Zend Blueprint for Continuous Delivery

The PHP Company

Implementation Guide:

Jenkins  and zend® Server

by Slavey Karadzhov

Implementation Guide: Jenkins and Zend Server

This pattern contains instructions on how to implement a Jenkins and Zend Server integration. For more information on the principles and methodology behind Zend's Pattern for Continuous Integration with Jenkins: http://www.zend.com/en/jenkins-pattern-paper

Installing the Continuous Integration Environment

The installation instructions mentioned in this paper are suitable for targeting Debian based systems, however similar steps should be applicable to other operating systems based on the appropriate commands.

Installing the Zend Server PHP Platform

Zend Server is a complete, enterprise-ready Web Application Server for deploying, running and managing PHP applications with a high level of reliability, performance and security both on-premise and in the cloud. Zend Server enables the implementation of a continuous delivery environment for PHP software development projects.

In order to install Zend Server you need to execute the following commands:

```
wget http://repos.zend.com/zend.key -O- |apt-key add -
sudo sh –c \
        'echo "deb http://repos.zend.com/zend-server/6.3/deb_ssl1.0 server non-free" > \
       /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install zend-server-php-5.3 # if you want to install PHP 5.3 or
sudo apt-get install zend-server-php-5.4 # if you want to install PHP 5.4
```

For detailed instructions on installing Zend Server 6.x, see: http://files.zend.com/help/Zend-Server-6/zend-server.htm#installation_guide.htm . For Zend Server versions earlier than 6.2, it is required to set the time zone by editing the 'date.timezone' PHP directive. You can set the timezone from the Zend Server UI, on the Configurations | PHP tab.

Installing the Continuous Integration server with Jenkins

For the continuous integration and delivery machine it is required to install the following applications:

PHP Stack Components

PHP

```
apt-get install php5-cli
```

Pear

```
apt-get install php-pear
```

PHPUnit

PHPUnit [3] is a PHP Unit Testing framework. We will use it to run our phpunit tests to ensure the high quality of our software and detect errors and regressions early.

```
pear install pear.phpunit.de/PHPUnit
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Phing

Phing [4] is a PHP project build system or build tool, similar to Apache Ant. Phing is written in PHP and uses simple XML build files and extensible PHP "task" classes provide a robust and flexible PHP application build framework.

```
pear channel-discover pear.phing.info
pear install phing/phing
```

Jenkins

Jenkins [5] is an application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. We will use Jenkins to help us build our application, test it and deploy it continuously throughout the software delivery process.

```
wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | \
        sudo apt-key add -
sudo sh -c 'e'b http://pkg.jenkins-ci.org/debian binary/ > \
       /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install jenkins
```

Zend Server SDK (Zs-Client)

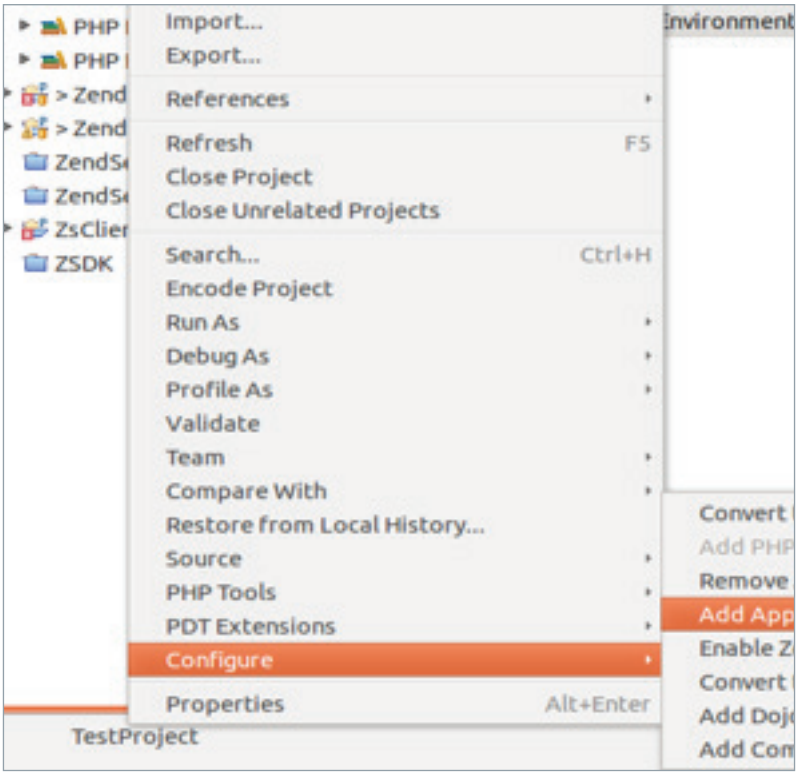
Zend Server SDK or Zs-Client, is a console command line client that allows remote control of Zend Server and Zend Server Clustered environments. It execute commands via a REST API eliminating the need to go login to the Zend Server administration web interface or to have SSH access to the machine on which Zend Server is installed. To install the Zend Server SDK:

```
cd /usr/local/bin
sudo wget https://github.com/zend-patterns/ZendServerSDK/raw/master/bin/zs-client.phar
sudo chmod a+x zs-client.phar
```

Configuring the Continuous Integration Environment

Adding application packaging support

In order to add packaging automation support to an existing PHP project you will be required to add two files, named deployment.xml and deployment.properties. These files describe how the source code should be packaged, dependencies, and what should be executed before or after the installation of the source code on the server. These files are added once to the main directory of the project. If you are using Zend Studio 9.x or above as your integrated development environment, you can enable deployment support directly from the Zend Studio interface. This is done by right clicking on the PHP project that you are working on, then choosing 'Configure' and finally clicking on 'Add Application Deployment Support', as shown below:



If you are not using Zend Studio as your PHP IDE, you can download the Zend Server SDK (zs-client) and add deployment support to the PHP project.

```
wget https://github.com/zendtech/ZendServerSDK/raw/master/bin/zs-client.phar
php zs-client.phar initZpk –folder=/<path-to-PHP-source-code>
```

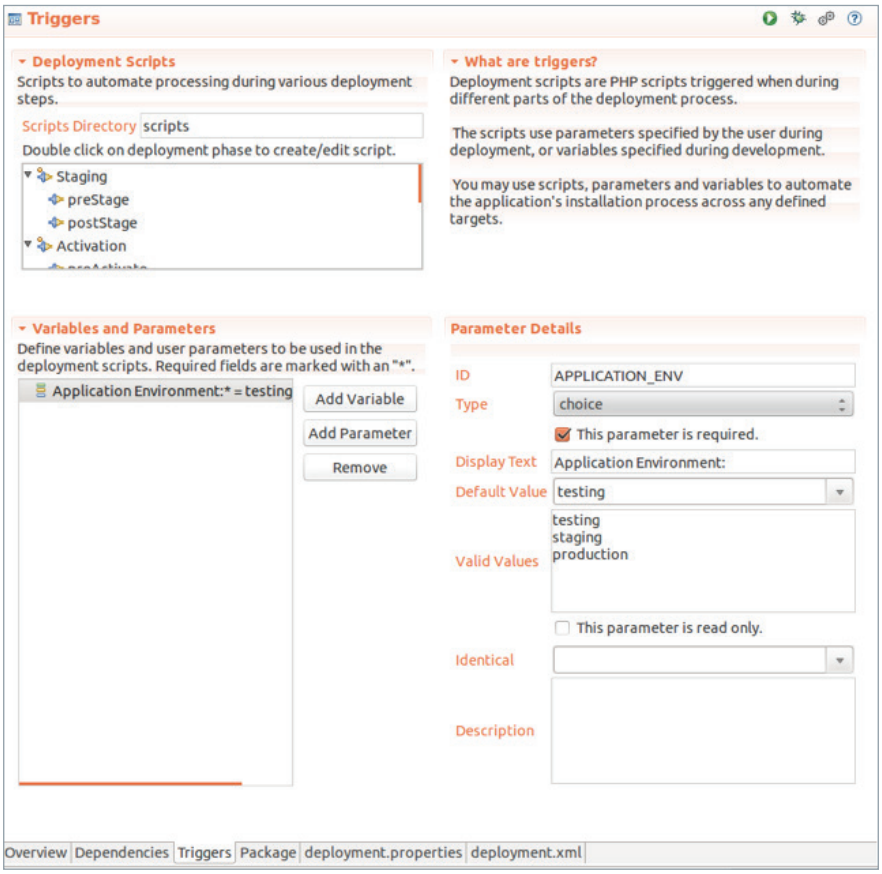
The deployment.properties file contains information about the files that will be part of the package. The appdir. includes lists the files that are part of the application and scriptsdir.includes contains the files that will contain the special scripts to be executed before and after the installation.

The second file that is created is called deployment.xml. This file contains information about the version of the application, parameters that should be passed when installing the package, required libraries, versions of PHP or PHP directives, and additional information required by the Zend Server Deployment automation system.

You can find more information about the package structure and the format of the files in the Zend Server Documentation at http://files.zend.com/help/Zend-Server-6/zend-server.htm#understanding_the_package_structure.htm

And at http://files.zend.com/help/Zend-Server-6/content/the_xml_descriptor_file.htm

In order to be able to deploy the same package on multiple application environments such as testing, staging or production, we need to provide the Zend Server Deployment system information during the deployment specifying the targeted application environment. In order to achieve this functionality we need to add a new user defined custom parameter of type 'choice' to the deployment.xml as shown below in the Zend Studio IDE:



If you are not using Zend Studio as your IDE, you can edit the deployment.xml file with any text editor and add the needed lines. Below are the changes in the deployment.xml file related to the addition of this user defined parameter (highlighted lines below):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<package xmlns="http://www.zend.com/server/deployment-descriptor/1.0" version="1.0">
 <name>AddYourUniqueApplicationName</name>
 <version>
  <release>1.0.0</release>
 </version>
 <appdir>data</appdir>
 <scriptsdir>scripts</scriptsdir>
 <dependencies>
 </dependencies>
 <parameters>
     <parameter display="Application Environment:" id="APPLICATION_ENV" readonly="false"
required="true" type="choice">
    <validation>
     <enums>
      <enum>testing</enum>
      <enum>staging</enum>
      <enum>production</enum>
     </enums>
    </validation>
    <defaultvalue>testing</defaultvalue>
  </parameter>
 </parameters>
</package>
```

When a custom parameter is defined, it is accessible to the deployment scripts executed in the different stages of the deployment cycle by the Zend Server Deployment automation system. The trigger (hook) scripts[6] are included in the deployment package. For example prior to "Activating" the package in order to set special connection settings, you can perform actions similar to the example given below in the 'pre_activate.php' script:

```
<?php
// code from pre_activete.php
$applicationEnv = getenv('ZS_APPLICATION_ENV');
$applicationBaseDir = getenv('ZS_APPLICATION_BASE_DIR');
$config = new Zend_Config($applicationBaseDir.'/config.ini', $applicationEnv);
```

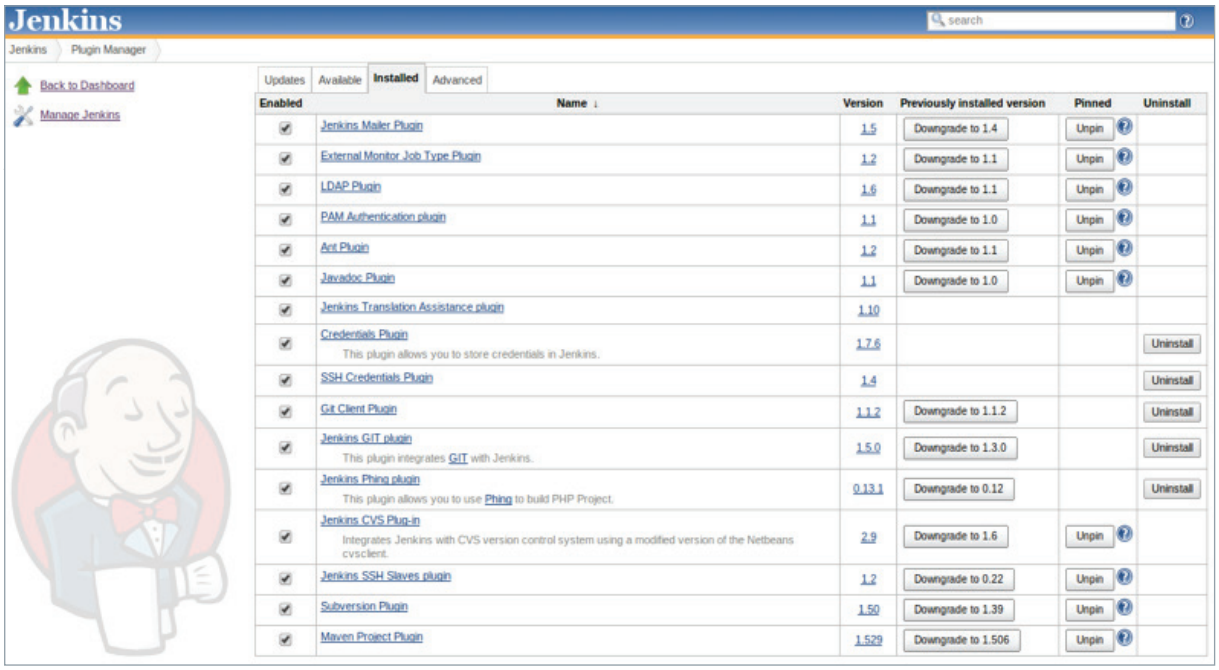
Adding Build Support

The build process is comprised of a set of distinct tasks that are in general executed sequentially and depend on the success of previously executed tasks. The process is orchestrated by the build tool Phing, and handles the creation of the package from the source code and resources to a deployable self contained package that can then be deployed on the different targets. To ensure proper quality, it is essential to run unit tests with sufficient coverage of the code execution scenarios. Following the successful unit test suite completion, Phing will package the application into distributable zpk file which is consumed by the Zend Server Deployment system and finally deploy this file on the desired server and application environment to ensure successful deployment of the application. It is also recommended to certify the application package (zpk) by running validation tests to ensure that the package is successfully deployed on a test server and the application is functional following its deployment.

In order to achieve this sequence of build tasks we will add the build.xml descriptor file in the root folder of our project. The build.xml file describes the steps to complete the build process. The example build.xml below can be directly copied and used with Phing:

```
<?xml version="1.0"?>

<project name="ApplicationBuildName" basedir="." default="unittest">
 <target name="unittest">
  <phpunit bootstrap="tests/bootstrap.php">
    <formatter type="summary" usefile="false" />
    <batchtest>
      <fileset dir="tests">
        <include name="**/*Test.php"/>
      </fileset>
    </batchtest>
  </phpunit>
 </target>

 <target name="zpk" depends="unittest">
    <exec command="php /usr/local/bin/zs-client.phar packZpk --folder='${project.basedir}' --destination='${project.basedir}' --name='application.zpk'" checkreturn="true"/>
 </target>

 <target name="deploy" depends="zpk">
  <exec command="php /usr/local/bin/zs-client.phar installApp --zpk='${project.basedir}/application.zpk' --zsurl='${host}' --zskey='${key}' --zssecret='${secret}' --baseUri='${base}' --userAppName='${app}' --userParams='${params}'" checkreturn="true"/>
 </target>

</project>
```

This build.xml descriptor file is used by the build system (Phing in this case) to execute the described build steps on the Continuous Integration and Deployment server (CI/CD server). Additional user parameters can be passed to the build file for specific build scenarios. In the sample above, the ${host} parameter will be replaced with the host parameter given during the build process.



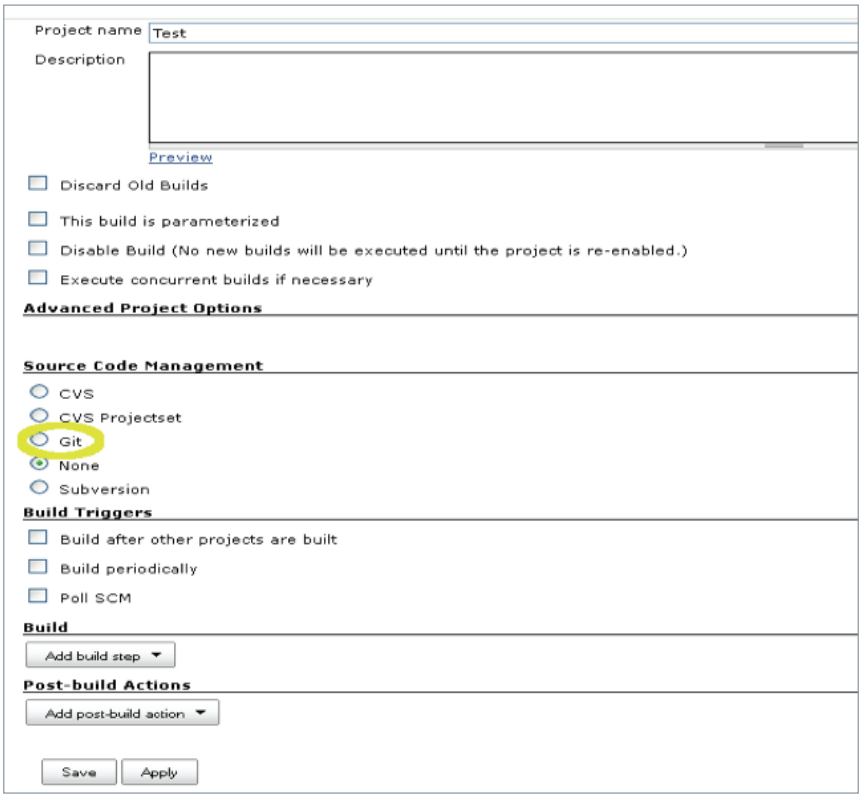
Implementing the continuous integration process

Create New Jenkins Job

The first step in constructing the continuous integration process is creating a new job. Create a new Job from the Jenkins web interface. Go to 'New Job' and provide a unique name for the new job in the form, for example "StagingBuild", and choose 'Build a free-style software project'.

Choosing the configuration management repository

Now edit the new job configuration. You should be able to specify what versioning system you want to use. Specify git and as show in the image below:



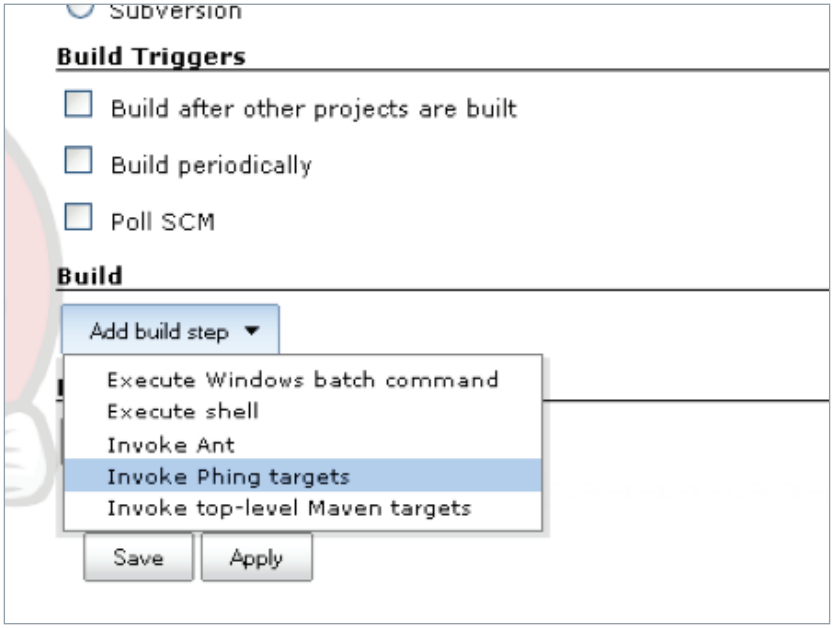
Next, specify the 'Repository URL', 'Branch Specifier", Config user.name Value' and 'Config user.email Value'.

For example:
Repository URL : "https://zendtechnologies@bitbucket.org/zendtechnologies/deploymentapp.git"
Branch Specifier: "master"
Config user.name Value: "Slavey Karadzhov"
Config user.email: "slaveyy@zend.com"

For every new build, Jenkins will try to define a tag in the repository which will specify exactly which files were used to build the package. If you prefer not to define this tag, please check the 'Skip internal tag' option.

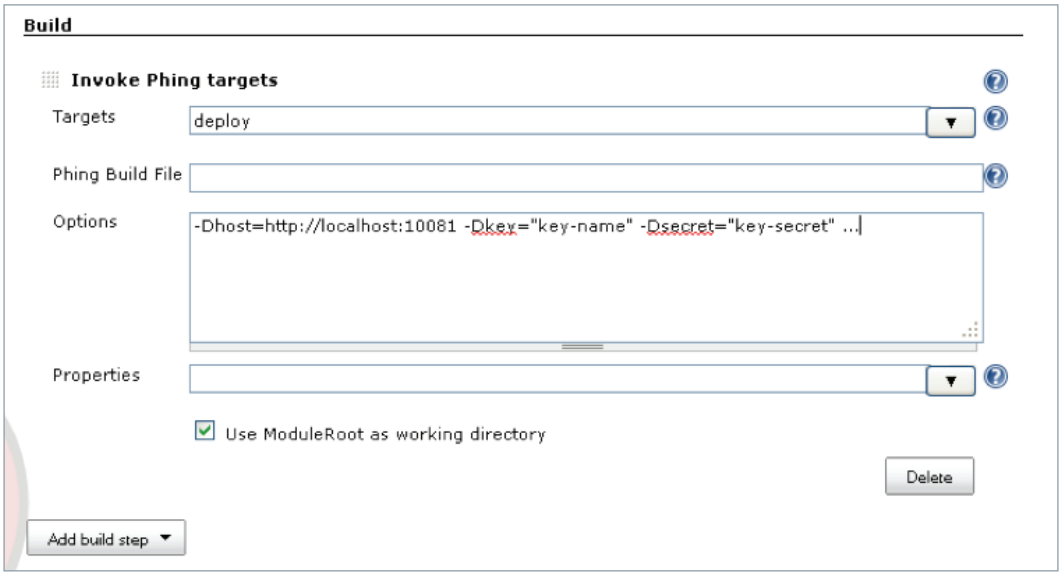
Passing parameters to the phing build

The next step is choosing the build system for building the application. Select 'invoke Phing Targets' as shown below.



This option should be available in the drop down menu however if it does not exist in your Jenkins web interface, please make sure that you have installed and enabled the Phing Jenkins plugins.

Next, you will need to specify the target that you intend to call. In our example build.xml we have 'unittest', 'zpk' and 'deploy' defined as targets. Using our build.xml defined above as an example, call the 'deploy' target. Calling 'deploy' will also call 'zpk' and 'unittest' to be executed, before executing the actual deploy task. Now click on the 'Advanced' button and you will be able to see text area where you can pass Options. In the text area for each option that you type it is required to add '–D<name>=<value>'. For example "–Dhost=http://localhost:10081/" will pass a 'host' parameter to the build.xml and the value will be "http://localhost:10081/". Additional options that are required should be added separated with a space. See the example below.



In our sample build.xml we expect the following mandatory parameters to be passed during the build process:

host – pointing to an URL where the Zend Server web interface can be accessed

key – a valid Zend Server WebAPI name

secret – a valid Zend Server WebAPI secret for the provided Zend Server WebAPI name

base – base URL to which the application should be installed

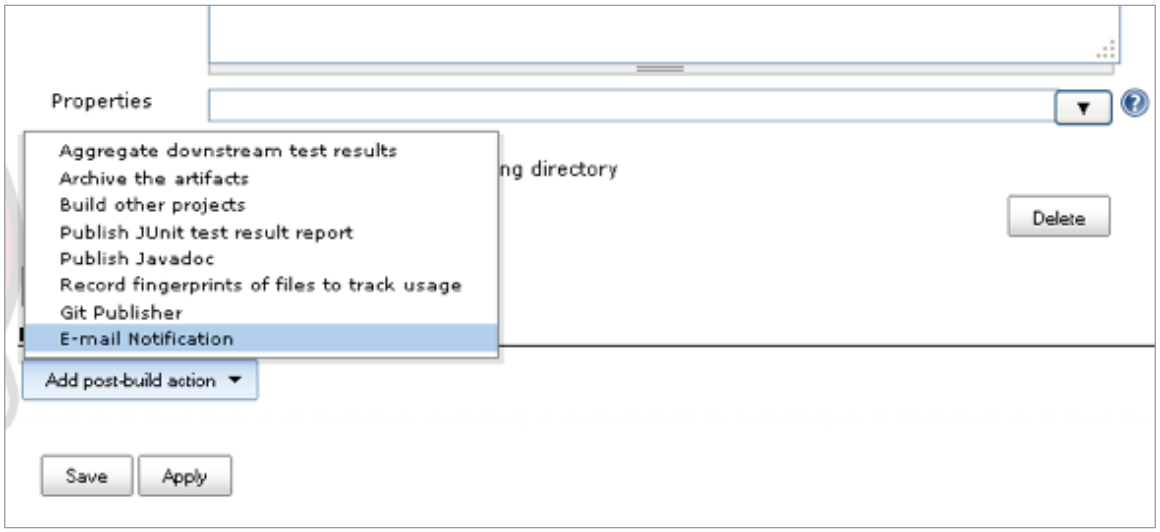
In addition to the mandatory parameters there are also two optional parameters:

app – display name of the application if different then the one specified in the deployment.xml

params – the value should be a string formatted as a valid HTTP query string that contains the custom parameters that we need to pass during our build. For our case we can add in the options text area the following string:
-Dparams="APPLICATION_ENV=staging"

Email Notification on build results

To send a notification in the event of a successful build, we can add a post-build step. Choose from the available post-build actions "E-mail Notification" as shown below:



Fill the space separated email addresses of the individuals that need to get the build notifications in the 'Recipients' field.

Launching the continuous integration process

After Jenkins is configured correctly to manage the continuous integration process, clicking on the 'Build now' button for the Jenkins project enables a completely automated delivery process that will also deploy the package on a server. The same deployment package which has been tested and validated is being used from this point onward for any deployment of this software version. If the target server is part of a Zend Server cluster, then all the other Zend Server nodes in the cluster will get the application deployed automatically. Deployment to other target testing and staging environment can be automated in a similar manner. Deployment to production can be automated as well by defining the production deployment as another target enabling a continuous deployment process. However, unless there is a business value in releasing every build to production or in the common case in which more comprehensive user acceptance testing is warranted, the final step of pushing the release to the production target requires manual authorization.

Pipeline build

Continuous Integration environments can become very complex, with many tasks to run in order to complete the deployable software package including long running tasks such as automated functional tests. The basic approach outlined above is comprised of one running task, which in turn executes a collection of more granular build tasks. To better address increasing complexity in the continuous integration process, it is possible to break down that main task into smaller, manageable subtasks, and build a pipeline in which every job will trigger the next upon completion.

One advantage of the pipeline approach is that it becomes much easier to visualize the flow of the build process, and see exactly when it succeeds and where it fails and react accordingly. A pipeline also provides more control over the process by allowing the workflow to stop in critical steps waiting for a manual intervention. Such steps usually include user acceptance validation or approvals moving to the next delivery step such as deploying the build to staging or production. Creating a task pipeline enables parallelization of tasks in cases where there is no direct dependency between steps. Some tasks can run independently of one another accelerating the overall delivery process. Tasks such as generating reports for instance can typically execute independently of testing or packaging tasks.

In the following example, the tasks package (to package the application), phpcs (to generate a code sniffer - code style - report) and report_converage (to generate the unit test coverage report) can all be run in parallel, following the success of unit tests. The pipeline will thus progress, and run faster, until it reaches the deploy_to_prod task where it will pause and wait.

The delivery pipeline presented above is built using the build-pipeline plugin for Jenkins (https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin). It provides a visualization of a series of jobs in which the last executed task triggers the next until the delivery pipeline is complete.

References and resources:

[1] The Zend Blueprint for Continuous Delivery, Zend Technologies, www.zend.com/continuousdelivery

[2] http://www.zend.com/en/products/server/

[3] http://phpunit.de/manual/current/en/index.html

[4] http://www.phing.info/

[5] http://jenkins-ci.org/

[6] http://files.zend.com/help/Zend-Server-6/content/understanding_the_package_structure.htm#Hook_Script_Constants

About the Author

Slavey Karadzhov is a Senior Consultant at Zend Technologies. He is a PHP 5/5.3, Zend Framework and MySQL Certified Engineer. Proud owner of two master degrees, in Computer Science from Sofia University, Bulgaria and in Software Technologies from Stuttgart University of Applied Science, Germany. Slavey is a strong open source supporter and recognized software innovator as well as the author of a Zend Framework 2 book called "Learn ZF2: Learning By Example" (http://learnzf2.com).

Zend Professional Services lead implementations of agile methodologies, application design, optimization and continuous delivery solutions at global enterprises running mission-critical PHP applications. The consultants focus on mentoring customers' development and operations teams to deliver faster releases of high-quality PHP applications with higher performance and availability.

About Zend

Zend partners with businesses to rapidly deliver modern apps across web, mobile and cloud. We helped establish the PHP language, which today powers over 200 million applications and web sites. Our flagship offering, Zend Server, is the leading Application Platform for developing, deploying and managing business-critical applications in PHP. More than 40,000 companies rely on Zend solutions, including NYSE Euronext, BNP Paribas, Bell Helicopter, France Telecom and other leading brands worldwide.

Corporate Headquarters: Zend Technologies, Inc. 19200 Stevens Creek Blvd. Cupertino, CA 95014, USA · **Tel** 1-408-253-8800 · **Fax** 1-408-253-8801
Central Europe: (Germany, Austria, Switzerland) Zend Technologies GmbH, St.-Martin-Str. 53, 81669 Munich, Germany · **Tel** +49-89-516199-0 · **Fax** +49-89-516199-20
International: Zend Technologies Ltd. 12 Abba Hillel Street, Ramat Gan, Israel 52506 · **Tel** 972-3-753-9500 · **Fax** 972-3-613-9671
France: Zend Technologies SARL, 105 rue Anatole France, 92300 Levallois-Perret, France · **Tel** +33-1-4855-0200 · **Fax** +33-1-4812-3132
Italy: Zend Technologies, Largo Richini 6, 20122 Milano, Italy · **Tel** +39-02-5821-5832 · **Fax** +39-02-5821-5400
Ireland: Zend Technologies, The Digital Court, Rainsford Street, Dublin 8, Ireland · **Tel** +353-1-6908019

0106-M-WP-0314-R1-EN