creating tests that increase coverage of their products. UFT One also supports CI/CD integration with Jenkins and version control integration with Git and Subversion.

CONCLUSION

Automated testing has taken hold as the dominant mechanism for testing software — and for good reason. Even with advancements such as code-based unit and integration test frameworks, creating automated tests can still require a significant amount of prerequisite coding knowledge. Cloud-based automated tools abstract a large portion of these intricacies and allow testers to access their test suites from anywhere.

Even though the cloud-based tooling space is still maturing, tools such as mabl, Katalon TestCloud, and UTF One can provide a significant boost in performance to both large and small teams.

WRITTEN BY JUSTIN ALBANO,
SOFTWARE ENGINEER, IBM

Justin Albano is a software engineer at IBM responsible for building software-storage and backup/recovery solutions for some of the largest worldwide companies, focusing on Spring-based REST API and MongoDB development. When not working or writing, he can be found practicing Brazilian Jiu-Jitsu, playing or watching hockey, drawing, or reading.

DZone
600 Park Offices Drive, Suite 300
Research Triangle Park, NC 27709
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.