# Constructing and Managing Appliances for Cloud Deployments from Repositories of Reusable Components

Matthew S. Wilson
*rPath, Inc.*
*Raleigh, North Carolina*
msw@rpath.com

## Abstract

In order to efficiently utilize Cloud Computing environments (more specifically, Infrastructure as a Service offerings), developers must be able to quickly incorporate their applications into integrated systems commonly called software appliances. This paper describes a system that can be used to construct and maintain software appliances called *Conary*. The core of Conary is a software configuration management system that places all the components included in an appliance in a versioned repository. The version control system properties of Conary's software configuration management design naturally facilities image creation and updates. An approach utilizing Conary solves many of the challenges faced by adopters of Cloud Computing.

## 1 Introduction

The availability of virtualization and cloud computing can allow Information Technology (IT) organizations to be more flexible and responsive to business needs [1], but this flexibility does not come without new challenges. Enterprise developers are adopting new platforms (such as LAMP scripting languages [2] and Ruby [3]) to more rapidly and efficiently build applications, but IT organizations are often slow to support deploying them. Much of the time spent in the deployment of new applications is fitting the application onto one of several standardized operating systems. This is a major pitfall in traditional software deployment which is only exasperated by virtualization; a better solution is to build Virtual Appliances [4].

The construction and maintenance of virtual appliances requires a holistic approach. As operating system, support libraries, application frameworks, and application components are delivered as a preconfigured unit, the mechanism that assembles the unit and maintains it over time should be designed to manage the system as a whole. Some projects make appliance creation easier, however they achieve it though existing automated operating system installation and post-install scripting capabilities. The operating system and application are continue to be managed separately and the appliance creator still has to design and implement an update mechanism, be that through distributing new virtual machine images or an integrated online update system.

Conary [7] embodies a new approach to constructing and maintaining complete systems of software. This paper will detail Conary's approach to system management with a holistic software configuration management system. Design considerations derived from Conary's approach will be discussed. Finally, implementation will be discussed.

## 2 Approach

Traditionally software configuration management practices have been applied to components that, when combined, constitute a software product [8]. Typically software configuration management is a software development practice, enabling developers to manage the dependencies and interconnections of source code in order to produce consistent binary objects. This approach is sufficient when the software product is an application that is installed on a general purpose operating system. However, if the product is a fully integrated stack of software — from the operating system to the application — the scope of software configuration management must be widened to cover every included component. Furthermore, once all the components are versioned individually, the amalgamation of these specific versioned components must also be versioned [9].

Once a full set of software is managed under version control and grouped together, several operations come naturally. A deployment image can be generated from the versioned group of software. This deployment image can be used to install on bare metal systems, imported into a local virtualization system's inventory, or deployed on off-premises infrastructure in the cloud. This image is a transient object; it can always be recreated from the repository. The image is simply a vehicle to deploy the application stack.

Once a system is deployed, the same repository of versioned components used to create the image can be used to update a running system. The deployed image contains a database that records the system state including

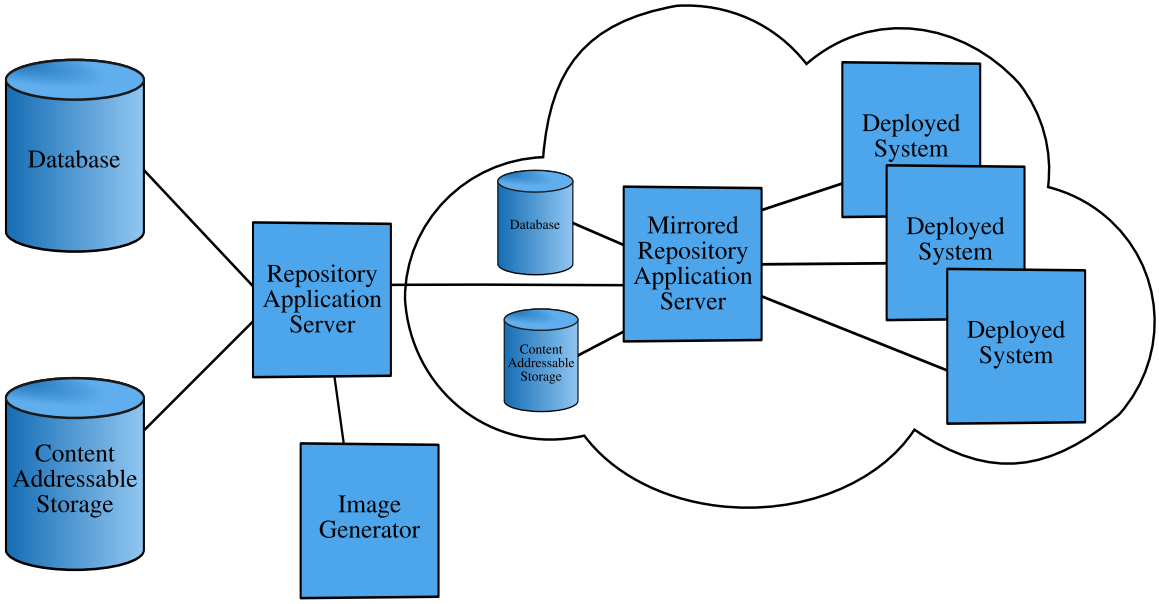


*Figure 1: Conary Repository Architecture Overview*

all the versions of the components that comprise it. Updating the system is much like updating a source code checkout from a traditional software configuration management system. Files are added, removed, and updated in accordance to the versioned objects in the repository. This incremental update function is particularly important when deploying into the cloud where transferring large amounts of data should be avoided.

## 3 Design and Implementation

When laying the architecture for Conary, unique design challenges were considered. Storing files and distributing them to clients had to be efficient for both repository disk space and network transfer. The file format used by clients to obtain incremental changes had to be compact and facilitate operations more akin to software configuration management than package management. Coalescing software components into a functioning, dependency closed set had to be easy for both the developer and the systems engineer. This section will provide information on the implementations that met these design challenges.

### 3.1 File Contents Store

Conary's purpose is to manage large numbers of interdependent software components over time. This results in several design challenges. Because Conary is placing every software component, including the operating system, under version control, an efficient storage mechanism was required. Traditional software packaging systems store the same file contents redundantly as new versions are produced. This leads to wasted storage in package repositories and unneeded data transfer by each client downloading the package to apply an update.

Instead of storing the same file contents in multiple atomic package archives, Conary uses a content addressable storage system to record each distinct file in the repository. A SHA-1 [10] digest of the file contents is calculated. The file is compressed using gzip [11] and hexadecimal representation of the digest is used as the file name. Unlike most version control systems, Conary does not store file deltas. The typical rate of change for files managed by Conary is much lower than source code files. Storing whole compressed files saves the client and the server from reconstructing file contents from a series of file deltas.

Concerns have been raised [12] regarding the safety of assuming that a cryptographic hash is sufficient to declare that two inputs are actually different. These concerns have have been reasonably addressed [13], therefore Conary does not take any additional measures to ensure that SHA-1 hash collisions do not occur.

The space saving benefits of storing file contents using this method have been measured in two Conary repositories populated from software packaged in RPM [14] format. See Table 1. As updated RPM files are imported into the repository, the space saving effect increases. As most servers hosting a software repository have large disks, the space saving benefit is not of highest value; the greater value comes in reducing data transfer across the network to geographically distributed systems.

### 3.2 Changesets

Conary uses a custom changeset file to transmit data to and from a repository. Both the repository and a Conary-

| Repository | Conary Size | RPM Size | Savings |
|---|---|---|---|
| CentOS 5 | 1,080 MiB | 1,307 MiB | 17.4% |
| SLES 10 SP2 | 1,259 MiB | 1,902 MiB | 33.8% |

*Table 1: Storage requirements for packages imported into a Conary repository versus the native RPM package size*

managed system use the same changeset format. When a repository applies a changeset, the resulting new object is stored in the database and unique file contents are recorded in the contents store; when a system applies a changeset, the new object is stored in the local database and the file system is updated to the new state. Conary treats the local system much like a repository. The main difference is that a repository can store multiple versions of an object, whereas a local system stores only one.

Changesets can be either *relative* or *absolute*. A relative changeset contains the differences between two versioned objects and are smaller than absolute changesets. An absolute changeset contains the entire object and can be used to update a repository or a system that does not contain a previous version of the object, or to update a system if the installed version is not known.

## 3.3 Groups

Building a group in Conary means composing software components into a cohesive set, closing the set's dependencies, and recording the specific versions of the components included in the group. This feature enables an appliance builder to truly manage the entire system of software as a unit composed of versioned objects, instead of a loosely coupled set of installed software. Furthermore, during the build process policies are applied to the software included in the group. This catches common errors such as file path conflicts and including multiple versions of the same package. In the future, these policies can be extended to inspect package metadata to discover hints regarding package incompatibility or version dependencies. In cases where programmatic determination of compatibility is unreliable, software providers can use groups to define sets of system software that are known to work correctly together.

Groups are created by processing a Conary recipe file. Recipe files are small Python programs that, when interpreted, locate the requested software components from the repository, ensure dependency closure, and create a changeset that can be committed to a Conary repository. By using Python as the recipe language, Conary can take advantage of object oriented features such as inheritance. In the following group recipe example a superclass is loaded from the "group-appliance" component in a Conary repository. This superclass is provided by the operating system platform and ensures that the minimum system software is included in the group.

```
loadRecipe('group-appliance')
class Appliance(GroupApplianceRecipe):
    name='group-jira-appliance'
    version='1.0'

    def addPackages(r):
        r.add('openssh-server')
        r.add('tomcat')
        r.add('jira')
```

By using inheritance, the appliance builder can focus on the application and its dependencies. Often the required system software components differ between virtualization targets. For example, if an appliance will be deployed in the 32-bit EC2 environment then a Xen-optimized "nosegneg" glibc package should be included in the appliance. If VMware is the target, the `vmxnet` and `vmblock` drivers should be included to optimize disk and network performance. The `GroupApplianceRecipe` superclass handles these differences; the appliance developer can focus on the application. At build time the appliance developer specifies the required targets (32-bit, 64-bit, Xen, VMware, etc.) and the appropriate flavors [7] of the group are produced.

## 3.4 Deployment

Once the full set of versioned software components for an appliance are bound together as a group, a deployment image can be created. rBuilder [15] includes a system that automates this process. A file system large enough to house the appliance software is created. The appropriate group is then installed onto the file system. Any finalization scripts included in the software components are executed in a contained virtual environment. The resulting file system is then manipulated as required for the target environment. For example, the file system can be transformed into a a VMware formatted VMDK image [16] or a HyperV VHD image [17]. To deploy the appliance to the Amazon Web Services EC2 cloud, the file system image is bundled and uploaded using the EC2 deployment tools.

Building an image is a starting point for a software appliance. To efficiently maintain multiple deployed appliances in geographically distributed locations, a Conary repository should be co-located with the deployed appliances. This mirror Conary repository is updated using relative changesets to reduce data transfer over slower links. Mirrors can be configured to propagate only changes made to a particular group, reducing the repository footprint.

There are several options for executing updates on the appliance from simple cron-based scripts to WEBM/CIM solutions. Update coordination between multiple appliances is an area that needs additional thought.

## 4 Related Work

Several other projects and products are constructing virtual appliances. In one such project, virt-factory [5] uses Red Hat's kickstart [18] operating system installation scripting language and Puppet [19] manifests to perform operating system customization and application deployment. One problem with this approach is that the operating system and application are still managed disjointedly. Organizations adopting a similar approach with scripting and automation tools like Puppet must set up multiple software repositories, build synchronization scripts to distribute these repositories geographically, and rely on incremental scripting to manipulate systems from state to state. Neither kickstart nor Puppet have integrated version control systems, therefore an organization must take additional steps to version control these files [20].

Online offerings such as RightScale [24] help deploy and scale applications in the cloud. This is beneficial if your deployment target is EC2, but the same technology cannot be used for other cloud targets, private virtualization infrastructures, or bare metal deployments. SUSE Studio [23] and CohesiveFT's Elastic Server [25] create software appliance images, but updates are not coordinated between all the included software components. When updates are applied, the appliance retrieves the latest version of each installed package from the repository.

## 5 Conclusion

The software appliance model aims to unify applications and operating systems into a single unit which brings new requirements to system maintenance. It is more effective to build manageability into a system from day one than to retrofit [6]. Existing server automation tools help reduce a system administrator's workload but do not make the systems themselves more manageable. As virtualization and cloud computing increases the number of deployed systems to manage, new approaches to building inherently manageable systems must considered. No reputable software development organization would attempt to deliver a quality product without a software configuration management system [21]. It is time the same methodologies are employed to manage entire software systems.

## 6 Availability

Conary is freely available under the Common Public License and can be downloaded from `ftp://download.rpath.com/conary/`. Alternatively, rPath makes Conary available to users under a commercial license. rPath's commercial rBuilder product automates building deployment images for bare metal, virtualization, and cloud targets.

## 7 Acknowledgments

The development of Conary and related technologies is partially supported by the Department of Energy under grant number DE-FG02-06ER84505. Erik Troan assisted by writing a tool to calculate the storage utilized by the contents store versus binary RPMs. Thanks to Elizabeth Yarbrough, who helped proofread, and to the anonymous reviewers for their helpful feedback.

## References

[1] Jeffrey C. Mogul, *Operating Systems Should Support Business Change*, USENIX Tenth conference on Hot Topics in Operating Systems (2005).

[2] George Lawton, *LAMP Lights Enterprise Development Efforts*, Computer, Volume 38, Issue 9 (September 2005).

[3] Steve Vinoski, *Enterprise Integration with Ruby*, IEEE Internet Computing, Volume 10, Issue 4 (July 2006).

[4] Changhua Sun, Le He, Qingbo Wang and Ruth Willenborg, *Simplifying Service Deployment with Virtual Appliances*, 2008 IEEE International Conference on Services Computing.

[5] David Lutterkort, Mark McLoughlin, *Manageable Virtual Appliances*, Linux Symposium (2007).

[6] George Candea, *Toward Quantifying System Manageability*, USENIX Fourth Workshop on Hot Topics in System Dependability (2008).

[7] Michael K. Johnson, Erik W. Troan, Matthew S. Wilson, *Repository-based System Management Using Conary*, Linux Symposium (2004).

[8] Reidar Conradi, Bernhard Westfechtel, *Version Models for Software Configuration Management*, ACM Computing Surveys Volume 30, Issue 2 (June 1998).

[9] Erik W. Troan, *A version to rule them all*, journal entry, `http://ewtroan.livejournal.com/23225.html`

[10] Federal Information Processing Standards (FIPS) Publication 180-1 (1995).

[11] L. Peter Deutsch. *GZIP File Format Specification version 4.3*. Internet RFC 1952, May 1996.

[12] Henson, Valerie Aurora, *An analysis of compare-by-hash*. USENIX Hot Topics in Operating Systems (2003).

[13] Black, J., *Compare-by-Hash: A Reasoned Analysis*. 2006 USENIX Annual Technical Conference.

[14] Mark Ewing, Erik Troan, *The RPM Packaging System*, First Conference on Freely Redistributable Software (1996).

[15] http://www.rpath.com/corp/products/rbuilder

[16] http://www.vmware.com/interfaces/vmdk.html

[17] http://www.microsoft.com/windowsserversystem/virtualserver/techinfo/vhdspec.mspx

[18] Red Hat, Inc. *Red Hat Enterprise Linux Installation Guide* http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Installation_Guide-en-US/ch-kickstart2.html

[19] http://reductivelabs.com/trac/puppet/

[20] http://reductivelabs.com/trac/puppet/wiki/VersionControlPuppet

[21] Jacky Estublier, David Leblang, Andr van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, Darcy Wiborg-Weber, *Impact of Software Engineering Research on the Practice of Software Configuration Management*, ACM Transactions on Software Engineering and Methodology Volume 14, Issue 4 (October 2005).

[22] Darrell Reimer, Arun Thomas, Glenn Ammons, Todd Mummert, Bowen Alpern, Vasanth Bala, emphOpening black boxes: using semantic information to combat virtual machine image sprawl, Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (2008).

[23] http://studio.suse.com/

[24] http://rightscale.com/

[25] http://elasticserver.com/

[26] http://moka5.com/