Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2021, 8(6):131-134

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

On-Demand Test Automation Environments: A Comprehensive setup guide

Srividhya Chandrasekaran

Bedford, USA
Email: srividhya.chandrasekar@gmail.com

Abstract In an era where businesses increasingly embrace digital solutions to optimize workflows, enrich user interactions, and unveil groundbreaking products, the call for resilient and flawless software has reached unprecedented levels. Manual testing, while essential, often needs to be a complete solution to meeting the demands of modern software development due to its time-consuming nature, susceptibility to human error, and inability to scale efficiently. Additionally, the push to deliver high-quality software and achieve value faster has resulted in more companies automating their unit, integration, and even performance tests. Automated testing contributes to cost-effectiveness by minimizing the reliance on manual labor, allowing teams to allocate resources more efficiently and focus on higher-value tasks such as design, innovation, and improving overall system functionality. However, to run automated tests, the need for automated environment provisioning becomes a necessity. With the ability to provision and tear down a virtual test environment at will, running automated tests regularly and monitoring the results becomes possible. This will also improve the cycle time and reduce the time to release. This white paper will address the importance of on-demand test environments for improving product software quality and reducing time to production.

Keywords Test environment, automation, containerized environments, Virtual machines, Infrastructure as Code

1. Introduction

In software development, a test environment is like a special playground where the testing team verifies and validates software. This playground comprises specific software and hardware setups, including environments such as pre-production or staging environments. Think of it as a practice version of the real thing, like a rehearsal space for software. It helps the testing team discover any issues with the software before it's officially released. This way, test engineers can fix any problems and make sure the software works smoothly in production.

While building these test environments manually on a need basis is an option, it slows down the entire testing process due to the time taken for setup, configuration, and tear down. This has led to product teams automating the entire process of setup, configuration, running the automated tests, and tear down. Provisioning and terminating environments for testing on demand saves resources and reduces cost when compared with on prem static environments that are always up and running.

2. Demystifying Automated Environment Provisioning

Provisioning is the process of creating and setting up infrastructure. It also includes steps to permit user and system access. An environment comprises 3 main areas [1]: infrastructure, configuration, and dependencies. While infrastructure defines where the environment will run, configuration directs how the infrastructure behaves concerning the application under test and how the application interacts with the infrastructure. Dependencies are the services and libraries that the application under test depends on.

Automated provisioning is a key DevOps capability that delivers computing capacity on demand without manual intervention [2]. To achieve the goals of automation and reduce waste, a good amount of planning and assessment needs to be done.

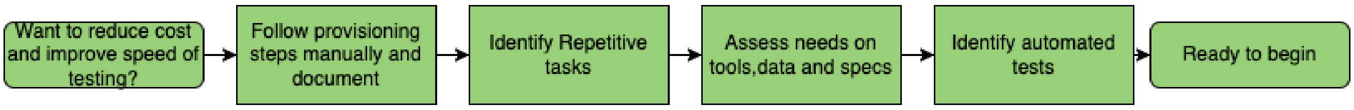
Want to reduce cost and improve speed of testing?
Follow provisioning steps manually and document
Identify Repetitive tasks
Assess needs on tools,data and specs
Identify automated tests
Ready to begin

Figure 1: Thinking through the needs of on demand automated environments

3. Deconstructing the Planning Steps

- Identify the need for automated environment provisioning
- Go through the process manually and document it
- Identify repetitive tasks and key areas that are time-consuming and can be automated.
- Assess tools, and scripting languages needed for environment automation.
- Assess hardware and software specs, including CPU, memory, disk space, OS, database, middleware, and applications.
- Evaluate network setup, considering bandwidth, latency, firewall settings, and VPNs.
- Plan for pre-seeding necessary data into the environment and assessing needs for data masking.
- Define access and security policies, covering user roles, permissions, encryption, and backup procedures.
- Identify needs to streamline resource allocation, scale on demand to prevent unnecessary costs, and avoid delays.
- Identify the automated tests that need to run on the environment and the schedule of these runs.

Once the planning is complete, the actual work to automate the provisioning can begin. This primarily includes the below steps after setting up the tools and writing the scripts needed.

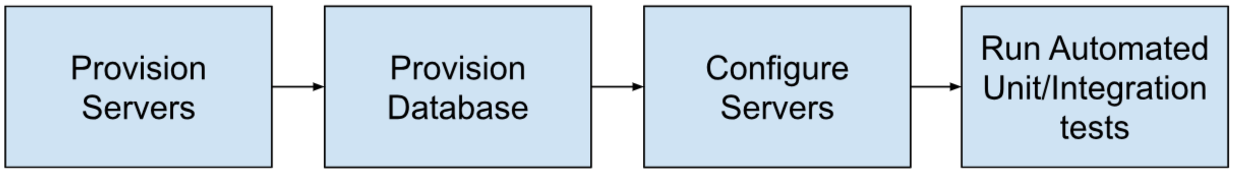

Figure 2: Steps for setting up the environment

In the present day, automation, particularly through Infrastructure as Code (IaC), has simplified many provisioning tasks. Infrastructure as code (IaC) involves storing infrastructure details in configuration files, enabling developers to set up the same environment by simply running a script. By turning infrastructure into code, IT teams get a reusable template for provisioning. All IAC follows Idempotency. Idempotency refers to the property of an operation or function where applying it multiple times has the same effect as applying it once.
These scripts can be adapted to provision a new environment and also to upgrade an existing environment also called fresh installs and upgrades.

3. Unleashing Efficiency by Using the Power of Containerization

Automating tests can be intricate due to the ongoing maintenance of browsers and WebDriver versions. Moreover, having Java, Python, or Node.js installed is necessary, with a keen focus on ensuring compatibility with the appropriate versions. While on-prem infrastructure as code setup is also a possibility, Containerization[3] eradicates these problems. A test in a container runs consistently on any machine, be it CI or local, regardless of the underlying operating system and dependencies. Because these tests operate within isolated containers, there is an opportunity to scale up testing infinitely. Containerization [4]accelerates and enhances the security of application deployment, resolving challenges in code transfer across diverse computing environments. By sharing the machine's OS kernel, containers are inherently more compact, ensuring quicker startup times, increased server efficiency, and cost savings.

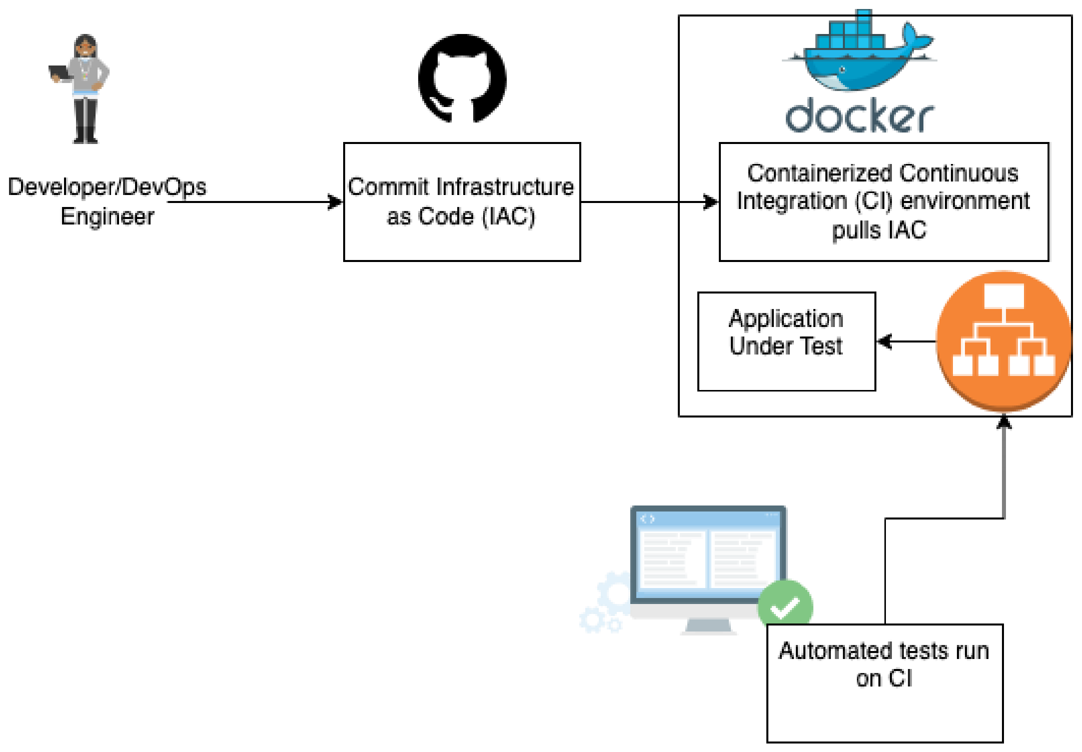
Developer/DevOps Engineer
Commit Infrastructure as Code (IAC)
docker
Containerized Continuous Integration (CI) environment pulls IAC
Application Under Test
Automated tests run on CI

Figure 4: An example setup of a provisioned environmen

3. Post-Test Run Actions: What to Do After Completion

Upon completing the test run, it is advisable to terminate these environments, releasing resources for other purposes until the next scheduled testing run. Test engineering should also remember the importance of saving logs in an accessible location for evaluating test failures and facilitating prompt bug fixes, thereby contributing to elevated product quality.

4. A Peek into what Success Looks Like

- Faster provisioning of computing resources
- Improved scalability and flexibility
- Access to an on-demand pool of computing resources
- Less manual workload
- Dynamic infrastructure with minimal waste and automated scaling

5. Conclusion

In conclusion, this comprehensive guide has thoroughly illuminated the intricacies involved in creating On-Demand Test Automation Environments. Through a detailed exploration of setup and configuration nuances, we've delivered a valuable resource for practitioners aiming to enhance efficiency, scalability, and agility in their testing processes. As the technological landscape continues to evolve, embracing these insights will equip testing teams with the tools to adeptly navigate the dynamic realm of software development, fostering resilient and adaptable test environments that contribute to the pursuit of quality and innovation.

The automation of repeatable flows, as emphasized in this guide, brings about greater consistency, significantly reducing the likelihood of errors. This, in turn, results in a swifter time to market. Teams incorporating automation for establishing on-demand test environments not only benefit from increased efficiency but also lay the foundation for a more robust and streamlined software development lifecycle. The integration of automation proves to be a pivotal strategy in meeting the demands of a fast-paced and ever-evolving technological landscape.

References

[1] "Why environment provisioning is a key part of DevOps?," Clarive, Mar. 19, 2018. https://clarive.com/why-environment-provisioning/

[2] A. Ghosh, "Containerized Automated Tests in Azure DevOps - Ashish Ghosh," Medium, Dec. 10, 2020.

[3] Whaiduzzaman, M., Haque, M. N., Karim Chowdhury, M. R., & Gani, A. (2014). A Study on Strategic Provisioning of Cloud Computing Services. The Scientific World Journal, 2014. https://doi.org/10.1155/2014/894362

[4] J. Sandobalín, E. Insfran, and S. Abrahão, "End-to-End Automation in Cloud Infrastructure Provisioning," unknown, Sep. 06, 2017. https://www.researchgate.net/publication/318040301_End-to-End_Automation_in_Cloud_Infrastructure_Provisioning