Scaling Continuous Delivery Deployments

A whitepaper on how to scale Continuous Deploy with DeployHub

By Tracy Ragan, CEO, DeployHub

Contents

Summary

Scaling your Continuous Delivery process is highly dependent upon the ability to scale your Continuous Deployment. DeployHub is a highly scalable software deployment solution that can be called by your Continuous Delivery workflow, orchestrated by Jenkins or any other CI Server. Jenkins manages a CI Job which is built upon scripts that do the heavy lifting. When scaling Jenkins, Jenkins "Slaves" are used to transfer files from point A to B, using a delivery script. This model does not scale to larger test and production environments. This whitepaper reviews how DeployHub can be used to perform the heavy lifting of the deployment step in your Continuous Delivery process, offloading Jenkins and allowing your Continuous Delivery process to scale across the lifecycle.

The Challenge

Development teams define their software deployment process for the purpose of moving binaries to a handful of development servers. Written in a popular scripting language and called as part of a Jenkins Job, the development deployment process works quite well. When this same process is handed off to production and test environments, it often fails to scale and struggles to deploy to hundreds, or thousands of end point servers. In addition, Continuous Delivery increases the frequency of software deployments, demanding faster, easier and more transparent methods of releasing code. Software deployments need better automation with less overhead in

When is it Time to Automate Deployments

"Developers will use Jenkins with no issue, while testing and production begins experiencing the symptoms of an over loaded process."

Development, Testing and Production use Jenkins with different levels of intensity. As the process is shared across environments, problems begin to occur. A deployment executed by the development team needs to be tweaked to support the needs of testing and tweaked again for production.

Developers will use Jenkins with no issue, while testing and production begins experiencing the symptoms of an over loaded process.

Jenkins was not intended to be the only tool in the toolbox. When it is used beyond what it was design to do, it can become a bottleneck requiring extra work and troubleshooting. The solution is to allow Jenkins to orchestrate the process, with the heavy lifting achieved by solutions architected to automate application releases. This is the function of OpenMake DeployHub.

"Your software release process should not rely on older agent based technology."

Common issues found when using Jenkins as a software deployment solution include:

1. Jenkins requires Agents. When you are running deploys in development, you may already have a server that is configured with a Jenkins 'slave.' Moving up the lifecycle with a repeatable deployment process requires that a Jenkins Slave be installed on all test and production machines. Your software release process should not rely on older agent based technology. This will become even more obvious as Containers become the norm. .
2. Jenkins knows nothing about different types of Servers such as a database server, application server or load balancers. This level of intelligence is critical for automating software deployments. Jenkins does not automate software deployments. It executes an external process, often a script, to do that work.
3. Jenkins Servers run Jobs but have no concept of Environments (Dev, Test, QA) or Domains (East, West, etc.). Tracking what version is running in what Environment and in what Domain cannot be done without digging through log history and having a manual way to track where a server is.

"Jenkins lack the concept of security making it difficult to setup up different environments with different ownerships."

4. Jenkins relies on scripting to do the actual deployment; therefore your deployments can only be as smart as the script itself. There are deployment requirements that are not easily done with a script, such as
    - rolling a deployment backward or forward with version jumping and database updates,
    - automatically mapping components to servers as you move up the lifecycle dynamically adding more servers to your deployments,
    - calendaring and pipeline management,
    - coordinating deployments based on environments (groups of servers) ,
    - creating a *release* train that includes a deployment of more than a single application.

5. Jenkins lacks the concept of security making it difficult to setup up different environments with different ownerships. Lack of security controls and approval gates may work for development but does not work for testing and production control where deployments are more tightly managed.

Jenkins was not designed to be a production level software deployment solution. Giving Jenkins a little help in the area of application release automation will provide you with the best of both worlds. Jenkins can be the central point for orchestration and collaboration, while DeployHub can deliver the 'full strength' needed for larger deployment requirements.

Off-loading Jenkins

DeployHub was designed and written to support both the tactical needs of the development team and the strategic needs of the enterprise. Its agentless delivery technology makes it easy to expand or contract the number of endpoint targets required by a deployment. Developers can define the entire logic of a release, including database updates and server configurations. The release is easily passed to other environments, such as test or production, where the logic is assigned to the proper roles and automatically expands to environments with more target endpoints.

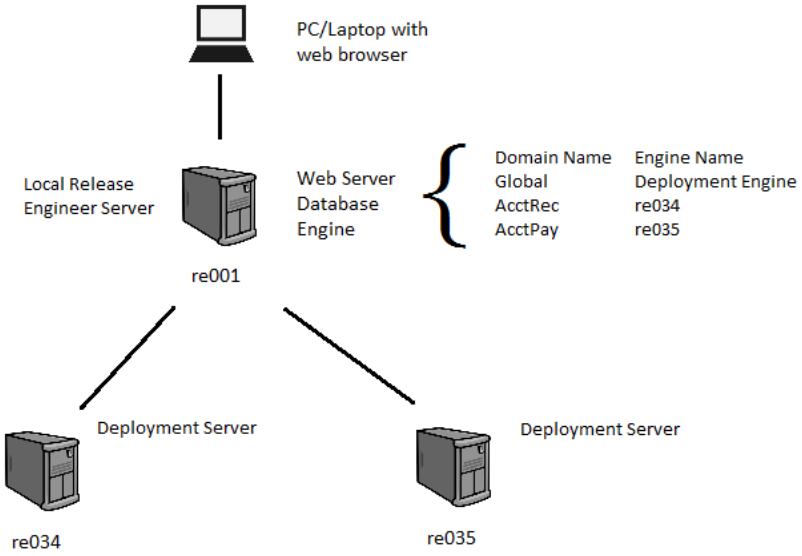
How DeployHub Scales

DeployHub can easily scale to environments that use thousands of endpoints. It does this by using 'engines' whose job it is to distribute the workload of the deployment. The Engine is used to deploy an Application to endpoints using:

Target Server: A server that will have an Application deployed to it.
Local Server: A server that has a full install of DeployHub including the Web Browser, Database, and Engine.
Deployment Server: A server with only the RE Engine installed on it. This is used to deploy Applications to Target Servers in networks other than the Local Servers and allows DeployHub to handle a release to thousands of endpoints.

"DeployHub streamlines the software release process from a strategic viewpoint while supporting the development team's need for independence and speed. "

Deployment Servers can be assigned to perform the heavy lifting of an application release based on a 'Doman.' This allows you to configure how the release will be scaled to handle large environments minimizing network traffic and accelerating the release process. The Deployment Server will execute the deployment logic managing a cluster of end-points, including those that are a mix of application servers, load-balancing servers or database servers.



In the above example, the Local DeployHub Server and both Deployment Servers deploy Applications located in individual Domains within this installation and can each deploy to hundreds of Target Servers. *RE001* is a full installation of DeployHub which includes the central database that contains all of the information concerning Applications, Domains, and other DeployHub objects as well as the Release 'Engine'. . *RE034* and *RE035* are Deployment Servers and have only the 'Engine.' The Deployment Servers report back to the Local DeployHub Server all log, history and meta data reporting. Applications are contained within Domains, and will be deployed using the assigned Engine running on the Local DeployHub Server or on a Deployment Server.

Giving Jenkins a Little Help

DeployHub includes a Jenkins plug-in that will allow you to define your DeployHub deployments as part of your Jenkins Job. A DeployHub *Application* or *Release* (multiple Applications) is deployed to an *Environment* which contains one or more *Servers*. An *Application* contains *Components*, which in turn contain *Component Items*, which access *Repositories* that contain all of the files that are to be deployed.

"The DeployHub plug-in allows Jenkins to call DeployHub as a step in a job. This enables Jenkins to control deployments to large server environments, without overloading Jenkins".

Both *Applications* and *Components* also contain *Procedures* and *Functions*, which can be executed before and after each *Application*, and each *Component* contained within it. This gives DeployHub a tremendous amount of flexibility, whether it's used alone or in tandem with Jenkins as a plug-in.

The DeployHub plug-in allows Jenkins to call DeployHub as a step in a job. This enables Jenkins to control deployments to large server environments, without overloading Jenkins. The DeployHub plug-in is displayed in the Jenkins Configure window, and provides a way to configure what is required from DeployHub. The basic configuration, which shows under the title "Use DeployHub to run a deployment", includes the following:

"Jenkins plug-in parameters indicate to Jenkins just how to execute your DeployHub deployment. "

Username and Password	Determines security access to the various objects within DeployHub, including the *Applications* and *Components* that are available from the Jenkins Plug-In.
Target Environment	A DeployHub *Environment* contains all of the *Servers* that will be deployed to.
Application	A DeployHub *Application* contains all of the *Components* that make up a deployment. It is deployed against an *Environment*.
Wait for Deployment to Complete	This tells Jenkins to wait until DeployHub is finished with the deployment before moving on to the next step in the Job. If this option is chosen, the remaining steps in the Job will only run if the deployment was successful. If left unchecked, Jenkins will continue to the next step in the Job as soon as DeployHub begins the deployment, and ignore the success or failure of the deployment.

Scaling DeployHub

Scaling DeployHub can be done based on stress testing that showed a 100MB file takes approximately 7.2 seconds to be distributed to multiple machines. Scaling DeployHub involves adding Deployment Servers to the distribution model to share the workload and keep deployment times to a minimum. On average it is recommended to use one Deployment Server per 1000 Target Servers to distribute the workload and keep overall deployment times under 1 hour.

Size of Deployment files	**Number of Target Servers**	**Number of Deployment Servers**	**Time Estimate**
100MB and Under	Under 500	0 (Local Server only)	Less than 1 hour
+100MB	500-1000	1	1 hour
+100MB	2,000-3,000	2-3	1 hour
+100MB	3,000+	1 Deployment Server per 1,000 Target Servers	1 hour

During stress testing, several deployments were made deploying a 100 MB file to 1000 Servers. Results showed an average time required to move a 100MB file to 1000 Servers was 2 hours or 7.2 seconds per Target Server. A deployment that placed the 100 MB onto a single machine took 18 seconds. This includes the processing done at the drop zone prior to the distribution. When deploying to more than a single Target Server, the drop zone processing is not repeated, reducing deployment times for all subsequent Target Servers. The wildcard for calculating the time for any deployment is the bandwidth of the network and the disk drives on each individual Target Server.

Stress Testing Methodology

Target Servers were used to deploy large files across a network which included the internet. It was decided that the Amazon Elastic Compute Cloud (EC2) would be used to run the deployments and gather statistics, since it is the most widely used cloud service. Ten Target Servers with no EBS volumes were used for the deployment. DeployHub was installed on a Windows 2008 Server also within the EC2 Cloud. DeployHub Server contained a typical install, which contains the DeployHub Engine, Database, and Web Server.

The 10 Ubuntu Linux EC2 Cloud Target Servers each had 100 directories on them, so that a total of 1000 directories were available. Each directory was mapped to a DeployHub Local Server, and all 1000 Servers were added to a single DeployHub Environment. A DeployHub Application consisting of a single 100 MB file was deployed to this Environment, meaning that the equivalent of 1000 servers were being deployed to during a single run, with each one having the same 100 MB file deployed to it. The original copy of the 100 MB text file was stored in a directory on the DeployHub Server, and a filesystem Repository within DeployHub was created to access this file for use during the deployment.

Because DeployHub uses a 'drop zone' for pre-deploy package processing, it only accesses the file Repository location once for each Component Item. All of the retrieved files were kept in the 'drop zone' on the disk of the Local Server for use anytime during the stress test deployment. This feature eliminates the need for accessing a Repository more than once during a deployment (any number of Repositories and their files can be accessed during a given deployment), which not only decreases their deployment time, but Procedures and Functions can access this area on the disk and manipulate the values stored there.

DeployHub is an agentless deployment system, which means the Local or Deployment Servers performing the deployment have full access to the Target Servers without a remote agent installation. The DeployHub Local or Deployment Server is limited only by the username and password contained in the Credential that is used to gain access to it (a Credential is a DeployHub object that contains a username and password).

Conclusion

Scaling your Continuous Delivery process must include the introduction of an Application Release Automation solution such as OpenMake DeployHub. Using DeployHub to take over the Continuous Deploy step of the Continuous Delivery process allows you to easily scale your deployments from Development through Testing and Production deployment.

On average, DeployHub takes approximately 7.2 seconds to distribute a 100MB file to a single Target Server. It can be easily scaled to handle production installation requirements with thousands of Target Servers using multiple Deployment Servers to distribute the Workload. DeployHub includes a Jenkins plug-in that allows you to continue to use Jenkins to orchestrate your Continuous Delivery process (CI Jobs), but hand-off the heavy lifting of software deployments to a fully automated process.

By allowing Jenkins to call DeployHub, you achieve the ability to scale to your larger environments without the use of 'slaves' or 'remote agents.' You also gain the ability to easily jump versions backward or forward, handle database updates, track inventory across the lifecycle with security and define approval gates with calendaring. These are critical features that Jenkins cannot deliver. By giving Jenkins a little help in this area, you can move forward to a more consistent, repeatable Continuous Delivery process.

DeployHub Differences

When evaluating ARA solutions, the following features should be considered and carefully evaluated:

Features	DeployHub	Open Source, Declarative Language solutions
Agentless Delivery	DeployHub does not require an agent be installed on every end target. This eliminates a substantial amount of overhead supporting and managing thousands of agents for the purpose of performing deployments. This was largely the downfall of earlier delivery solutions built around network monitoring systems. DeployHub addresses this issue directly by removing the requirement of Agents.	The large majority of ARA solutions require the management of Agents on each end target server. The reason for which these solutions are designed is the intended audience. For the most part, the audience is a single project team, which may not have an application that is deployed out to 400-500 servers, or more. In particular, Open Source solutions are designed for smaller project teams, and smaller environments, so the need for an agentless solution is not required.
Jump Version – move back or forward between any version	DeployHub allows you to jump forward to any release level or move back to any release level and handles all of the in between steps, including database updates.	While rollback is common in most solutions, the ability to incrementally roll forward is not. Important for test and production because they may not take all updates. This is when the database level and application levels can become out of sync. DeployHub handles these types of roll forward requirements.
Blueprint Planning- Ability to define and share components between, teams, releases and deployments	Reuse of all parts of a release is critical for the Enterprise. DeployHub allows for the sharing of *Components*, such as Websphere, across teams. It is defined once and used multiple times. *Application Versions* are made up of *Component Versions*, which allows for the management of reusable objects across teams.	Reuse is not often a consideration when the solution is designed to be implemented at the project team level. For this reason, most competing solutions cannot offer this level of component sharing and architectural blueprint planning.
Application Packages - Ability to package together any number of applications for a Release	Provides a method of collecting application versions and components to define a Release. This simplifies the management of the technology stack across the environment without this critical data being hidden in a manifest.	Most solutions, Open Source or Commercial, provide some level of packaging, but at the file level. A manifest file is coded that builds out the technology stack dependency. This also means that your manifest file may change between environments. This level of information needs to be dynamic and not managed statically.

Built in approval processes with appropriate approval gates	DeployHub incorporates approvals and gates into the release process. Security can be defined to prevent unauthorized users from performing a release.	Many solutions are not designed for users on both sides of the house – development and production administration. These tools were written as production administration solutions to simplify the execution of large releases. For this reason, the concepts of approvals and gates were not needed and are not part of the solution.
Multi-platform	DeployHub supports multi-platforms and multi-languages. • **IBM WebSphere** • **Jetty** • **Microsoft IIS** • **Microsoft SQL Server** • **Oracle** • **Sybase** • **Windows 2008, Windows 2012** • **Solaris, HP-UX, AIX** • **Ubantu, Redhat, SuSe, Debian** • **AWS, Kubernetes, Google, Azure** • **Tandem, Stratus, IBM iSeries, OpenVMS, Unisys, IBM 4690**	Most Open Source solutions are designed for a particular platform, with Linux being the most common. Open Source solutions are often build around Linux RPMs, which means that support for Windows may not be ideal. Additionally, many commercial solutions are written around the support of a single hosted web container, most commonly Websphere. This is due in part because Websphere is a complicated environment in which to perform deployments, so the commercial market delivered a solution. The problem is that for the enterprise customer, Websphere is not the only environment to support. The enterprise solution should be flexible enough to support Websphere, Weblogic, IIS, Oracle, SQL Server, etc. The ultimate goal should be a single solution to support any delivery target.
Interacts with Network Drives - Ability to read and write to different network drives	DeployHub does not require all the artifacts to be delivered to exist on the local drive where RE is installed. You can point to any location, on any network drive, to retrieve the artifacts to be delivered. The location of the artifacts are defined for each *Component* defined for the *Application Package* which means that different Components can be managed in different locations and collected together at the time of the actual delivery.	It is not uncommon to find that both commercial and open source ARA solutions will require that the artifacts to be delivered must be moved to the local drive of the ARA server. This is because the solution cannot retrieve files from a network drive. This adds complexity to the process, as scripts that move files to the ARA solution from an external network prior to the delivery must be managed.
Connect to any repository	Similar to the ability to support network drives, DeployHub connects to any repository for referencing artifacts including SVN, Git, Microsoft TFS or File System, and ensures files are “locked” prior to release.	Other solutions rely on binaries to be stored on a file system that is local to where the ARA solution is installed. This means that your release “package” cannot be locked and secured prior to your delivery, and must be pre-gathered so the ARA tool can perform the delivery.

Calendar - Ability to request deployments on a calendar	DeployHub includes a calendar where releases can be scheduled or blocked. This calendar, based on an *Environment,* provides a method for managing the release between developers and operations.	The majority of Open Source and many commercial solutions are not designed with a release calendar function. These solutions are often not designed at the Enterprise level, therefore a calendar function is not needed.
Create Own Actions - Ability to extend DeployHub to use existing scripts or create new ones quickly	DeployHub includes a built-in workflow engine which allows for the execution of external scripts, or the extension of out-of-the-box actions for performing releases.	Most competing ARA solutions will require the use of Jenkins to manage external processes.
No Client Install Required - Runs from a Web Site. No Fat Client	DeployHub is 100% web based. No need for any local client installation for the administrator or end users.	Many ARA solutions are command line driven. Those that have an upgrade from the command line will include a local install of the user interface.
Release Requests - Ability to request a deployment but have a 3rd party execute it	Because DeployHub was designed for the Enterprise, it provides the ability for a release to be “requested” with another group executing and managing the release. This allows for better organizational control over releases and improves security.	ARA solutions that were designed to support the “Linux Admin” or Production Administrator are not written to support the concept of a release request by other users.
Reuse of Deployment Steps Across Lifecycle - Define deployment once and reuse same deployment. Independent of lifecycle stage	Reuse is a core feature of DeployHub. The ability to reuse any steps in the workflow, regardless of the particular stage in the lifecycle, creates the repeatability required for the enterprise. There are differences between the Environments in managing a lifecycle release process. DeployHub removes those configuration changes from the release and manages them at the Environment level, allowing for dynamic processing of the meta data between all stages.	Lifecycle management is not a feature of many Open Source solutions. Developers often see the release process as a single stage “Development.” The process is written around the development environment, and then re-written to support the changes between development and test, or test and production. This leads to many versions of manifest files and scripts, as the information is managed statically.

Server Types and Component Types – Ability to automatically map Components to Servers	DeployHub includes Server Types and Component Types. This supports moving a release from 'Dev' with 1 server to a release in 'Prod" with thousands of Servers, automatically associating what components go on what servers based on their type.	Most deployment solutions require that a manifest, or list of servers be supplied that shows what components are to be installed on what servers. This is a static approach to what is a very dynamic requirement.
Life Cycle Models - Ability to create a lifecycle to fit SCM/ALM processes	DeployHub can support any type of development model, from Agile to Waterfall. Lifecycle models can be defined, with a promotion between states in the lifecycle. The definition of the lifecycle can be configured to meet the unique needs of the enterprise.	Project based tools are not often designed with a lifecycle model. The concept of promoting a release across the lifecycle is often a foreign concept to developers. Many version control tools do not manage source based on a lifecycle model, even though this concept may exist in the "real world" at the enterprise level. For this reason, the use of a lifecycle is not common to ARA tools.
Version Release Visibility - Ability to see what is deployed where	Once a deployment has been completed, there is traceability between the version of the release and the environment to which it was deployed. Anyone can view this information without interrogating log files.	Where something has been released, and what version was released can often times be difficult to determine without reviewing the details in a log file. This is a common problem with command line solutions and solutions that are not designed for the enterprise.

About DeployHub

DeployHub delivers highly reusable DevOps Solutions that allows our customers to master agile's last mile. DeployHub Pro, our continuous deployment solution, eliminates the friction in the continuous delivery pipeline as code moves between environments. Meister accelerates and streamlines the software compile process for highly efficient continuous builds.

As a 100% self-funded organization, we have the freedom to focus on our customer's needs, delivering innovation in software builds and release. An investment in our DevOps solutions or Professional Services forms a 'technical partnership' that provides you expertise and support to solve your toughest build and release problems for today and tomorrow.

DeployHub is an Open Source project at www.DeployHub.org.

DeployHub Pro has advanced security and auditing features. Learn more at https://www.DeployHub.com

About the Author

Tracy Ragan, CEO and Co-Founder

Ms. Ragan has had extensive experience in the development and implementation of DevOps for large organizations. She began her consulting career in 1989 on Wall Street specializing in build, testing and release management for the distributed platform. It was during her consulting experiences that Ms. Ragan recognized the lack of standardized development procedures on open systems that were common on the mainframe. Ms. Ragan served on the Eclipse Foundation Board of Directors as an Add-in Provider Representative for 5 years. She has been published on numerous occasions and regularly speaks at conferences including CA World where she presented for 15 consecutive years. She holds a BS Degree in Business Administration, Computer Technology from California State University, Pomona.