The PHP Company

Implementing PHP Application Deployment Best Practices

A Zend Technical White Paper

by **Eric Ritchie**, Zend Technologies

August 2013

Introduction

Once an afterthought, if thought of at all, an efficient and reliable application deployment process is becoming an essential component of modern application development projects. This is true for a number of reasons. The reputation and revenue of companies with public facing applications can be rapidly tarnished by deployment catastrophes. Any downtime for an ecommerce application, to give a non exclusive example, impacts revenue immediately as some customers will simply visit a competitor. Such a failure severely impacts future revenue as these customers who are presented with multiple competing options may not return on their next visit. A Gartner Research report suggests that more than 50% of mission critical application outages are caused by changes, configuration or release related issues. Many of these deployment failures can be attributed to the manual nature and often informal characteristics of application deployment tasks, and as a result the contribution of human error to overall application failures has been increasing while the proliferation of virtualization and cloud technologies has reduced the occurrences of hardware and load related failures. Therefore, enabling efficient and automated deployment and allowing organizations to deliver changes across diverse virtualized and physical environments is an essential component in establishing high availability and minimizing the impact of deployment failure on the bottom line.

The impact of manual and error prone application deployment practices goes beyond the cost of downtime. The price of software development projects has risen exponentially along with the complexity of their implementation. A large part of this cost can be attributed to the overhead of deploying large applications to the increasingly complex environments in which they will be hosted. The amount of time and effort dedicated by IT operations staff and developers to create and maintain scripts and communicate manual instructions is often overlooked. It then becomes a significant factor in the overall cost of delivering the application while inhibiting the ability of the organization to quickly respond to new business requirements. Therefore, efficient deployment processes help prevent projects from becoming money pits.

While there is much to love about PHP, it has not been without its challenges when it comes to application deployment. Ironically, these challenges are directly related to the very specification of the language, which was created expressly for rapid design and development of dynamic web sites. In fact, PHP suits its purpose so well that few languages can compete with its ease and speed of development, making it an ideal choice when implementing an agile development strategy. It's no surprise that fewer PHP projects have been specified, designed or implemented in the classical waterfall style which is probably for the best, given that project specifications are a bit like quicksand —they look rock solid until you try to build anything upon them. Application code is frequently developed quickly in short iterations to react effectively to business needs and is therefore required to be deployed in the same manner.

This white paper will demonstrate a surprisingly simple and yet robust way to solve the problems associated with application deployment for PHP development projects while avoiding the substantial costs. Leveraging the deployment automation features of Zend Server, Zend's PHP enterprise grade application server, in concert with Zend Studio IDE, allows organizations to streamline their application deployments and eliminate costly manual work, reduce the risk of errors and enable the IT organization to deliver the agility needed to respond to business requirements faster. Implementing a deployment automation practice delivers the following benefits:

- The process of application deployment is reduced to trivial automated tasks requiring no specialist application knowledge.
- No mountainous learning curves. The configuration files are automatically generated and any required deployment guidelines can be written in PHP which is already widely used by the development team.
- Your application will never outgrow the deployment process since the exact same packages that are

produced to be deployed to heterogeneous data centers or elastically scaled cloud based clusters can also be used on a single virtual machine running within a developer's laptop.

PHP Application Development Practices

It is a fact that modern virtualization technologies have served to make life easier for developers and the IT administrators who support them. Furthermore, the availability of free and fully functional virtualization solutions makes using these technologies a no-brainer. With this in mind, PHP development teams should be aiming to implement the best practice application development model visualized in Figure 1.

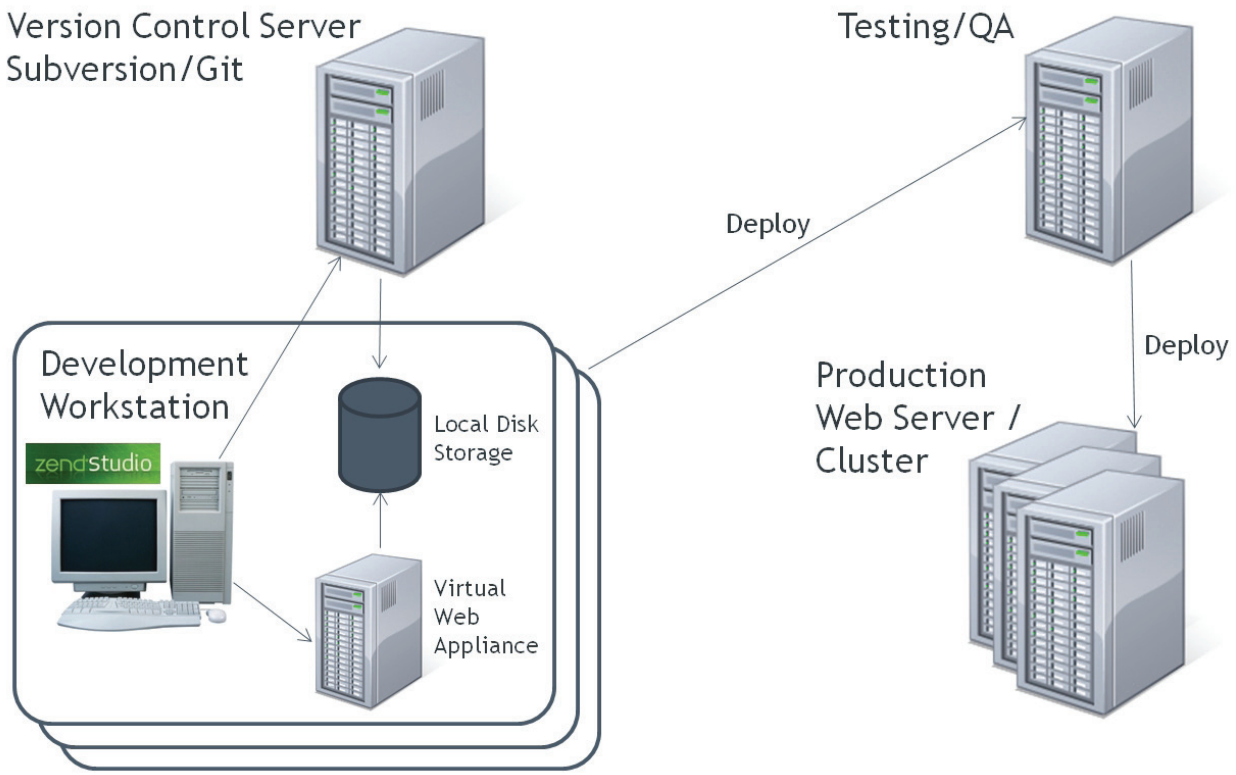

Figure 1 – Best practice application development and deployment model

The key design principles of this environment are:

- Every developer has an exclusive copy of the application code in addition to a local sandbox in which that code can be tested. This is particularly important when a subset of the development team is tasked with radically changing the application, perhaps requiring the use of different libraries or PHP modules. A local development model ensures that developers can function independent of their working environment and co-workers while also maximizing the performance of their development tools. The development appliance can be crafted to closely resemble the integration/production environment while still allowing individual developers to use whatever operating system they are most comfortable with. Furthermore, the work of creating the development appliance is something that system administrators only need to perform once before rolling it out to all developers with a tool like Vagrant.
- All software is maintained in version control. This provides a central source for new developers joining the team and the project midstream, and also allows for simplified backups and disaster recovery.
- When the developer has completed implementing and unit testing new functionality, the code is checked into version control. The developer then performs an application push to the integration/QA server. It should be noted that this is not a function of version control, as guaranteed repeatability is required.

Although version control can simulate repeatability, with the use of tags, it is all too easy for a member of the development team to delete the all important tag. Continuous integration solutions assist in automating this "push" and help to ensure code quality by providing hooks to execute a suite of validation tests and finally package the code when those tests pass. An example of this type of automation would be the marriage of CI tools like Jenkins and the powerful deployment APIs exposed by Zend Server. With this pairing at a developers fingertips, it is easy to automate the check out, test and packaging of the latest code and then deliver the resulting package to the QA environment all with a single button push in a browser.

- When QA or the integration team is satisfied that the new version meets required quality standards, the same application code is re-deployed to the production environment. This similarity is ensured by virtue of the fact that the same tested deployment package can be directly reused.

Development workstations are traditionally very hard to maintain, a fact that leads to many deployment problems down the road. Inevitably, the IT administrators of the developers' workstations must support a number of operating systems, and possibly a number of different IDEs, as well as users at different levels of technical knowledge and skill. By migrating all of the application dependencies into a virtual application server appliance, the majority of software incompatibility problems can be solved. Workstation requirements can be reduced to a version control client, an IDE and the virtualization client program. It is important to note that the working copy of the application should be stored on the hard disk of the workstation itself. It is far more convenient for the user; and most IDEs, especially those with advanced code analysis features, will perform radically faster when accessing local files.

In smaller projects, it is easy to underestimate the value of a testing and staging environment, and therefore it may be tempting to avoid the setup of such an installation. However, development machines are diverse and volatile and as a best practice it is vital to have a dedicated testing environment that should resemble the production server(s) as closely as possible including running the same application server and PHP stack. This is the safety net to catch the missing settings and application components that would otherwise cause frantic redevelopment/deployment in the early hours of release day.

This brings us to the issue that PHP has classically lacked a standard application deployment mechanism. This drove PHP developers, the highly inventive bunch that we are, to come up with a multitude of techniques to work around this problem, each with its own drawbacks. Common techniques used over many years of working with PHP include:

- Working directly on the application server via SSH and editing with vi/vim
- Using a shared file system (Samba/NFS)
- Performing a version control check out on the production server
- Using rsync to "clone" a development environment
- Creating deb/RPM packages
- Using Jenkins or other continuous integration tools

However when promoting the application through QA and then to staging and production environments, developers typically do not have access or control over the infrastructure and the set of deployment steps turn into a set of instructions to be performed by the IT operations team. The operations team often relies on the availability of the tools, settings and scripts used by the developers earlier to deploy and configure the applications in their development environment.

Many problems can therefore descend upon the poor release manager in the middle of the night. After much troubleshooting and analysis, the root cause usually stems from human error. The reliance on manual tasks developed originally for a different environment presents a problem when a list of installation steps pages long must be executed in short order to meet the maintenance window restrictions. To further complicate matters, the manual instructions are usually written by developers who typically have not invested the time nor have

the experience to account for all the aspects and dependencies of deploying to a production environment. All this leads to a degree of anxiety and uncertainty over whether the deployment will be a success and whether the application will be functional the next day. Experience shows that the majority of software deployments fail, or fail to run smoothly, for these exact reasons. It comes as no surprise then that surveys researching the causes of application downtime have shown that roughly 25% of overall application downtime is caused by human errors.

Effortless Deployment Automation with Zend Server

This is where the deployment automation and management features of Zend Server come into play. In contrast with manual processes and other deployment techniques, home-grown or otherwise, there are no complicated software packages to install. The complete deployment automation framework, along with a web server and certified standardized PHP stack included in the PHP application server can be installed on Linux, IBM i or Microsoft Windows Server. With the completion of Zend Server installation, the new instance of the application server is ready for use as a deployment target for the application. Assuming a reasonable internet connection and minimal system administration experience, one can go from bare metal, virtual machine or a Cloud instance, to having a functional application server in less than an hour. This solution also benefits developers, who only need to know the server's IP address in order to start pushing applications to it.

The deployment process begins with the developers who are responsible for creating an application package. From the developer's point of view the first task when joining a team is setting up a development environment on his or her new workstation which will be installed with a version control client, the web appliance, and a code editor usually in the form of an integrated development environment (IDE). While the developer is free to use any IDE or code editor, many choose Zend Studio PHP IDE as it incorporates the Zend SDK which offers deployment integration with Zend Server while also providing a simple set of wizards guiding developers through the creation and generation of the application deployment packages. When creating a new project in Zend Studio, the developer has a number of application deployment options. Once the project has been created, the deployment configuration is automatically generated and can be customized by the developer. Once completed, the developer can simply click on the "Deploy a PHP Application" link to automatically create a deployment package and push it to the pre-configured deployment target. The developer can specify project and deployment package details and, among other things, provide a file containing the application license agreement to be attached for display during installation. The developer can remain blissfully unaware of steps performed to build the finished package and execute the deployment. They simply need to click a mouse button and check the results in a web browser. There is no need for access to the integration/QA server and so the QA environment will remain a faithful representation of the production environment. By extension, if the application runs well in the QA environment, there can be every confidence that the same deployment process using the exact same deployment package can be safely performed by the IT administrators on the staging and production servers.

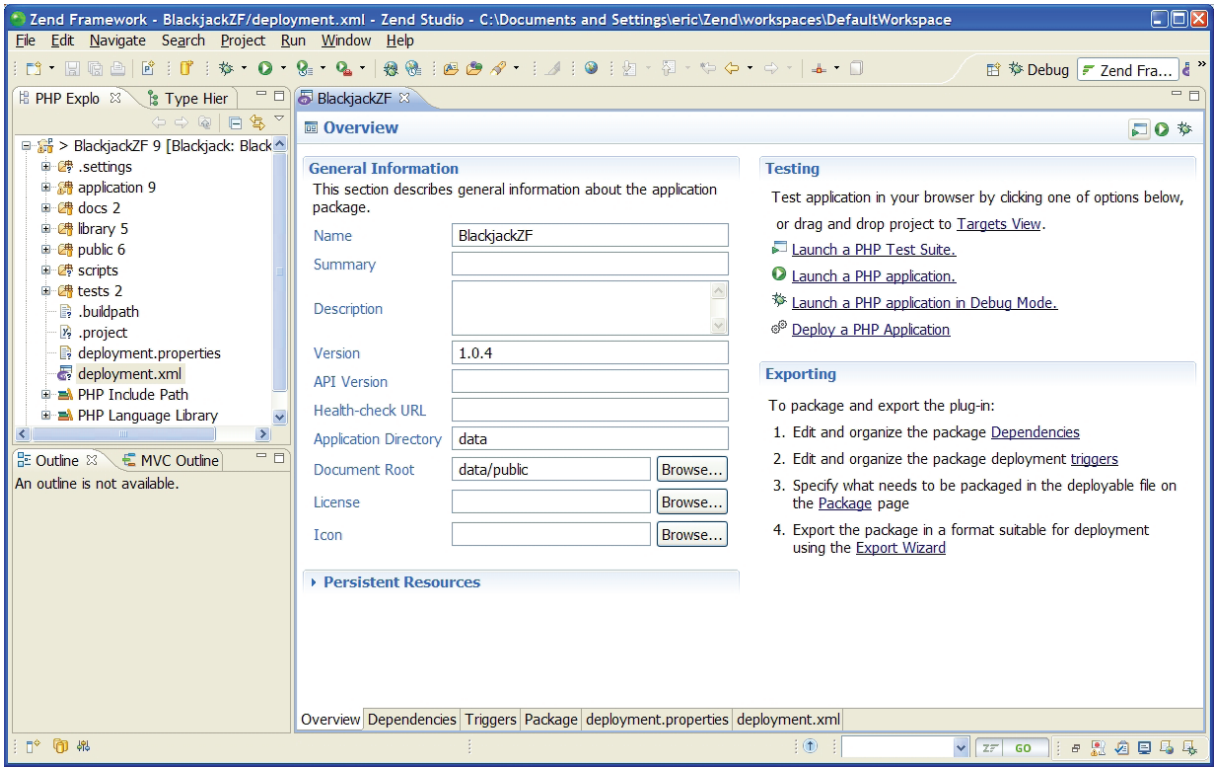

Figure 2 – Deployment configuration in Zend Studio

Although it is highly advantageous to keep the configuration of the QA servers and the production environment under the tight control of system administrators, this is not always possible or quite simply does not happen. In any case, upgrades to install security fixes can easily cause problems. The Zend Server deployment mechanism provides comprehensive dependency checking to allow the development team/manager to ensure that application pre-requisites are validated before the application is deployed. Two examples of the benefits of dependency checking relate to the use of external libraries. An application change may force the use of the latest version of a library that imposes a specific minimum version of PHP. This is quite common when using open source frameworks for example. In another example, a legacy application may be found to malfunction with newer versions of PHP and thus should not be deployed on servers that have been upgraded. This is very easy for the developer to configure. Simply selecting the dependencies tab of the deployment viewer in Zend Studio makes it possible to outline the application requirements as shown in Figure 3. Validation of the application environment covers its external dependencies including:

- PHP version
- Zend Server version
- Specific versions of separately deployed libraries whether that be Zend Framework or customer provided libraries
- php.ini settings
- Presence of PHP modules and their versions
- Presence of Zend Server components

In this way, a developer can ensure that a particular deployment will not fail because of intended or unintended changes that have been performed on the production servers. A package configured with dependencies will simply refuse to deploy until they are met.

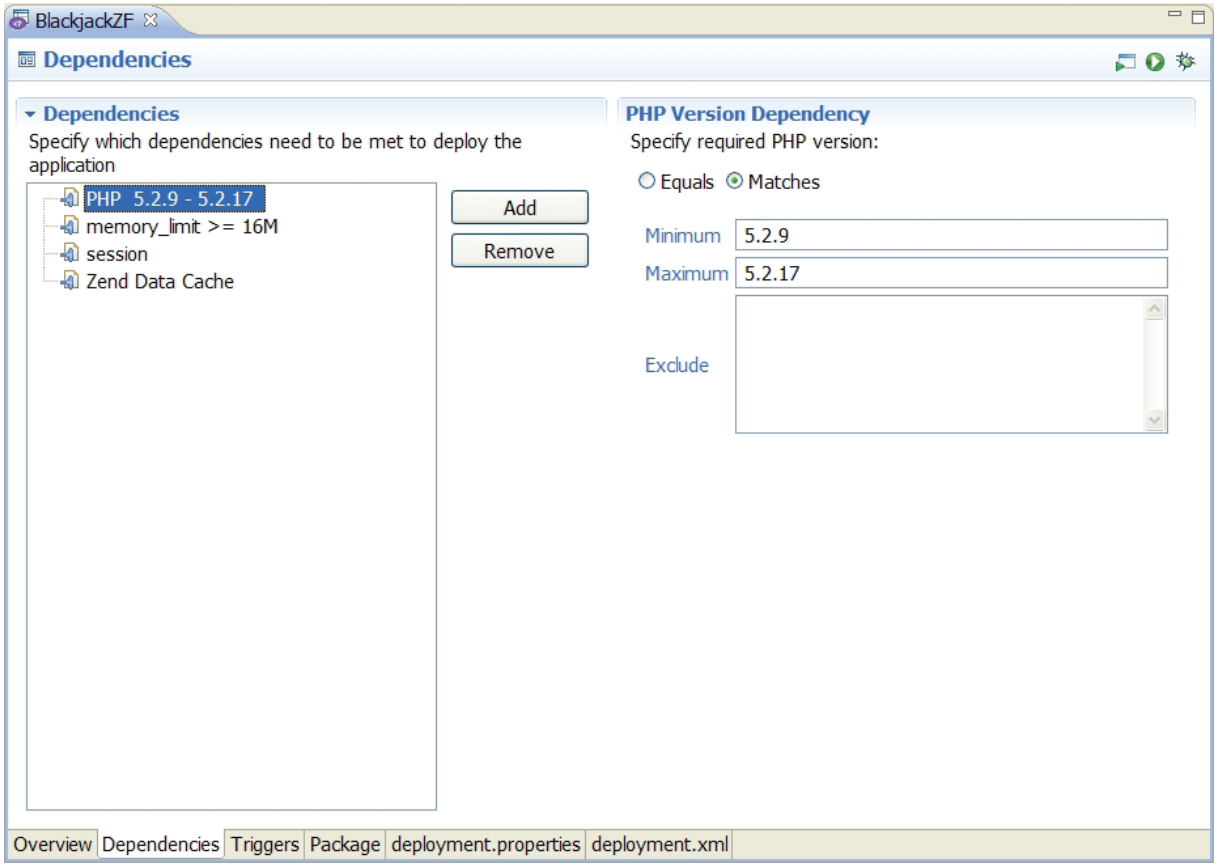

Figure 3 – Defining dependencies

One of the biggest challenges facing release managers is the fact that few modern applications can be simply packaged and pushed to a remote server. This explains the laundry list of instructions that must be included with each release, and the frequency of problems that occur during very short deployment windows. The amount of time, skills and manual efforts required by IT operations to fulfill the deployment instructions raises significantly the cost of releasing new applications and effectively limits the ability of the organization to implement true agility in addressing business needs. Due to the fact that many deployments fail or generate a wide spectrum of errors, often difficult to detect, release cycles tend to be slow.

To solve this challenge and completely streamline the application deployment process even when complex operations are required, Zend Server's deployment mechanism supports a number of trigger scripts which are called at various points in the deployment process. By including one or more of these scripts in the deployment package, a developer can arrange for the customization of an application for its target environment. Given that developers know exactly what needs to be customized in their applications and have the skills to write PHP scripts to perform this task, they are better suited to the task than the release manager. It is possible to write pre- and post-triggers for the five different stages of the deployment process:

- Staging – Copying of the application package to the target server(s)
- Activation – Publishing the application on the target server(s)
- Rollback – Reverting to a previously installed application version

- Deactivation – Removing the application from the public view
- Un-stage – Cleaning package files from the target server(s)

Because these scripts can contain any logic that can be expressed in PHP, they can be exceptionally powerful without requiring that the developers learn any new scripting style or learn to use a generic IT release orchestration tool. So, whether the new version of the application requires database schema changes, a maintenance page to be displayed during the update, or simple configuration adaptation for the QA or production environment, this can be planned for ahead of time. To ease the writing of trigger scripts, Zend Server provides a number of predefined parameters, including the directory where deployment package files were unpacked to, version numbers of the current application, the version being deployed, and the version of PHP installed on the application server. Obviously this information would be invaluable when performing the aforementioned tasks. Frequently there are cases where additional information must be collected from the person performing the deployment in order to allow the trigger scripts to function correctly (e.g. if a database server needs to be specified). In these cases it is possible to define parameters whose values will be prompted for as part of the deployment therefore allowing the developer to provide guidance to the release manager and IT operations staff while keeping the flexibility to apply environment specific inputs. These same guidelines and built in configurations will migrate with the application in testing, staging and production without the need for the IT administrators to develop their own manual scripts. The example shown in Figure 4 allows the application's environment type to be specified by the release manager, and then leveraged within the trigger scripts to allow target specific operations. The default value used here ensures that if this parameter is not modified by the release manager, a safe default value defined by the developer is assumed.

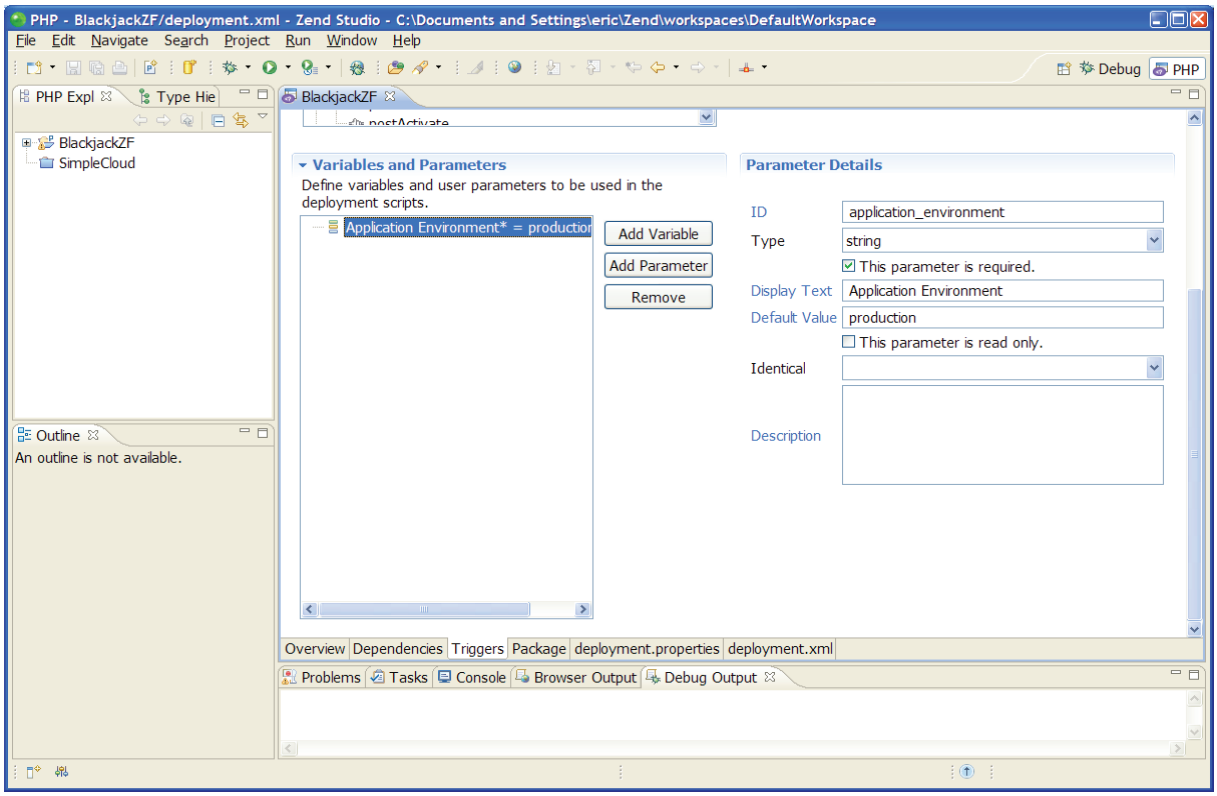

Figure 4 – Defining application environment parameters

The developers are therefore provided with powerful yet simple to use solutions to push their code changes out to the QA server. The release manager would take the deployment package produced by the developers and, after extensive testing in the QA environment, deploy it to the staging and production servers. Two things allow this to be done quite easily. First, Zend Server keeps a copy of all the deployment packages it receives in a packages directory and secondly provides the administrator with an interface allowing easy deployment of said packages to stand alone servers or throughout a cluster. By providing a fully programmable deployment engine, Zend Server delivers an automated solution within a very open and flexible environment which allows release managers to completely control the deployment to staging and production environments, under their management, and integrate with other systems such as workflow automation solutions or cloud management software.

Imagine it is 2:00 am and our release manager has just deployed a well-tested package out to the production servers. The deployment was apparently successful but nevertheless, against all odds, the release manager observes that the production website is mysteriously offline. A growing sense of panic is not calmed by the fact that the people who could help diagnose the problem are at home, asleep. Fortunately, Zend Server's deployment feature also has a rollback mechanism. Rather than having to diagnose the problem immediately, a short visit to the Zend Server GUI will allow the previous application version to be recovered. The trigger scripts associated with the rollback ensure that any database modifications, for example, are also rolled back. When the developers arrive in the office, they can review Zend Server event logs to determine what went wrong and correct the issue while the production servers continue to generate revenue. Thus a potentially damaging failure is reduced to a slight deployment delay.

One of the top initiatives for IT organizations is evaluating or actively migrating some of their applications to the cloud. This trend has driven the emergence of a plethora of cloud vendors offering complete management of the production infrastructure and supporting automatic scaling and provisioning of the servers. The good news is that the Zend Server deployment mechanism works exactly the same way on-premise and in the cloud. In fact, a number of the advantages already discussed point to Zend Server's unique suitability to this purpose. In the cloud, there is little to no control over the physical hardware being used. As seen earlier, this is not an issue given that Zend Server performs most of the work on our behalf. Furthermore, the deployment mechanism is designed with elastic scaling in mind. When an administrator or a cloud-based elastic service adds more application servers to the cluster, Zend Server ensures that all settings are copied to the new nodes and the deployment mechanism ensures that all managed applications and libraries are automatically deployed to the newly provisioned cluster nodes.

Summary

As enterprise applications take on increasingly business-critical tasks, IT departments must support more rapid development iterations. Risky, error-prone deployment processes that slow things down significantly increase the cost of delivering applications and damage the realization of their business value.

Zend Server's deployment automation mechanism provides organizations with a highly intuitive solution to eliminate the error prone, repetitive and time consuming manual work that is associated with releasing an application to production. It diminishes the uncertainties and potential failures that can beset complex projects on release day. As a complete application deployment and release automation solution for business critical PHP environments, Zend Server provides the set of tools for packaging and deploying applications. It also provides scripting functionality to deliver the flexibility required for deploying modern and multi tiered complex applications such as those with databases to be upgraded, application server changes to be made, or whose application configuration needs to be adapted to the target environment. Once the application is packaged, Zend Server allows the release manager to deploy the packaged applications throughout their environments located on premise as well as in the private or public cloud in a completely automated manner. If predefined requirements are not met, such as the version of installed software libraries, or if included setup scripts fail to execute successfully, deployment will be interrupted. Finally, if the updated application still fails despite all the checks that have been performed, the release manager has the option to roll back the latest deployment. Zend Server's deployment methodology ensures that the production servers will be provisioned and the application will become productive quickly, allowing organizations to respond to business needs and eliminate the costs associated with failed deployments and time spent on manual installation and configuration tasks.