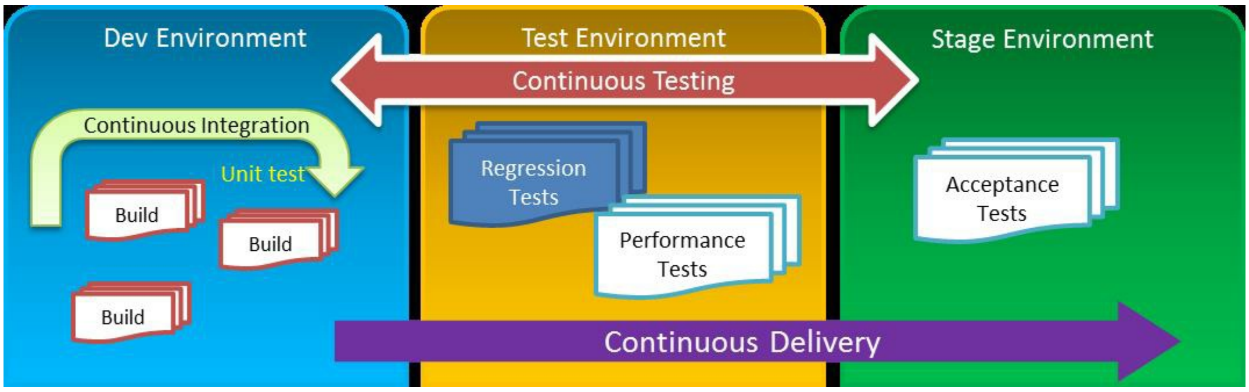
Automated Test Environments for DevOps

capgemini.com/algo.html

(1) Test Environments Management: Key Challenges

A testing environment is a setup of software and hardware in which the testing team tests a new software build. A test environment consists of pre-production or staging environments, and is generally a downgraded version of a production environment to help uncover pre-production defects.

Building and maintaining a test environment is important. Often, dedicated test environments are maintained at various stages like developer test, dedicated test, integration test, and pre-production or business readiness test.



In rapidly changing business, an IT organization has to align the IT strategy to business strategy and bring agility into the IT delivery. It is increasingly difficult to manage complex IT infrastructure dependencies, and manual intervention becomes unavoidable. Time and again, organizations consider management of non-production environments a secondary priority. This leads to unorganized and ad-hoc management of test environments and increased operational and maintenance costs for organizations. In addition, identifying and addressing environment-related defects becomes a major concern for QA teams. Effective and efficient management of test environments with structured automation can deliver significant benefits along with substantial cost savings to the organization.

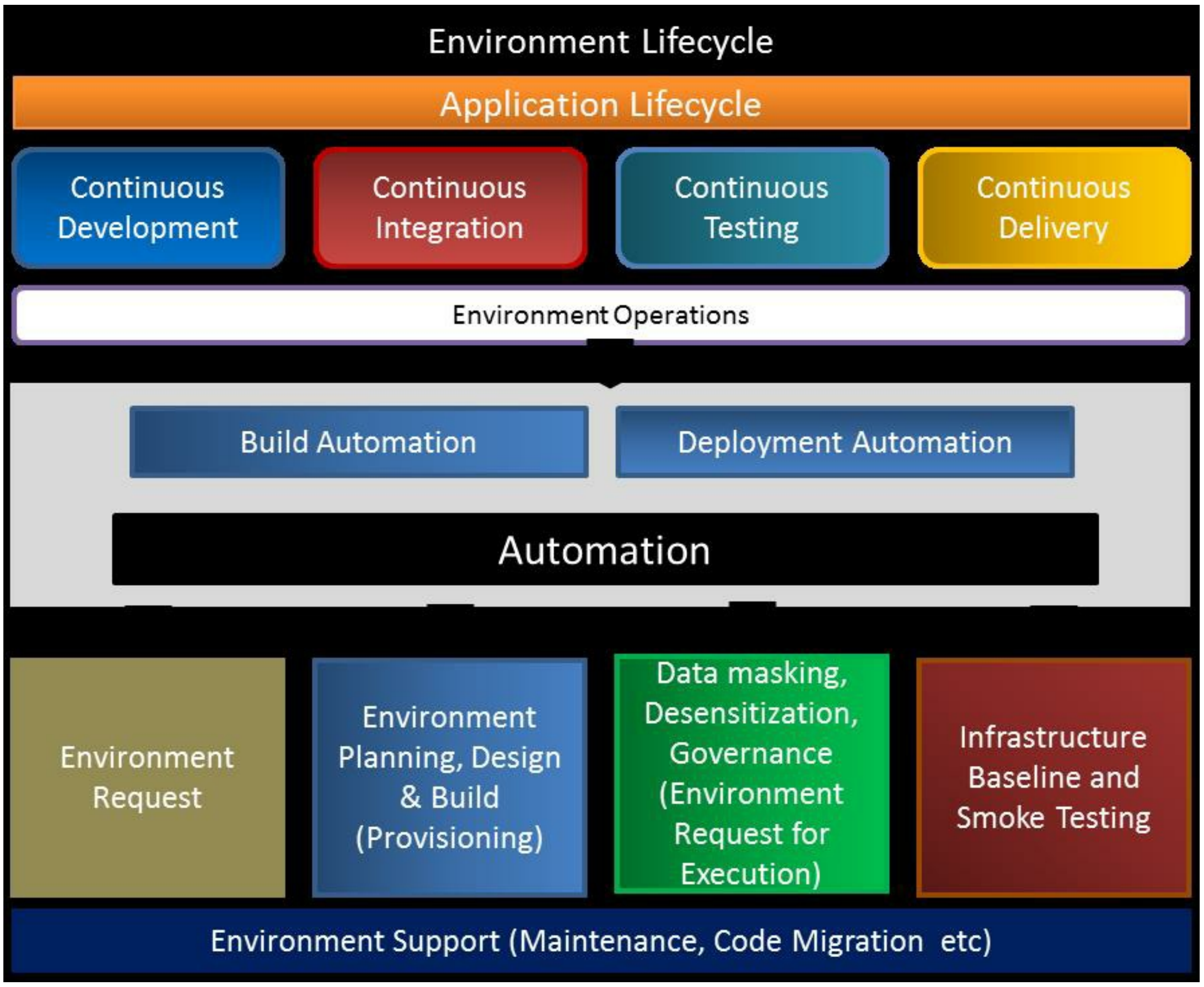
2. Test Environment Automation Approach for DevOps

Typically, release management is a complex and time consuming process. Even if applications are provisioned using templates, the infrastructure provisioning is often done manually. In a complex DevOps continuous integration environment requiring reduced cycle time for delivering and testing enterprise applications in DevOps methodology, the manual provisioning of environments is not sustainable. Adopting best practices and leveraging automation can drastically improve the cycle time and reduce the time to release without impacting compliance and standards.

An automation framework for environment management in DevOps lifecycle includes building and deploying automation for Continuous Development, Continuous Integration, Continuous Testing, and Continuous Delivery. An automation framework as illustrated below caters to automation in the following areas:

1. Environment request process
2. Environment planning, design, and build provisioning
3. Data masking, desensitization, and governance for environment request execution
4. Infrastructure baseline and smoke testing.

Environment support activities (e.g., configuration maintenance and code migrations) can also be automated.


Environment Lifecycle
Application Lifecycle
Continuous Development
Continuous Integration
Continuous Testing
Continuous Delivery
Environment Operations
Build Automation
Deployment Automation
Automation
Environment Request
Environment Planning, Design & Build (Provisioning)
Data masking, Desensitization, Governance (Environment Request for Execution)
Infrastructure Baseline and Smoke Testing
Environment Support (Maintenance, Code Migration etc)

A focused approach towards environment automation can reduce the dependencies involved in manual infrastructure provisioning. Test environment automation approach should support the following activities:

1. Environment provisioning, configuration, testing, deployment, and operational management
2. Test data management and related compliance requirements
3. Proactive monitoring of environment and self-healing for repeat incidents
4. Configuration management with auto-discovery for environment asset management and licensing

5. Patching and upgrading of infrastructure components compliant with enterprise policies

The approach to test environment automation varies across organizations. Common elements of this approach include a combination of pre-configured tools and scripts supporting the entire environment management lifecycle. Once the initial standardization is done, all the inputs, outputs, check points, and failure points in the tasks identified are captured. All the captured items are marked as candidates for local or centralized automation.

- **Localized automation** involves automation limited to specific components such as database refresh, operating system patch updates, and others.
- **Centralized automation** includes the orchestration and integration of various systems in the environment such as test environment provision requests received through a tool that needs to be executed by using a deployment tool and following standard operating procedures.

A robust automation script or workflow would lead to the reusability of the scripts that could be used in the future to speed up the provisioning process.

3. Benefits of Test Environment Automation

The benefits of test environment automation cover the range cited below:

- 30-40% reduction in time to market thanks to an improved provisioning cycle time
- Up to 50% reduction in manual health check efforts
- 20-30% enhanced environment efficiency through 30% reduction in IT service request fulfillment
- 30-40% reduced cost of operations

Additional qualitative benefits of environment management automation include:

- Elimination of errors due to manual interventions and delays due to dependencies
- Zero tolerance with respect to continuous availability of the environment
- Consistent and accurate environments through automated configuration updates
- Optimum utilization of infrastructure assets through better license management
- Availability of the environment as a service model

4. Test Environment Metrics to Consider in Release Management

Key metrics to be considered for support and management of test environments during release management fall into three main categories:

- ***Environment efficiency:*** Frequency of environment usage and frequency of conflicts arising out of environment demand. The environments posing regular demand conflict should be analyzed for impact on release plan. The analysis outcome (e.g., test execution stoppage of a release due to the non-availability of the test environment) could help in resolving the conflicts.

- ***Environment Stability***: The release cycle or schedule is affected in the case of test environment outages during a release cycle. We need to analyze the outage data and identify the events that impact the stability of specific test environments and pinpoint those events. The analysis could include the frequency of the environment outage due to project specific internal factors and external factors. Another area to investigate is the frequency of resorting to a manual work-around due to a specific environment issues or outages. Based on this, a plan needs to be created to achieve the critical path for release.
- ***Environment Agility***: The speed in which an environment can be created and configured is a measure of its agility. The tasks that need to be modeled and captured in a tool that understands how environments affect releases include environment retooling so as to match other code branches or releases and system or build validation prerelease.

5. Test Environment Automation Tools

Some of the commercial tools in test environment automation area include the following:

TEMS: Plutora's Test Environment Manager (TEMS) software provides comprehensive pre-production environment management capabilities to service all environment management functions.

Chef: Chef is a configuration management tool used to streamline the task of configuring and maintaining a company's servers, and can integrate with cloud-based platforms such as Rackspace, Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, and Microsoft Azure to automatically provision and configure new machines.

SmartFrog: SmartFrog is an open-source software framework, written in Java, which manages the configuration, deployment, and coordination of a software system broken into components. These components may be distributed across several network hosts.

SSH: Secure Shell, or SSH, is a cryptographic (encrypted) network protocol for initiating text-based shell sessions on remote machines in a secure way. SSH provides administrators with a secure way to access a remote computer for the launch of an instance to test a project version. SSH protocol is widely used in test environment automation.

If development and test environments live inside a virtual machine (VM), then launching a project version will need SSH authentication keys. Recreating the editor and command line sessions every time you develop and test is a waste of time. A terminal multiplexer allows you to create persistent development sessions running multiple commands in different panes once the SSH key are supplied.

Puppet: Puppet is an open source configuration management utility. It runs on many Unix-like systems as well as on Microsoft Windows, and includes its own declarative language to describe system configuration.

ServiceNow: ServiceNow is a platform-as-a-service (PaaS) provider of Service Management (SM) software for the entire enterprise. ServiceNow specializes in IT service management (ITSM) applications based on the ITIL standards.

6. Conclusion

In conclusion, test environment automation allows organizations to streamline repetitive tasks and activities and reduce overall operation efforts. It follows local and centralized automation approaches to gain more benefits in reducing release time. As a result, it helps organizations to free up resources and then deploy those resources on other critical activities that aren't developing automation assets.

Authors

Renu Rajani, Vice President, Capgemini India, renu.rajani@capgemini.com

Ranganath Gomatham, Solutioning Program Manager, Capgemini India, ranganath.gomatham@capgemini.com