DZone Refcardz

Brought to you by: CloudBees

Jenkins on PaaS

Continuous Integration with Jenkins for Java Projects

By Marcelo Gornstein with Harpreet Singh

ABOUT THIS REFCARD

In this Refcard you will find information about Jenkins, the #1 open source CI server, including:

- Review of the most useful plugins
- Best practices
- The CLI and how to achieve distributed buildings of your projects
- Security
- Publishing Reports
- Integration to an Enterprise environment

Other Related Refcardz

Hot Tip

You may want to also take a look at the following related refcards: #87 (CI: Servers and Tools), #84 (CI: Patterns and Anti-Patterns), #145 (Continuous Delivery Patterns and AntiPatterns in the Software Lifecycle).

ABOUT JENKINS

Jenkins is an open source CI server written in Java. It is licensed under the MIT license terms, and it is widely adopted throughout the world for different languages and environments. It also has an enormous community that contributes plugins.

JENKINS OR HUDSON?

Initially, Jenkins was called Hudson, and was developed inside Sun Microsystems' offices. When Oracle purchased Sun (and inherited the code base), they tried to change the way the project was managed. In early 2011, tensions between Oracle and the community lead to a project fork, and Jenkins was born. The main difference is that Hudson is still managed by Oracle, and Jenkins by most of the original Hudson developers (including the founder Kohsuke Kawaguchi), focused on the community's needs.

Hot Tip

Most of the information you will find here will apply to Hudson and Jenkins as well.

JENKINS RESOURCES

- Homepage: http://jenkins-ci.org
- GitHub: https://github.com/jenkinsci/jenkins
- Twitter: @jenkinsci
- IRC Channel: chat.freenode.net at #jenkins and #jenkins-commit (for real-time commit alerts).
- Mailing lists: http://jenkins-ci.org/content/mailing-lists
- Jenkins User Conferences: Are held multiple times a year, see http://www.cloudbees.com/juc2012.cb

GETTING JENKINS

You can download Jenkins from the homepage: http://jenkins-ci.org/.

RUNNING IT

Java Webstart (from the web)

A great way to start playing around with Jenkins is to use its Java Web Start version, available at http://jenkins-ci.org/jenkins.jnlp. Just click on it and you'll be ready to go. Your preferences and changes will be saved to the directory .jenkins inside your home directory. When started, you can access Jenkins at http://localhost:8080/.

WAR FILE STANDALONE

Just run it from the command line, like:

```
java –jar jenkins.war
```

The war will be extracted and will serve Jenkins on port 8080 by default. Some useful command line arguments:

If you want to change the listening port/address, or tweak more startup options, use the --help option to show the full list of available command line arguments.

Hot Tip

By default, Jenkins will store its files in the directory configured in the environment variable JENKINS_HOME.

War File From a Web Container

Last but not least, you can opt to move the jenkins.war file in the webapps folder of your web container of your choice (like Tomcat or Jetty, etc.) and manage it from there.

For the Jetty instructions, see: https://wiki.jenkins-ci.org/display/JENKINS/Jetty.

For the Tomcat instructions see: https://wiki.jenkins-ci.org/display/JENKINS/Tomcat.

When deploying on an application server, the path to access Jenkins may change. By default you should be able to find it at http://localhost:8080/jenkins.

Run Jenkins in the Cloud with CloudBees DEV@cloud

For folks who don't want to install software, Jenkins is available in the cloud through CloudBees DEV@cloud. Sign up for CloudBees, enable Jenkins, and a Jenkins master will be set up for you.

THE CLI

Jenkins comes with a suitable command line interface. To access it, point your browser to http://ip:port/cli (replace ip:port with your own Jenkins URL) and follow the instructions. You can also find more information at the wiki: http://wiki.jenkins-ci.org/display/JENKINS/Jenkins+CLI.

Running the CLI

```
java -jar jenkins-cli.jar [-s JENKINS_URL] command [options...] [arguments...]
```

Using Credentials

If your Jenkins installation required authentication, it is possible that you won't be able to use the CLI properly. In this case, go to http://ip:port/me/configure and configure your SSH public key.

You can then login with the cli by using the login command. This will automatically load your SSH public key from your home directory and use it to authenticate with Jenkins.

CONFIGURING

With the GUI

Clicking on the Manage Jenkins option in the main menu will take you to the Manage System screen. There you can tweak the system settings, read system information, logs, and statistics, and manage the plugins. The Configure System option takes you to the full system options.

Without using the GUI

Jenkins stores its configuration (and the jobs information) in plain files inside the directory specified by JENKINS_HOME. If you happen to modify these files manually and want the settings to take effect, click the "Reload Configuration from Disk" icon.

CONFIGURING THE TOOLS (JDK/ANT/MAVEN)

By default, Jenkins comes with support for Java projects without any additional plugins. So let's configure our Java tools (JDK, Apache Ant and Maven). Click on the Manage Jenkins -> Configure System option. The tools have to be already installed in the same host where you are running Jenkins. For every tool, you can configure:

- **Name**: Since you can configure any number of JDK/Maven/Ant installations, you have to give each one of those a distinctive name, so you can later assign a specific tool to a specific build.
- **Home**: The installation path.

Automatically Installing JDK, Maven, Ant.

If you don't have JDK and Apache Ant/Maven, you can make Jenkins install it automatically. Whenever you see the option "Install automatically," you can check it to tell Jenkins to download and install that tool automatically as soon as a project needs it. You can also download it yourself, and tell Jenkins to use the file you want by clicking "Add Installer" and selecting the "Extract *.zip/*.tar.gz" option, and then giving the full path to the file.

UPDATING

To check for new versions, go to the Manage System screen. If there's a new version available, you will see a notice saying so and asking you to upgrade manually or automatically, or to read the changelog. If an upgrade breaks your installation, you can also downgrade your copy to the last working version from this menu.

PLUGIN MANAGEMENT

The vibrant Jenkins community has provided (and still continues providing) plugins to extend the Jenkins features. This has lead to Jenkins being an incredible and versatile tool to manage software projects. Inside the Configure System menu, you will find the Manage Plugins option.

Clicking on it will take you to Plugin Manager, where you will find any available updates for the plugins installed, a list of new (not installed) plugins, and a list of the already installed plugins.

The advanced tab lets you configure an HTTP Proxy, the Update Site (Jenkins connects to this URL to get updates and plugins). You can also upload a custom plugin, packaged as a .jpi file. Jenkins' plugins are distributed in jpi files (hpi files are also supported), which are common jar files, with a custom tree layout. You can read more about Jenkins plugins here: http://wiki.jenkins-ci.org/display/HUDSON/Plugin+structure

For versions prior to 1.442, once you have installed new plugins, you need to restart Jenkins for the changes to take effect. You can do this by checking the "Restart Jenkins when installation is complete and no jobs are running" checkbox.

CREATING CUSTOM PLUGINS

Creating your own plugin is easy and powerful. You might want to create your own plugins to integrate Jenkins with a specific tool, to add or modify build steps behavior, add reporting, change the GUI, add CLI commands, etc.

For this, Jenkins provides Extension points. You can find more about them here: https://wiki.jenkins-ci.org/display/JENKINS/Extension+points

A complete tutorial can be found at: https://wiki.jenkins-ci.org/display/JENKINS/Plugin+tutorial

USEFUL PLUGINS

Here's a list of the most used plugins. You might want to install some of these right away for your projects: http://wiki.cloudbees.com/bin/download/Jenkins+Enterprise/WebHome/CertifiedPlugins.pdf. This listing takes in a number of factors to score these plugins, including popularity and active contributions to the codebase.

CHANGING THE LANGUAGE

Hot Tip

By default, Jenkins will show up in the language of your browser. If you want a difference language, change your browser's language. If, on the other hand, you want to show Jenkins always in the given language, use the "Locale" plugin (available through the Plugin Manager). When installed, you will find an extra option at the Jenkins Management options, Locale -> Default Language where you can define the default language and even instruct Jenkins to override the browser's language.

SMTP CONFIGURATION FOR E-MAIL NOTIFICATIONS

Hot Tip Jenkins can send e-mail alerts when a build breaks or goes back to normal. In order to do so, it needs to know the SMTP information, which you can configure in the Manage System section. Then, in your job configuration, check the Email notification option.

Advanced Email Notifications

A useful plugin is "Email-ext", which allows a finer-grainedcontrol over how and when emails are sent. Install this plugin from the Plugin Manager.

This plugin adds a few more hooks that can be used to send e-mail notifications, such as when a build is about to be done, continues to fail, or becomes unstable. You can also customize the emails — you can choose to send them to different people based on a specific list, or you can send them because they made the comments. You can also include the build output.

CONFIGURING A SOURCE VERSION CONTROL SYSTEM

CVS and Subversion

Jenkins comes out of the box with support for CVS and SVN. For CVS, select the CVS option in the Job configuration.

Git

To use GIT with Jenkins, you will need to install the "GIT" plugin from the Plugin Manager.

TRIGGERING BUILDS

Manually

Builds can be manually triggered by clicking the "Play" (Build Now) icon at the left sidebar when inside a job. To configure when a build should be done for a job, use the Build Triggers configuration.

Scheduling a Build

You can configure your job to be automatically built when a dependency is built or when a specific job has been built. Also, the Build Periodically and Poll SCM options let you setup a crontab-like schedule, either to build the job or to poll the SCM for changes and trigger a build when changes are found. The schedule consists of 5 columns: Minute, Hour, Day of month, Month, Day of the week. The complete format is available by clicking the help icon for this option.

Another powerful option is Post-commit hooks for running builds. A build gets triggered on a commit into the repository. This is more efficient than periodic builds or the poll SCM option.

Establishing an Upstream-Downstream Relationship

Hot Tip You can build a project as soon as another project is built. This is useful when project B (downstream) depends on project A (upstream) and you want to make sure you build a new B version for each new build of A. Check-mark the Build after other projects are built option. You can specify more than one project by separating them with a comma.

PUBLISHING REPORTS

To publish the reports for your code metrics, see the Build Settings section of your job configuration. Some plugins will add a Post-build action, like the Status Analysis Collector, so you might want to check those actions to find new publishing options.

Thresholds On Metrics

In general, plugins that report on code metrics can be configured to set thresholds on those values in order to make a build stable, unstable, or failed. You can configure these thresholds by clicking the Advanced button.

PARAMETERIZED BUILDS

Hot Tip This is a very useful feature that will let you enter some parameter values into the job either manually or by taking them from other jobs. In your job configuration, select This build is parameterized (if you don't see this option, try looking for the Parameterized Build plugin in the Plugin Manager).

To add a parameter, just click the Add Parameter button, and select from the dropdown menu the type of parameter you would like to use. When the build is triggered, Jenkins will ask for the values required.

Accessing the Parameter Values

Once defined, you can access the parameters just as any other environment variable. For example, from the shell (or when configuring your job build):

```
echo ${VERSION} > version.txt
```

Accessing variables from Ant

```
<target name="mytarget">
    <property environment="env" />
    <echo message="${env.VERSION}"/>
</target>
```

Accessing variables from Maven

```
<project>
    <version>${VERSION}</version>
..
</project>
```

Triggering Other Builds with the Same Parameters

Sometimes it is useful to trigger the build of another job from a parameterized build, passing on the parameters used. In this case, use the Parameterized Trigger plugin.

Once installed, you will find a new option in your job configuration page: The Triggering parameterized builds on other projects option. On newer versions, you can access it from the Post Build Actions section of your job configuration.

Remember that the job should be parameterized as well. You can also mix other parameters or read them from a properties file.

BUILD PROMOTION

Hot Tip Promotions are a great tool to add steps in the build process and also mark builds according to a number of things. Promotions are available through the Promoted Builds plugin. When installed, you will find a new option called Promote builds when in your job configuration.

In order to promote the build, you can opt for a number of available criteria:

- Manual
- Build completed
- Built one or more downstream projects
- One or more upstream projects get promoted

With a promoting condition configured, you can add actions to be performed when the actual promotion occurs, by clicking the Add action dropdown box.

Promoting Builds

To promote a build, click the Promotion Status link (either at the left navigation bar, or at the build page). The met and unmet qualifications will appear from here, and you can do manual promotions. Promotions can also be manually forced. Look for the Force Promotion button on the right side of the page.

Hot Tip Don't Deploy Directly from Promotions It might be tempting to make deployments as actions for promotions, for example, by copying artifacts. This is not recommended, because you should never trust the workspace content after a build is done. Instead, make Jenkins archive the needed artifacts (and fingerprint them) and use a separate job to make the deploy that will run when the promotion occurs. In this way, you are relying on Jenkins to choose the correct files to include in the build.

USING THE MAVEN RELEASE PLUGIN

You can have Jenkins execute the Maven Release Plugin and deploy to a maven repository by installing the M2 Release plugin. When installed, you will notice a new option, too, in the left navigation bar. To make a release, click the Perform Maven Release link, and the release configuration will be shown.

Configuring credentials

At the Jenkins configuration page, you will find a new optionallowing you to configure your repository information and your credentials that will be used by the M2 Release plugin to make the releases.

MANAGING BUILD VERSION NUMBER

By default, Jenkins increments the build version by 1. If you want to have a more custom version number (or string) for your job, use the Version Number plugin. When installed, a new version will show up in your job configuration at the Build Environment section named Create a formatted version number.

This allows you to create an arbitrary version number string and set an environment variable with it, suitable to be used as any other environment variable. You can specify a number of variables to be used in the final string (see the examples by clicking the help icon).

Resetting the build number

If you wish to set the next build number to an arbitrary number (or maybe even reset it to 1), locate your job directory, and modify the file nextBuildNumber. For example, for a job named Jenkins, the path would be:

```
$JENKINS_HOME/jobs/Jenkins/nextBuildNumber
```

Afterwards, restart Jenkins or reload the configuration from disk from the Management menu.

Hot Tip You could also use the plugin "Next Build Number" to change the next version number that Jenkins will use for a given job.

CLAIMING A BROKEN BUILD

From time to time, someone breaks the build. It's inevitable. When this happens, someone needs to look into the build log and fix the error. Jenkins has a nice plugin for this exact situation. The Claim plugin, available from the Plugin Manager, hen installed, will add a new Post Build Action that you can use in your job configuration.

Claiming a broken build serves two purposes: one is to explain what happened, and the other one is to take responsibility to fix it, and then to let others know about it. When you claim a build, the other developers will notice it and they will expect you to fix the build, so they will carry on with other tasks.

When the plugin is enabled, and when your build breaks, it will show a new option in the build menu.

Clicking on the Claim link will display the text area where you can write your excuses about how the build was broken.

FILE FINGERPRINTING

File Fingerprinting will let you create an MD5 sum for all archived artifacts. Jenkins will automatically save into its db the fingerprint of all the generated artifacts.

Hot Tip At the left navigation bar, you will notice the option Check File Fingerprint, where you can give Jenkins a file and ask it to return the corresponding job and version that generated it.

This is also useful for Jenkins so it can link downstream build jobs to upstream ones. To have Jenkins save the fingerprint of the generated artifacts, look for the Record fingerprints of files to track usage options inside the Post build Actions.

SECURING JENKINS

Hot Tip Jenkins comes out of the box without any kind of security. This is OK for intranet installations, but on public projects you would want to control who can edit your jobs configuration and/or trigger builds, etc. In the Jenkins configuration page, look for the Enable Security checkbox. Activating it will show more options.

Authentication

- Delegate to servlet container: If you are running Jenkins on Glassfish, Tomcat, or another servlet container, you have the option to let it manage your users authentication for you.
- Jenkin's own user database: The most simple way to authenticate users is to let Jenkins handle all the work. Users can register themselves (if you check the Allow users to sign up checkbox) and then you can decide who can do what. Sometimes you will notice users that can't log in to Jenkins but still show up in the People menu. These are disabled accounts. You can set up a password for them by clicking on the configure option once in the user configuration screen. NOTE: It is recommended to use the LDAP or Active Directory plugins to connect into the enterprise user database over this option in an enterprise environment.
- LDAP. Lots of companies use LDAP as an authentication backend, mostly because Active Directory can also use it. If you choose this option, some more data will be displayed, where you will need to enter the LDAP server information and DN information.
- Unix user/group database: The last option will let you authenticate your users against a PAM service in your unix box. This has a downside: you will need to create the accounts on your unix machine just to let the users login to Jenkins.

Hot Tip Sometimes you will notice users that can't login to Jenkins but still show up in the People menu, these are disabled accounts. You can set up a password for them by clicking on the configure option once in the user configuration screen.

Authorization

Once you authenticate your users, you need to authorize them to do whatever they need to do. Right below the authentication options, you will find the authorization settings.

- Anyone can do anything: This is enough for most installations where you trust your users.
- Matrix Based: If you want to fine tune what users can do, this is the way to do it. First, set up anonymous and administrator permissions:

The Anonymous user represents the users not logged in. You should create the admin user with User/group to add. It represents your administrator user, and should have access to do everything.

Project-Based Matrix configuration
It might be desirable to give permissions based on the project. In this case, select the Project-based Matrix Authorization Strategy. You can then define the generic permissions that will be overriden with the permissions you define in the job configuration page, where you will find a new option: Enable project-based security. This will show a new matrix that will override whatever you have configured in the Jenkins configuration page.

Lost your password?

Hot Tip

If you happen to lose your password and need to reset it, there is a solution. Open the file config.xml inside the Jenkins home directory (specified by JENKINS_HOME), and edit it, changing "useSecurity"from true, to false. Then restart Jenkins. You should be able to configure it without logging in. Change your password (and/or security settings) as quick as possible and restart.

CloudBees Roles-Based Access Control Plugin
This is the most sophisticated (and simple) plugin for authorization. It allows users to set up roles, assign groups to roles and assign users to the groups (Roles -> (Groups*) -> (Users*).

RBAC can use external groups (such as through LDAP) or it can let admins create custom groups for Jenkins (admins do not need to go to an IT administrator to set up groups).

A matrix of checkboxes can be used to grant permissions for specific roles to jobs or folders. Permissions can be additive or subtractive on the folder or jobs within those folders.

DISTRIBUTED BUILDS

If you have lots of jobs in your Jenkins installation, you might want to distribute the work it takes to build them.

Hot Tip

Jenkins nodes can be arranged in a master/slave fashion, where the master can delegate some builds to the slave nodes, allowing you to scale in the number of jobs configured in a given installation.

Use the cloud
Delivery of software products is spiky by nature. Commits and build activities peak during the release phase of the product. Thus, more slaves can be kicked off during peak usage. Teams can also use the Amazon EC2 plugin to offload their builds into the Amazon IaaS. With this setup, teams are responsible for management of masters and slaves in the cloud.

Hot Tip

A more advanced option is to use CloudBees DEV@cloud service and let CloudBees manage Jenkins masters and slaves, so teams can just focus on development.

Creating a Node
Click on the Manage Nodes option of the configuration page. You will notice the New Node option on the left sidebar. Click on it to create a new node, and enter the information for the new node, mainly the name. Next, configure the node settings:

- # of Executors: How many jobs can be built on this node at the same time.
- Remove FS root: Where the Jenkins files will be stored in the remote node, analogous to JENKINS_HOME.
- Labels: Custom labels that you want to assign to this node and others, to create groups of nodes suitable to build some specific jobs.
- Usage: You can set this node to build any jobs or only the ones specifically tied to it.

Launching the Slave agent via SSH
Jenkins has an embedded SSH client, which is very useful (in this particular case) to make it connect to the target machine, by providing the connection information and credentials, and run the slave agent automatically.

Select the Launch slave agents on Unix machines via SSH option, and fill in the required information.

The node should then start right away as soon as you save the node configuration.

Launching the Slave agent via Java Web Start
Another option is to run the slave agent manually, via Java Web Start. Select the Launch slave agents via the Java Web Start option. Then log in to the target machine, and use this command line:

javaws http://ip:port/computer/nodename/slave-agent.jnlp

Replace ip:port with the location of the master Jenkins node, and replace nodename with the name of the node you've created (in this case, for example, it would be "My new Node" with spaces and without the quotes).

Launching the Slave agent headlessly
You can also run the slave agent without a GUI, suitable to make it automatically run at system startup, by downloading: http://ip:port/jnlpJars/slave.jar

and then running:

```
java -jar slave.jar -jnlpUrl
```

http://ip:port/computer/nodename/slave-agent.jnlp

Configuring environment variables
You can set any environment variables needed in the slave node by setting the node properties, these will help you customize the build and maybe dynamically generate files or reports.

Configuring the slave node Java tools
If the path for Maven, Ant, and the JDK is different from the ones in the master node, you can set these in the Tool Locations section.
Once the node is configured and connected, you should see the new node Queue Status right below the main build queue.

Restricting where jobs are built
You can set (restrict) where a job can be built right in your job configuration by choosing the "Restrict where this project can be run". A label expression that allows boolean operators can be set, describing which nodes can handle this job. To see the complete list of available boolean operators, click the help icon at the right.

ENVIRONMENT VARIABLES

Hot Tip

Jenkins has a number of environment variables very useful to customize the generated build and output build information. You can use these directly from your own job and reporting tools.

Note that plugins can add their own variables, so you might want to check the documentation for each one of them if you are interested. An official list can be found at: https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project

Accessing variables from Ant

```
<target name="mytarget">
    <property environment="env" />
    <echo message="${env.JOB_URL}"/>
```

Accessing variables from Maven

```
<project>
    <url>${JOB_URL}</url>
```

BEST PRACTICES WITH JENKINS: USING ENTERPRISE PLUGINS

You can find an official list of best practices at: https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Best+Practices Secure Jenkins

Securing Jenkins

Securing Jenkins starts with the assessment of who should access the Jenkins master. The second step is to choose the right underlying authentication mechanism (discussed earlier).

Last, is to secure the right roles, groups and users. This usually involves settling on a flexible administration topology where secure projects are secure, can be spun off easily and can administer themselves without too much intervention from a uber-administrator.

Backup Jenkins

Backing up and restoring is as easy as to copy to/from the directory specified in JENKINS_HOME

With the CloudBees Backup plugin, creating a backup is as simple as creating a new job in Jenkins. Backups can be local or to a sftp server.

Organize Jobs

Use the CloudBees folder to organize jobs into folders. The plugin allows you to create nested hierarchies of namespace aware jobs. Tie this with CloudBees RBAC plugin and you can set sophisticated permissions on folders. .

Capture best practices with the CloudBees Templates plugin

A significant part of a Jenkins administrator's time is spent onboarding projects. Most jobs differ by small number of parameters that ideally should be templatized. These templates capture sameness of jobs across projects and can be used to easily on-board new projects, jobs. Furthermore, a change in a template is automatically and instantly propagated into all jobs.

Shield repository from failures: Make builds unbreakable

The validated merges (works with Git) or the "unbreakable builds" feature allows only good commits to make through to the repository. Commits are made to an intermediate repository maintained by Jenkins, who takes the onus of merging code with upstream branch, run tests, and do the final push to the main repository. The developer just fires and forgets.

Eliminate Jenkins downtime

The High Availability feature, offered within Jenkins Enterprise by CloudBees, eliminates downtime due to master failures. Multiple Jenkins masters act as backups waiting for a primary master failure. Once a failure is detected, a backup master automatically boots up and acts as a failover.

Ensure plugin and binary compliance

Teams often end up working with different versions of plugins or binaries. This version mismatch results in subtle failures in the application that are discovered late in the development cycle. The Custom Update Centers plugin offered by CloudBees, allows administrators to create their own update centers, so they can upload plugins, specify the version of the plugin that is available to downstream Jenkins instances, and more.

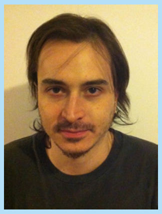
REFERENCES

Jenkins, the definitive guide book: http://www.dzone.com/links/free_ebook_jenkins_the_definitive_guide_continuou.html

Jenkins with PHP Projects with Phing: http://marcelog.github.com/articles/ci_jenkins_hudson_continuous_integration_php_phing.html

CloudBees Jenkins Enterprise User Guide: http://wiki.cloudbees.com/bin/view/Jenkins+Enterprise/Jenkins+User+Guide+and+Release+Notes

ABOUT THE AUTHORS



Marcelo Gornstein is a software developer by heart and a self learner, with more than 15 years of experience (in total) in different languages, ranging from Assembly to Java, C, Php, Ecmascript, Erlang, and Ruby. He's the author of different open source software projects (mostly related to VoIP) available at github and very passionate about best practices, and new technologies. He has been a system administrator, a software developer, lead developer, architect, and now a product owner. In his free time he likes to read a lot, play with his cats "qwerty" and "dvorak", learn new stuff, and develop more open source software (and from time to time, write an article). You can find more about him (his projects and articles) at his personal homepage: http://marcelog.github.com/

Harpreet Singh has 12 years of experience in the software industry in various roles. He came to CloudBees from Oracle where he was a Senior Product Manager in the Application Grid group - he helped onboard GlassFish into Oracle. He was at Sun Microsystems for 10 years in various roles such as Group Product Marketing Manager leading marketing efforts for Java EE 6, GlassFish 3.1 and monetization program for GlassFish Portfolio. He was also the Product Manager for Hudson and launched it as a supported product within Sun's GlassFish Portfolio. In his prior life, he was an engineer in the Java EE RI, GlassFish teams and was the technical lead for GlassFish 2.1. He has an MS degree in Computer Science from University of Cincinnati and an MBA from Santa Clara University. He lives in the San Francisco Bay area with his wife and their puppy.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome

refcardz@dzone.com

Sponsorship Opportunities

sales@dzone.com

ISBN-13: 978-1-936502-56-1
ISBN-10: 1-936502-56-9

50795

9 781936 502561

$7.95

Version 1.0