BladeLogic, Inc.

Technical Disclosure Publication Document

Neeran Karnik, Amol Vaikar

Image-based Server Provisioning using Superset Stacks

Posted: March 23, 2012

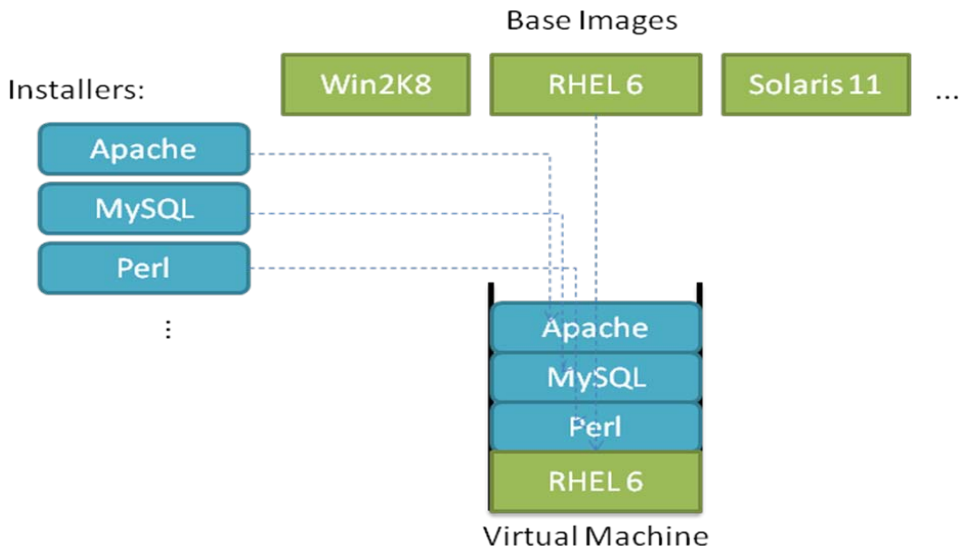
Overview

This document describes a system for rapid, image-based server provisioning that minimizes the management overheads of maintaining a large library of images. The key novelty of the solution is that instead of identifying useful *subsets* of software stacks to pre-provision, the administrator identifies a (likely much smaller) number of *supersets*. For software in a stack that typically co-exists on a server (e.g., a Windows J2EE stack, or a LAMP stack), instead of maintaining all (or some) combinations of images, the administrators maintain only one image that has *all* the stack components installed and configured. Once a machine is provisioned using a superset stack, the excess software in that stack is deactivated and optionally uninstalled.

Background

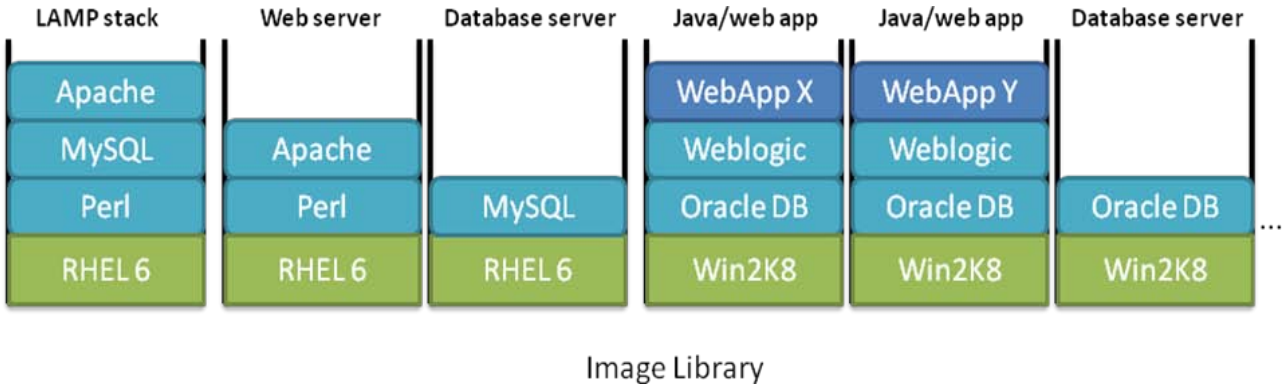
Cloud and provisioning solutions such as BMC Cloud Lifecycle Manager and BMC Server Automation, are moving to "full stack provisioning." It is no longer enough to provision a barebones physical or virtual server with just the base operating system. In fact, users have varied requirements for the software stack on top of the operating system. Different users may request different combinations of applications, installed on top of different operating systems. As a result, there are a large number of different stacks that users can potentially request.

To fulfill such requests, the underlying provisioning system can adopt different approaches, which result in a spectrum of tradeoffs between *management complexity* and *provisioning speed*. At one extreme, the administrator could maintain a small set of base operating system (OS) images and a large collection of software installers. An OS image would be used for initial server provisioning, followed by copying and executing the desired set of software installers on the newly-provisioned machine. An example is shown below:



While this minimizes the library of images that has to be pre-created and stored, the provisioning process is very time-consuming. This is because each installer must be copied over the network to the new server, and apart from copying the application bits to disk, the installer also applies a significant amount of business logic – performing environment validations, for example. As software stacks grow larger and more complex, this performance overhead becomes even more prominent.

At the other end of the spectrum, the administrator could pre-create images for all possible service offerings – each image containing the base OS as well as a pre-installed software stack. Provisioning would then merely consist of streaming the image to the server over the network, and writing the bits to disk. The figure below depicts what such an image library could look like:



This would yield fast provisioning times, however, the diversity of software and OS versions and their possible stacks leads to a combinatorial explosion in the images that must be created, stored, and managed over time.

A hybrid approach is to create images only for commonly required *subsets* of software stacks, and run installers for any additional software specified in a user request. While this controls the image proliferation problem to some extent, the performance of such a solution would not satisfy the typical expectations of a cloud customer. Also, the library of subset images keeps growing over time, adding to the management overheads. Note that these problems are also applicable to the *pre-provisioned*-machine-pool approach to server provisioning, wherein a cache of already-provisioned (physical or virtual) machines is maintained. When a provisioning request is received, the closest machine from the pool is chosen, customized (by running additional installers), and delivered to the user.

The best one can achieve (in the prior art) is a tunable tradeoff between provisioning speed and management overheads, using the hybrid approach above. This Technical Disclosure Publication Document describes an inventive solution that breaks this tradeoff.

Solution

An administrator first creates a small number of "superset stacks" (abbreviated as *super-stacks* from here on), based on the services they want to offer to their customers. E.g., the administrator may offer web applications running on a LAMP stack, Java apps running on a J2EE stack, and a test/development environment for use by software developers and QA engineers. They could create the following super-stack images:

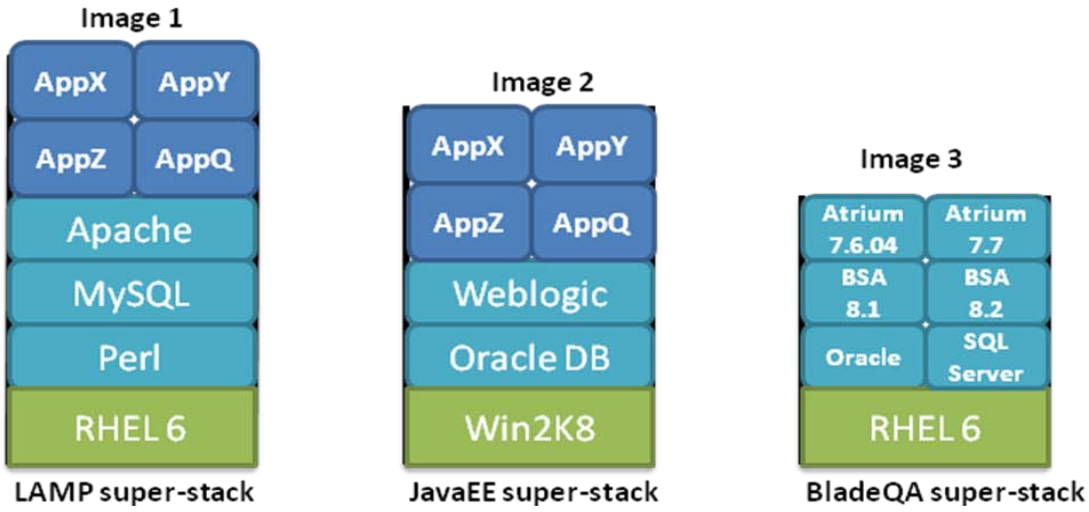


Image 1 contains the typical LAMP stack and also a set of web applications deployed on top. A service offering may include a specific web app (e.g., AppY) running on the LAMP stack, or just a MySQL database server (without Apache, Perl or the webapps). Image 2 contains a set of J2EE components (a Weblogic server, a database, etc.) on a Windows OS along with some pre-deployed WARs/EARs. Image 3 contains two versions of the BMC Bladelogic software, two versions of the BMC Atrium CMDB

product, two different databases, etc. A QA engineer could request a specific combination of these components for testing or integration scenarios.

When a request for *any subset* of the super-stacks above is received, an appropriate image (or pre-provisioned machine) can be used to complete the request. Based on the requested components, it can be determined which super-stack to use. For instance, assume that a user asks for a server with Linux and MySQL on it – perhaps the user is looking only for a MySQL database server on Linux, not the entire LAMP stack. The provisioning solution will then choose Image 1 and provision the requested machine. In general, the system would maintain metadata such as a manifest of each super-stack, describing the components installed in it. A user request would contain a list of desired components. The system could choose the *smallest* super-stack (by image size) that contains all the user-requested components. This image would then be provisioned using standard image-based provisioning techniques specific to the underlying platform (physical or hypervisor).

In the alternate approach of using pre-provisioned machines, the above provisioning step would be performed ahead of time – not in response to a user request. Using each super-stack created by the administrators, several machines are provisioned and added to the pool. A user request is satisfied out of the pool where possible. As before, the machine with the smallest super-stack that contains the requested components is selected.

Next, one of the following approaches can be used for dealing with software components that were not requested, but happen to be installed on the newly-provisioned machine (Apache, Perl and some web applications in our example):

a. **Do nothing:**
   If the additional software is unlicensed by default, its own license enforcement mechanisms would prevent the user from using it.

b. **Delete critical registry entries, configuration files or binaries:**
   The signatures used by the BMC Atrium Discovery and Dependency Mapping product identify the critical files, registry entries etc. that help discover the software. The system could use this signature database to decide what to *delete*, instead. This will disable the software, and also *prevents its discovery* by the BMC Atrium Discovery and Dependency Mapping product or similar tools – thus leaving any license compliance metrics unaffected. The system would also remove any startup entries that attempt to start the applications on server boot-up.

c. **Uninstall the unnecessary software:**
   This can be done silently *after* the provisioning request has been completed. Uninstallers for all the additional software could be triggered in the background upon the *first boot*. Alternatively, in the interests of

harvesting wasted disk space, the install directories of the software could be deleted without going through a complete uninstall.

This solution has the following benefits:

- Administrators only need to create, store, and manage a relatively small number of super-stack images. This greatly reduces management overhead and storage requirements when compared with the prior art. Also, when the super-stacks are created, only the necessary components of each application would be installed. For example, the administrator could choose not to install clustering or L10N support for a database, if that is not being offered in any service offering. Thus, the image created may be more compact than the corresponding application's installer, which carries the complete set of components.
- Provisioning is significantly quicker. While the image itself may be somewhat larger, no software installers are run on top, which avoids all environment checks, configurations, etc. The post-provisioning step also takes no time (for option a), negligible time (for option b), or is done on first boot (for option c) so that it doesn't add to the user's perceived provisioning time. No time is spent on post-install configuration because that is done during the image creation itself.

The tradeoff is potentially additional disk space usage, and slightly increased time for the initial image provisioning. The superset image is larger than the subset images used in the prior art. The time required for copying the additional bits, however, is likely much smaller when compared with the overheads of running standalone installers that perform checksum validations, environment checks, and configuration tasks (apart from copying the bits). If superfluous components are not uninstalled or deleted from the provisioned machine, disk space may be wasted. This should be an acceptable tradeoff, given that storage is cheap. Even this, however, could be minimized by using a copy-on-read file system or de-duplicated storage underneath, for example, or by deleting the files / uninstalling the application as a post-provisioning background activity.