



**A New Approach to Deploying and Managing  
JavaEE™ Applications**

**A White Paper**

Phurnace Software, Inc.  
[www.phurnace.com](http://www.phurnace.com)

## Introduction

Deploying and migrating JavaEE applications is hard work. Specifically, it's work that's error-prone, repetitive, and time consuming because of the complexity of setting up or tweaking web application servers. The result is lost man-hours, soaring costs, and potential problems associated with hidden or unseen configuration issues that might pop up at a later time. Furthermore, often the situation is muddled by a "fog of war" -- technical personnel and their managers really don't have any easy way to take an accurate snapshot of current application server configurations, and no easy way to make decisions regarding those assets and how they can (or can't) support an enterprise's line of business.

Virtualization, stack applications like Portals, and now the entry of cloud computing only exacerbates configuration challenges. Increasing abstraction away from the core hardware has resulted in more servers to manage. Virtualization has resulted in a proliferation of images across the data center. Stack applications, with their own separate configuration space, are increasingly complex to manage. As these applications are becoming increasingly easy to develop, they are becoming harder and harder to maintain and correctly configure.

Frankly, the situation can sometimes seem like a big house of cards right before the catastrophic tremor. One day everything is fine, the next, maybe not so fine. The tipping point itself can be something benign (upgrading various servers with a new patch) or complex (bringing whole new server environments online to support marketing or sales efforts), but throughout the exercise there's a sense of unease driven by how opaque and fast-changing the environment can be.

In response to this environment, Phurnace has crafted a philosophy and a software application that accelerates the deployment of JavaEE™ applications. In doing so, they have eliminated the error-prone, repetitive processes and headaches associated with the set up and migration of software running on web application servers -- regardless of vendor or version (e.g. IBM WebSphere, Oracle WebLogic, JBoss/Tomcat, etc.).

This white paper presents three brief use cases that examine the ways that JavaEE applications are moved across environments. We also show you how the Phurnace philosophy can be applied to these use cases to minimize errors, increase deployment speed, and assist in troubleshooting applications in ways that don't otherwise exist in the marketplace.

## Our Philosophy

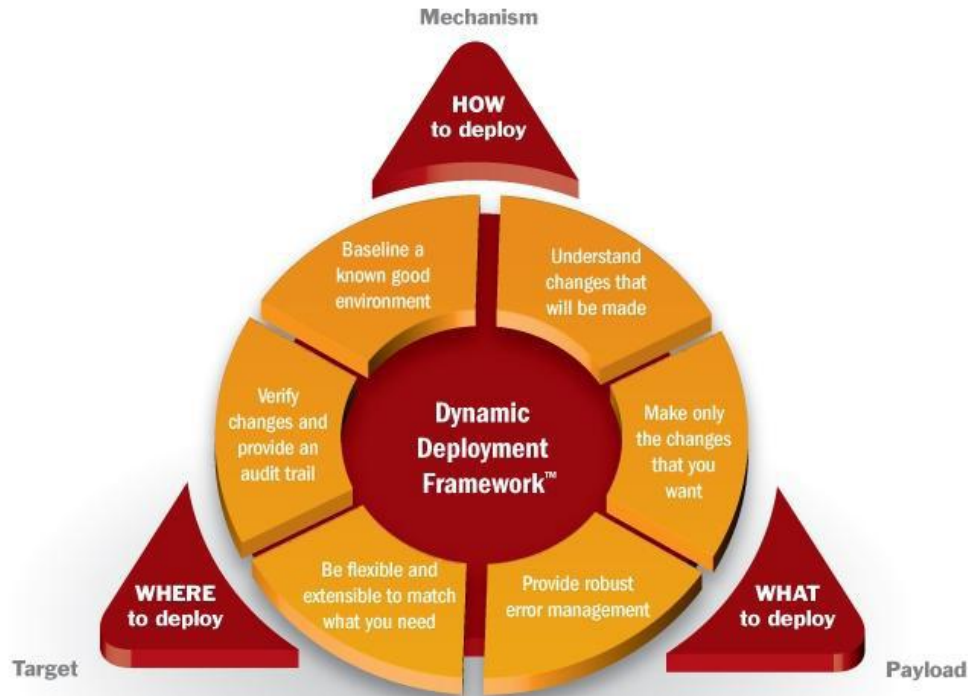
Our philosophy encapsulates the way we think deployments should be managed. Today, deployments are most often thought of as atomic instances, as actions, rather than as a process. At Phurnace, we think that deployments should be treated holistically. For example, deploying EAR files should not be parsed apart from configuring a server. The method for deploying to the test instance should not be different than the method used in staging, production, or development. Furthermore, troubleshooting a configuration is just as much a part of deployment as any other step. At the end of the day, all of the pieces need to be considered together to make the application work the way it was intended.

In short, deployments are dynamic because they go through a growing, changing and unpredictable life-cycle. The professionals who manage the application environments are forced to make assumptions about the use and growth of those environments; for the most part those assumptions are usually correct. But over time, those environments will invariably change in unexpected ways: new technologies are adopted; business requirements change; even the personal style and best practices of the IT professionals managing the process may evolve. All of these issues

make the application environments highly variable.

Each deployment into an environment can be quite different, presenting many problems unless the deployment process is flexible enough to adapt to those differences. Deployments inherently are dynamic because each environment may differ slightly from every other environment, and the environments themselves may be in flux. Also, keep in mind that many companies today may develop in one environment and deploy in another (e.g. develop on JBoss and then deploy on WebSphere).

Our way of thinking provides an opportunity to conceptualize the deployment process and approach. Figure 1 outlines the steps and components of an effective deployment framework.



**Figure 1: The Phurnace Framework**

We believe that a deployment process must be able to:

- Baseline a “known good” configuration environment.
- Understand what changes will be made if that configuration were to be applied to another environment.
- Apply only those changes that are required and provide a point of control for “optional” configuration changes.
- Provide robust error management including rollbacks, uninstalls, logging (for debugging), and troubleshooting.
- Support environments of virtual, physical, or cloud based infrastructure.

- Be able to drop seamlessly into an existing application life-cycle process.
- Provide extension points for other configuration parameters and be flexible enough to deploy to the needed environments without undue reliance on infrastructure.
- Provide an audit trail for changes and monitor the application to verify that it continues to be configured in the way that the business requires.

Furthermore, the above steps must be combined with three component parts. Unfortunately, many people today do not separate these three parts for deployment and this causes problems. The three components are colloquially termed the What, the Where, and the How. If these three components are intermingled, the result can be a very messy problem. If you separate them, the result is always a predictable, repeatable process that eliminates runaway errors and costs. The three component parts are specifically known as:

- **Payload (the What)** – the bundled application binaries (EARs, WARs, etc.) and the desired configuration (JDBC, JMS, etc.). The configuration should be abstracted from and independent of any target specifics (such as topology, application server vendor, and version, and infrastructure (virtual, cloud, physical)).
- **Target (the Where)** – environment-specific details (hostname, db location, cluster size, etc).
- **Mechanism (the How)** – application that can deploy any element in a payload to any given target, and also generate payloads from a target.

The decoupling of the payload, target, and mechanism are critical to the successful implementation of a successful deployment process. Properly implemented, this process enables consistent, repeatable, and predictable deployments.

### Examples of our Philosophy in Action:

The following are brief examples of specific use cases that demonstrate the value of our approach. While not inclusive of all of the benefits, the examples present concrete reasons to adopt this philosophy.

#### Use Case 1: Modeling an Existing Application Infrastructure

This use case is most commonly done at the beginning of a deployment – a simple baseline to determine what is or is not existent on the server, either as an inventory exercise or to prepare for a deployment change or upgrade.

Doing even this simple activity with existing methods can be difficult. With most application servers there are really two ways to go about doing an inventory, and both are fairly cumbersome. The first is to page through the management console and note the applications installed, how are they configured, and what other configurations are present (JMS, etc). A second way involves parsing the raw configuration files from the server to discover the required information. Neither approach is simple; both feature multiple opportunities for error even if done by experienced staff.

An increasing number of servers now live in virtual environments, and the speed with which they are brought up

and down, cloned and moved, is increasing rapidly. The inventory of applications and configurations generated yesterday may not be accurate today. In fact, it probably isn't. But that doesn't mean that creating those inventories isn't important anymore – in fact it is more critical than ever for exactly that reason. What is obvious, though, is that using a manual process to generate the inventory just won't work.

Phurnace's approach transforms the modeling task into a straightforward procedure. The mechanism defined in our approach has the ability to not only deploy applications and configurations, but also generate payloads from a target environment. For this use case, the deployment mechanism would be pointed at the target to generate an abstract representation of the target's configuration. This output would serve as both a model for the application infrastructure and a baseline for applying that same configuration to another environment.

For example, with Phurnace's product line, the output is an XML representation (human readable) of the entire application configuration. This output is both easy to generate (because the mechanism is doing all the work for you) and not subject to error. Furthermore, the XML representation is an extremely portable semantic representation of the configuration. In other words, technical personnel can use a text editor (any text editor, really) to read the contents, or feed the contents of the file to any process to transform the captured semantics into some other output, be it an HTML rendering or a set of instructions.

## **Use Case 2: Migrating Applications from Development to Quality Assurance (QA) to Production**

Every JavaEE application goes through a development/test life-cycle. Most commonly, promoting applications from environment to environment is achieved via two routes: manual or scripted deployment. In our customers' experience either route is problematic. Manual deployment is not only time consuming, but also extremely error-prone. The scripted approach doesn't give much advantage, as scripts must be continually updated and kept in sync between environments. In either case, if something goes wrong during deployment the target environment can be left in an unknown, "Frankenstein-like" state – neither on the previous release or the new one, but some amalgam of both.

With our approach, deploying from development to QA to production is straightforward, automated, and robust. For example, let's imagine that the target environment for deployment is a two-way WebSphere cluster. Prior to deployment, the server variables would be identified and defined, most likely in a simple properties file containing all the environment variables that should change from one deployment to the next. For simplicity, we will assume that the only variables that will change from one environment to the next are: hostname, admin port, db location, and topography. In this case, the properties file would be a simple key-value pair for each of those attributes.

At deployment time, the payload would consist of the actual application binaries, the XML document that describes the configuration, and the server properties. These artifacts would be fed into the deployment mechanism, which would, on the fly, translate the environment variables and payload into the target environment. Hostname, admin port, db location, and topography are automatically merged into the generic configuration document using variable substitution and the previously defined key/value pairs.

The deployment mechanism then interrogates the target server and only makes the changes necessary to the environment to make it match the abstract configuration (after first identifying the changes that it will make). If at any point in the deployment process something goes wrong, then all the changes are rolled back, leaving the environment exactly as it was prior to the attempted deployment. Verbose logging of the deployment aids in the identification and correction of the underlying problems.

When the process goes well and it is time to deliver the application to a “production” environment, the same mechanism and configuration document are used to deploy the application (as was used in the test deployment), even though the production environment may be a series of virtual images. The only change is to the pre-defined server properties, which should remain fairly static across time. What is tested, both application and configuration, is what gets deployed.

### **Use Case 3: Troubleshooting a Configuration Problem**

Let’s assume there are two servers with the same applications installed on them. One works and one doesn’t. This is a surprisingly common situation in installations.

Today, comparing those servers in an effort to figure out “what’s different” is an arduous task. “Diffing” the raw configuration files is a daunting if not impossible task – most configuration files contain too many false positives. The only other existing method, going screen by screen in the management console, is so difficult that it rarely makes the list of “reasonable” approaches.

Fundamentally, our approach recognizes that troubleshooting is part of the deployment process. Because it assumes the ability to export configurations from servers and present that information in a standard, abstract format, it now has the ability to compare those configurations side by side and show only those differences that matter. In our customers’ experience, approximately 25 to 40 percent of all their application defects are configuration related. With the ability to quickly identify a configuration error, or at least rule out configuration as a potential source of the behavior, conflict resolution is faster, easier, and more specific. In addition, if you utilize an application like Phurnace in your process, this situation is much less likely to occur in the first place.

### **Conclusion**

Deploying applications on web application servers doesn't have to be an arduous, expensive, and time-intensive process. Using the Phurnace philosophy, organizations can separate payload, mechanism, and target and create fast, automated deployments without the use of scripts. Furthermore, our philosophy delivers these benefits:

- “Preview” to identify configuration changes prior to deployment.
- Update and upgrade from any previous web application server version.
- Quickly troubleshoot and compare faulty configurations with a “known good” server.
- Deploy applications regardless of whether they exist on physical, virtual, or cloud hardware.
- Provide the same functionality to WebSphere Portal customers.
- Create an inventory of your existing configurations.
- Check-in your configurations with your source code.
- Audit existing configurations to make sure they comply with a baseline.
- Roll-back in case of errors at deployment time.
- Ensure that what is tested is, in fact, what is rolled out to production or to your customers.

## Find Out More About Phurnace Deliver™

Here at Phurnace we have leveraged this new way of thinking to create a line of products that can dramatically accelerate your deployments. We call the suite Phurnace Deliver™ which offers an easy, agentless way to implement all of the core features mentioned above; based on the separation of the three components of payload, mechanism, and target.

Please learn more - Request a product demonstration at <http://www.phurnace.com/forms/evaluate-phurnace.html>

## About Phurnace Software, Inc.

Phurnace creates software that accelerates the deployment of Java EE™ applications. It eliminates the error-prone, repetitive processes and headaches associated with the set up, configuration, and migration of software running on web application servers -- regardless of vendor or version. Developers and system administrators use Phurnace to reduce errors, save time, and reduce overall deployment costs.

**Phurnace Software, Inc.**  
**707 Rio Grande, Suite 100**  
**Austin, Texas 78701**  
**512.382.1050**  
<http://www.phurnace.com>