A Broadcom Company
technologies

White Paper

Automic® Continuous Delivery Automation

Packaging, workflows, and a generic deployment model.

Complexity Changes Everything

Applications and services are made up of more components and integrations today than they were just a few years ago. Coupled with a much more competitive business world, this complexity has changed everything for both development and operations.

Dev is pressed to deliver more functionality at a faster pace, focusing on small, incremental changes and using agile methodologies and practices, such as continuous integration.

The ops environment is much larger these days, with many more servers and services than ever before. The production environment, which used to be all physical, is now almost entirely virtualized, so ops can spin up servers in an instance. Ops is also managing many more resources today than ever before.

As development focuses on fast delivery by chopping up services, developing more in parallel, and making applications work as a whole, more configuration steps are required, and more interdependencies are created.

Furthermore, with agile methodologies, developers are spending a lot less time on internal projects that could have simplified some aspects of the complexity. Ops hates to fix developer problems, and developers do not want to spend time on configurations and deployment. Over time, this has become a major finger-pointing exercise: *It's not my code, it's your machines* versus *It's not my machines, it's your code*. The DevOps movement puts forth the idea that dev should think like ops and ops should think like dev. This movement has addressed such problems and invites ops to participate early in the development cycle.

The use of advanced configuration management solutions, such as Puppet and Chef, also helps greatly. But we are never going to be home-free until we tackle the last mile. This white paper focuses on automating application deployment across DevOps, the capabilities that are required, and how those capabilities apply to either development, operations, or both throughout the lifecycle.

Growing Application Matrix and Losing Control

The application lifecycle is an ongoing iterative process where applications are constantly developed, tested, patched, upgraded, and re-developed across a number of stages, including development, functional testing, integration testing, staging, and production. While stage names and numbers vary between applications and enterprises, the process is roughly the same. Moving from stage to stage always involves deploying parts, or possibly the entire application, to a new set of servers, also known as an environment. Additional requirements include uninstalling previous versions, in some cases, or installing to a parallel location in others, such as testing.

The DevOps challenge increases every time a new component is added. With every variation in a configuration or target environment, there is an ever-growing matrix of components, environments, and deployment requirements. With a vast array of groups developing various applications across different locations that ultimately have to directly, or indirectly, integrate with one another, many organizations simply lose control. The past few years have seen a number of publicly visible failures of mission-critical systems across all verticals. Banks, communication service providers, and Internet juggernauts have been brought down by uncontrolled or unexpected changes. Headlines, such as *Faulty Deployments* and *Botched Upgrade Processes,* explained how these failures occurred.

However, scale is also a significant issue, specifically in relation to the magnitude of servers in the operations environment. People need tools to manage configurations and baselines on a large scale, as evinced in the emergence of Puppet and Chef, among others. A critical issue is being tackled with these tools, assuring basic infrastructure specs in even the largest-scale environments. But while these solutions are excellent for standardizing large server environments, they are less suitable for application processes that cross environmental barriers or are cross-tier by nature. You could say that configuration management tools use a declarative (after state) scheme, while Automic Continuous Delivery Automation uses a methodology (do it this way, not that way) scheme. One is more environment-bound (*What exactly do I need on each server in my environment?*), and the other is more lifecycle-bound (*What are the steps necessary to move this application version/patch/release from this environment to the next?*). Automic Continuous Delivery Automation tools provide control, efficiency, and effective use of the DevOps toolchain across all layers of the IT stack, and for all environments where applications and their dependencies are deployed. Automic Continuous Delivery Automation tools combine modeling and automation capabilities that are specifically designed to fit into any application lifecycle and management technologies.

Application, Pipeline, and Environment Models

An Automic Continuous Delivery Automation solution allows you to model an automated software delivery process that will enable collaborators to easily follow the progress of builds coming out of continuous integration processes through multiple stages of testing and deployment. Automic Continuous Delivery Automation solutions let you abstract physical artifacts, environments, and deployment steps into models that can be more easily applied and adapted to new changes, criteria, and situations.

Application Models

Modeling applications are a fundamental requirement of any serious Automic Continuous Delivery Automation solution. Applications are made up of numerous binary artifacts, configuration settings, scripts, and dependent services. Artifacts originate from multiple sources (including repositories, build servers, file shares, and FTP servers). And each artifact requires exact versions to match the current deployment. Designing a logical model of your application provides a layer of abstraction between all the physical bits and bytes and your deployment pipeline, enabling the same underlying automation mechanics to execute in any environment. Some continuous delivery automation or deployment automation solutions directly map physical artifacts to modeled application components or environment deployment targets. This means each new app version will require a continual redesign of your logical model and require you to redesign your deployment pipeline. This one-to-one mapping (not modeling) does not provide the significant benefits one expects to leverage while troubleshooting, since the automation mechanics themselves need to be evaluated as part of your problem resolution efforts.

A proper application model will provide consistency and repeatability across all environments and release stages. This reduces audit concerns and troubleshooting efforts. Additionally, a logical application model will allow the automated deployment mechanics to execute on a centralized development server or container, or across distributed production servers or containers.

Application models should then provide a packaging construct to act like a bill of materials (BoM) for each deployment pipeline, representing a mash-up of the different versioned components that should be deployed together. Packages should also provide a state flow that allows you to accept or reject them for promotion. This will provide key insights into the productivity and efficiency of your continuous delivery practice and provide the forensics for process improvement.

Pipeline Models

Automic Continuous Delivery Automation solutions employ two kinds of automation mechanics: declarative (desired end state) and workflow (how to get to the end state). Declarative models prescribe how the deployment process will occur without the software vendor allowing flexibility for DevOps teams to determine their own best practices or adapt to technology changes. Workflow models, by contrast, provide DevOps teams with complete flexibility and visibility into how software will be delivered.

Workflows are the workhorses of scalable, enterprise-ready Automic Continuous Delivery Automation solutions. They replace the manual tasks that are required to install, upgrade, patch or roll back an application. And they can be reused, time and again, across any environment the application lives in. Mature Automic Continuous Delivery Automation solutions allow you to model workflows visually on a canvas, much like a Microsoft Visio-type block diagram. Workflows are assembled by dragging and dropping steps or tasks from a huge library of existing behaviors and integrations for most of the common application hosts.

The examples below illustrate the breadth of infrastructures and tools for which out-of-the-box workflows can be assembled:

- Application servers such as Oracle WebLogic, IBM WebSphere and JBoss/IIS
- Database servers such as Oracle and Microsoft SQL Server
- Integration servers such as BizTalk
- Container technologies such as Docker, Kubernetes and LXC
- Application servers such as WebLogic, WebSphere and JBoss/IIS
- Cloud providers like AWS, Microsoft AZURE and others
- OS commands, package managers, and source control systems such as GitHub and Bitbucket
- CI tools such as Jenkins, Bamboo and Travis CI

Out-of-the-box integrations provide a standard and quality-assured way to consistently deploy new changes to various servers. Out-of-the-box integrations can also be executed using the different credentials (impersonation) that are required during the process, without compromising security. These integrations are also automatically audited 100% of the time.

Important workflow examples are described below:

Generic Workflows

Generic workflows should be completely decoupled from any environmental setting, credential data, or permission data, as well as from the content that the workflows execute. This allows the same workflow to be executed across any environment, from test to production.

Version-Controlled Workflows

Just as application components and services are developed by multiple teams in parallel and result in different versions, so can workflows. In fact, the same teams that develop code can also develop workflows. The only way to work in parallel across multiple teams is through a version control system. It is therefore a critical ability for your Automic Continuous Delivery Automation platform to enable multiple teams to collaborate and work effectively with multiple deployment workflows and multiple versions.

Deployment Workflows

You need deployment workflows that provide simplicity along with sophistication. Simple serial automated processes are overly optimistic and too rigid to handle the complexities, nuances, and fault tolerance that you need for the modern enterprise technology portfolio. Workflows should be able to handle retries, if-then-else branching, partial failure or partial success processing, runtime decision making, workload distribution and much, much more. Enterprises need sophistication and flexibility along with ease of use.

Environment Models

Many errors are caused by differences in the environment settings of a particular deployment, and not by workflow bugs. The problem is that software and applications are susceptible to even the smallest variations in environment and settings. A simple example is the differences in the file folder structure between QA and production servers. More subtle differences might include the port configuration of a service, execution credentials, and even the configuration of the environment, such as cluster sizes. Accounting for these important variables during deployment execution is a tedious and error-prone process. Users have to verify and adjust for all of them on every target machine that they touch, as well as the system as a whole. When you consider tens or even hundreds of servers that need to be updated, the matrix quickly becomes unmanageable by manual methods, and scripting requires hard programming that is frequently done on a one-off basis.

Automic Continuous Delivery Automation has a built-in model that lets operators and developers control runtime settings in a single place. The system will propagate the correct settings to the correct executions at the correct moment, automatically. The model takes care of the variations existing between environments (such as the number and types of servers), server settings (such as directory and version differences) and configuration data (such as variable values). The model includes deployment targets, profiles, logins and more. The model not only assures the correct settings are propagated at the right time to the correct execution, but also plays a crucial role in keeping workflows simple and manageable. In fact, not having a built-in model will result in very complex workflows and having to store the runtime settings in external sources, such as XML or databases. This can be both insecure and complicated, as the user needs to read, parse, and branch execution flows as part of the workflow.

Conclusion

Automic Continuous Delivery Automation is the only way to gain true control of a complex matrix of applications, releases and environment variations. The inability to fully control and audit change, deployment, and release processes results in IT failures that can cost the business millions and make DevOps unattainable. Automic Continuous Delivery Automation is the last hurdle in an enterprise continuous delivery practice.

To overcome this hurdle, Automic Continuous Delivery Automation solution provides these three critical capabilities:

- Application packages and promotion paths, so that you can be confident that what is deployed is correct.
- Workflows, providing a standard and quality-assured way to consistently deploy new changes.
- Environment models, ensuring the correct setting and configurations are applied to each environment.

While there are other factors to consider, only by ensuring that your solution encompasses the above capabilities can you be certain that you will be able to meet the ever-increasing demands of the business to go faster, for less cost, all while remaining in control.