WHITE PAPER: APPLICATION RELEASE AUTOMATION | JUNE 2014

Application Release Automation with Zero Touch Deployment™

Eran Sher
Application Delivery

ca technologies

Executive Summary

Challenge

Today's agile organizations pose operations teams with a tremendous challenge: to deploy new releases to production immediately after development and testing is completed. To ensure that applications are deployed successfully, an automatic and transparent process is required. We refer to this process as Zero Touch Deployment™.

This article reviews two approaches to Zero Touch Deployment—a script-based solution and a release automation platform. The article discusses how each can solve the key technological and organizational challenges faced by agile organizations when they set out to implement an automatic deployment system.

The article begins by recounting the business and technological contexts that drive agile organizations to seek deployment automation solutions.

How to Automate Deployment in an Agile Organization and Get Results

The highly competitive environment of the enterprise software industry has forced organizations to adopt software production methodologies that better ensure rapid delivery of new services to production.

In order to meet business objectives, organizations must fulfill customer expectations for constant improvement and innovation and establish short feedback loops between the market and the development teams.

Development teams discovered that established software development methods were often unfit to cope with the increasing pressure to innovate and create new functionality. One solution that arose was designing new software development methodologies around the concept of rapid software releases.

These development frameworks—broadly referred to as "agile" methodologies—set out to shorten the software development lifecycle by implementing an iterative and incremental approach to the process.

The Agile Iteration: A Micro-Development Lifecycle

The basic principle of iterative and incremental development is that software development moves forward in small cycles, or iterations. Each iteration is a micro-implementation of the general software development lifecycle. It starts with planning, continues to development and testing, and ends in deployment. The initial iterations implement a subset of the software requirements, and the subsequent iterations gradually enhance the evolving application.

This cyclic approach enables organizations to be more receptive to their customer base, as user feedback on current iterations impacts the development course of future ones. The business value of the application increases because its functions are constantly being evaluated, modified and perfected. Also, the iterative approach can have the significant advantage of allowing the application to generate revenue at a much earlier stage than traditional waterfall-like development methods.

The iterative approach spans the complete development process, from design through development and testing and finally to deployment. However, this transformation is not easy to accomplish. Faster methods need to be devised to carry out the testing of the new functionality, its integration into the general build, and its deployment to pre-production and production environments within the confines of the single iteration. For the testing and integration stations, a solution comes in the form of Continuous Integration (CI).

Continuous Integration: Extending the Agile Methodology to Testing and Integration

Continuous integration uses automation to apply testing and integration at the single iteration level. Whenever a developer commits a new code to the version control system (preferably at least once a day), the CI server automatically runs a new build and applies a full range of automatic tests, including unit tests, component tests, integration tests, and functional as well as non-functional acceptance tests. Manual tests are confined only to those testing aspects that cannot be performed by machines. Problems that are identified as a result of failed tests can be fixed by the development team before the latter moves on to the next iteration. This depiction is schematic, of course, but it captures in broad strokes the reality in many organizations, whose business survival depends on their ability to release to production a constant flow of new functionality.

However, a software development cycle does not end with the testing and integration stations. To complete the agile development cycle, organizations have to extend the iteration to include the deployment of the new functionality to all environments in the application lifecycle. But although deployments are, at least in theory, a part of the CI framework, it seems that this last phase of the software development lifecycle is also the last to adapt to the agile ecosystem.

The Bottleneck Simply Shifted to Operations

Nowadays, organizations realize that they cannot adopt an agile methodology for development and testing and then stop short before extending it to deployment. Even when agile methodology is executed to perfection and every iteration produces well-working and integrated functionality, it is not effective if it cannot be deployed in time. An overwhelmed operations team cannot handle the flow of new updates and the bottleneck that overloaded development and testing simply shifts to operations.

As it is, operations teams are facing complicated challenges without even taking into consideration the increasing rate of deployment events. Deployment flows have become more complicated, with multiple tiers and service-oriented architectures. Physical and virtual environments are simultaneously supported and maintained, and the number of deployed servers has increased exponentially since the introduction of virtualization and cloud-computing technologies. In the context of this business reality, the adoption of agile methodologies and continuous integration makes matters worse by adding additional complexity to an already complex organizational and technological environment.

In many organizations, the delayed adaptation to the agile mode of work in operations undermines the overall objective of an agile and iterative framework. Instead of being deployed in a dedicated release, the products of single iterations are packaged with products of other iterations. This unfortunate outcome counteracts the basic tenants of agile methodologies, since new functionality is waiting for deployment instead of running in production, generating income and being reviewed by customers.

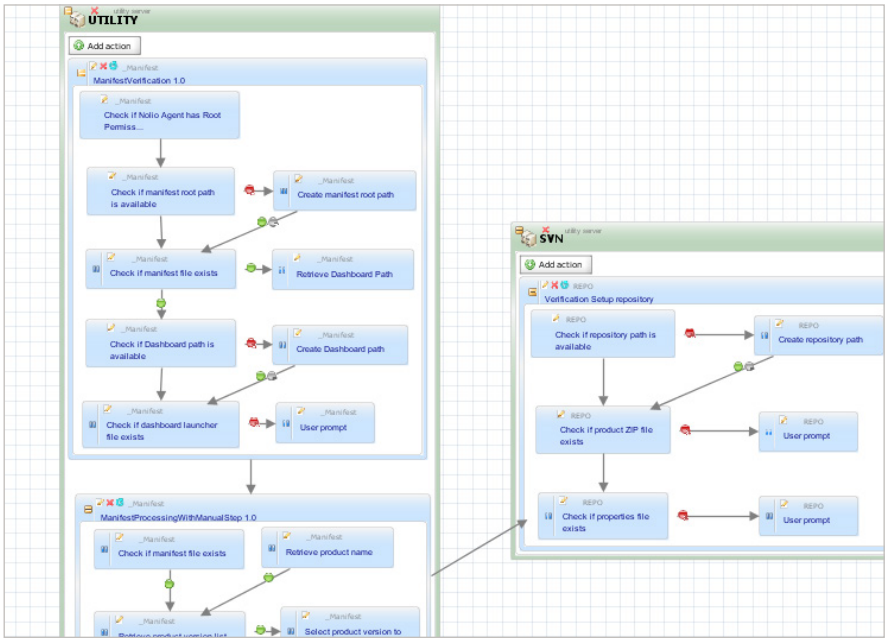
The Objective of Zero Touch Deployment in an Agile Organization

An agile organization, then, poses one tremendous challenge to operations: the ability to deploy new releases to production, where they can be reviewed by users and possibly generate revenue, immediately after successfully completing development and testing. In other words, the deployment process should be fully automatic and transparent as soon as it is activated. We refer to this integration of the deployment station to the iterative and incremental approach that characterizes the agile development and testing stations as Zero Touch Deployment.

In its extreme form, Zero Touch Deployment depicts a process whereby release to production is automatically triggered by the successful promotion of a new build from the acceptance testing station. Tailor-made deployment manifests are used to enable Zero Touch Deployment for complex releases. Manifest deployments separate the dynamic elements of the application release from the fixed processes. This involves storing all dynamic, changeable elements of a deployment in a manifest, including such details as which application resources should be taken, their specific location and the version. These details can be changed quickly and easily per release. The simple fixed processes, directing how the release process should be executed, are kept separate and can be employed and repeated time and time again.

Figure 1:

Consumption of Manifest in CA Release Automation.



The Complexities of Deployment Automation

However, automating the deployment of large-scale data center applications is not a simple task to achieve, especially if we set out to provide a real and complete answer to the deployment needs of agile organizations. Whether it is comprised of a collection of deployment scripts or managed by a specialized application release automation platform, the automatic deployment system has to cope with numerous complexities on multiple levels.

One complexity arises from the different types of deployment events agile organizations produce. That means that different automatic deployment processes have to be created and maintained to handle deployment events ranging from a full installation of the application to a minor update or even a single configuration file update. An automatic deployment system must be able to simplify application deployment tasks and mitigate risks, turning complex, manual operations into reliable, repeatable and error-free processes. This accelerates the time-to-release of individual releases and enables the simultaneous deployment of multiple applications.

Moreover, each of these deployment events can involve more than one application tier. A minor deployment event, such as a bug fix, can involve changing a parameter value in the configuration file of a middleware server as well as applying a patch to a library file in an application server. A major deployment event, such as installing an application from scratch, usually involves simultaneously deploying the application to a database server, application server and/or a Web server.

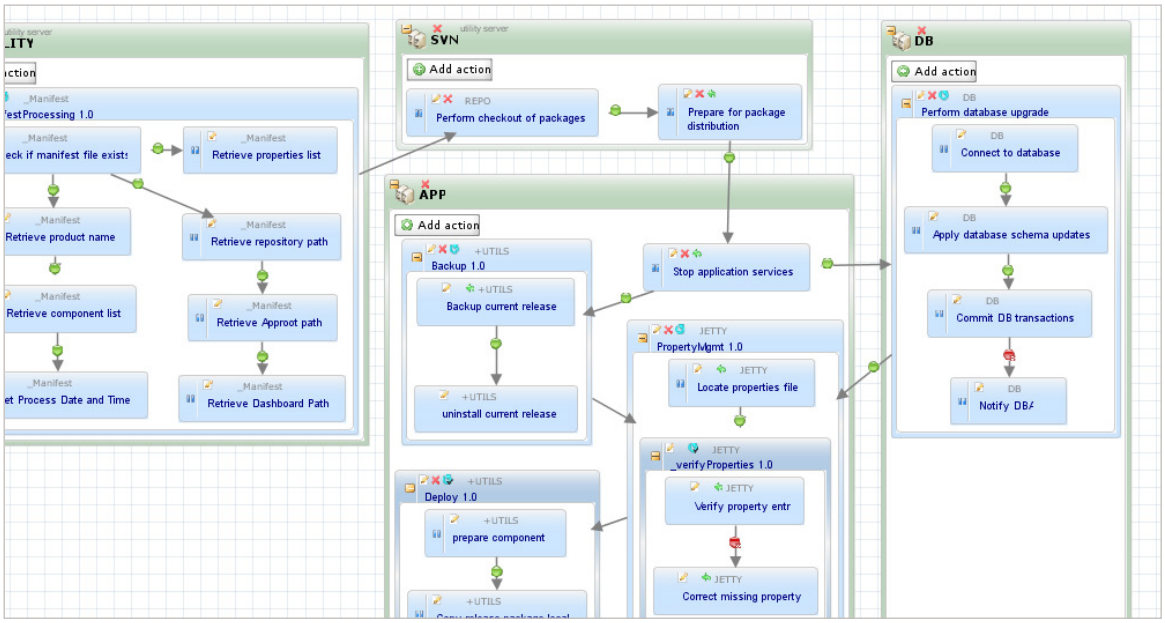
The implications of this on any automatic deployment process are that it must be comprised of separate deployment flows, one for each application component or tier. Therefore, to enable the deployment of multi-tier applications, any automation deployment system must be able to capture the tier-based architecture of the deployed applications. A script-based system would map a separate deployment script to each tier. A dedicated automatic release platform can employ a graphical approach to enable the user to visually represent and create the different tier-dedicated flows of the deployed application.

Co-dependent relations between tier-specific deployment flows represent another complicating factor that the automatic deployment process has to be able to contain. Deployment flows are said to be co-dependent when one flow can start execution only after another flow has run its course; for example, the website deployment flow cannot start before the database deployment flow is completed.

An automatic deployment mechanism is expected to be able to coordinate the execution order of co-dependent tier-specific deployment flows. A script-based system can use a build management tool to coordinate between the execution of the different tier deployment scripts. An automated release platform can enable the user to specify the co-dependent relations by drawing connectors between nodes representing the different tier-dedicated flows.

Figure 2:

Multi-Tier Dependency During Application Deployment



In addition to the challenges inherent in the multiplicity of deployment events and flows and in their execution management and coordination, an automatic deployment system has to manage multiple deployment destinations. The rise of online applications serving a global user base, accompanied by the spread of virtualization and cloud-based technologies, has increased by tenfold the number of instances targeted by a single application release. Consequently, an automatic deployment system has to execute one deployment event to hundreds or thousands of servers residing in multiple data centers.

To be able to manage application deployments to a multitude of servers, each with its own configuration information, automatic deployment systems must be able to support clean separations between the deployment process and the configuration information. For each deployment destination, an instance of the same deployment process is executed with a unique configuration information set.

To support such a separation, a script-based system has to set up a strict configuration management policy. An automatic release platform, on the other hand, has a dedicated grid that maps and represents the environment in which the application is deployed. Each physical, virtual or cloud server is represented graphically in the context of its environment, as well as in its relation to the deployment process. The configuration information of each server is represented, stored and maintained in the context of this representation.

Automating deployment along the principles of Zero Touch Deployment can help organizations fulfill the objective of the deployment station in an agile framework—namely, to deploy new code to production immediately after it has completed the development and testing stations.

Automatic Deployment: Not Just to Production

However, an automatic deployment system that is comprehensive, adaptive and robust enough to face the challenges posed by an agile organization must be in a constant process of tuning and perfection. For this reason, it is imperative not to limit the use of the automatic deployment process to the deployment station at the end of the iteration. On the contrary, the same automatic process that is used to deploy to production should be used to deploy to the development and to the testing environments throughout the iteration. This directive ensures that the deployment process is tried extensively before release day to avoid any last-minute surprises.

The development team should use the automatic process whenever they need to deploy the application to their local development station, and the testing team should also use it when they create their testing environment. This way, any negative impact of the new code on the automatic deployment process is discovered early, when it is relatively easy to correct.

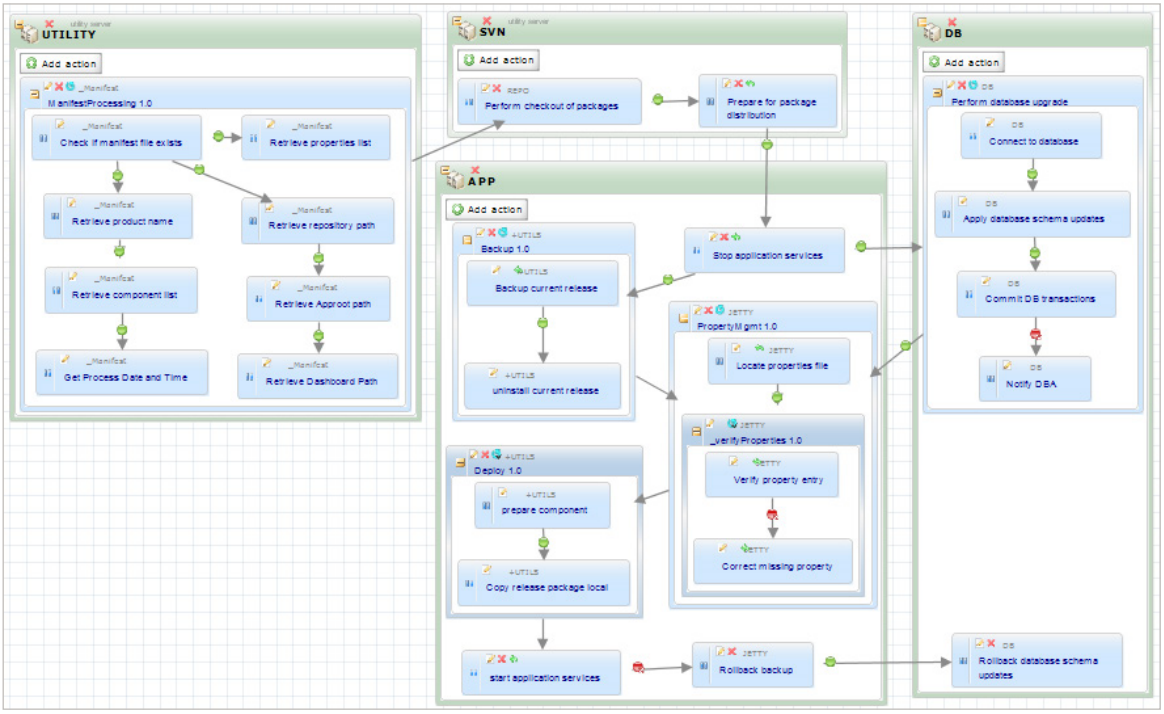
Because development and testing environments are different from production environments, some modifications should be implemented to make these environments more production-like. Such modifications may involve some costs, but these expenses are minimal in comparison to the cost of unsuccessful release days.

A Deployment Language Common to All

This brings us to the question of who "owns" the automatic deployment system. Deployment used to be the sole responsibility of operations, but the reality of agile organizations has made deployment the business of everyone involved in software production. Since developers, testers and operation personnel alike must use the automatic deployment system in their daily work, they must share the knowledge regarding the technology, terminology, work practices and release strategies that are involved in everyday running and maintenance.

Figure 3:

High-level Graphical Representation of Application Deployment Process



The implication of this organization-wide shared ownership on the automatic deployment system is that it should be accessible to everyone participating in the development lifecycle. In the case of a script-based system, only personnel with coding skills can develop and maintain the deployment scripts. This limits the accessibility of the system from the point of view of the organization. However, in the case of a dedicated release automation platform, a task-specific GUI enables the user to define, configure and maintain the automatic deployment processes without recourse to script coding. In this context, there is a good chance for the formation of a common language between development, testing and operations, and the successful integration of the deployment station in the agile development lifecycle.

For more information, visit **ca.com/releaseautomation**

Connect with CA Technologies at ca.com

CA Technologies (NASDAQ: CA) creates software that fuels transformation for companies and enables them to seize the opportunities of the application economy. Software is at the heart of every business, in every industry. From planning to development to management and security, CA is working with companies worldwide to change the way we live, transact and communicate – across mobile, private and public cloud, distributed and mainframe environments. Learn more at **ca.com**.

CS200_87158_0914