JUSTIN SAMUEL, JEREMY PLICHTA,
AND JUSTIN CAPPOS

centralized package management using Stork

Justin Samuel is an undergraduate student at the University of Arizona with an interest in security. In addition to research, he teaches a secure Web application development course.

jsamuel@cs.arizona.edu

Jeremy Plichta is a senior majoring in computer science at the University of Arizona. He plans on pursuing a career as a software engineer after graduating.

jplichta@cs.arizona.edu

Justin Cappos is a Ph.D. student at the University of Arizona. His research interests revolve around building large distributed systems.

justin@cs.arizona.edu

MANAGING THE SOFTWARE INSTALLED on multiple systems can be one of the duller aspects of system administration. One has to deal with varied sets of packages, each replicated on numerous machines, and bring up new systems, with the complication of those that are almost like the others, but not quite. In many cases, great amounts of time could be saved and more than a few mistakes avoided by using tools specifically created to make this job easier.

A Better Way

Picture an admin sitting down to upgrade a certain package on 40 boxes, install new software on 100 others, and remove an unused package from every box on the network. Rather than undergo hours of tedium, the admin starts up her Stork management tool and five minutes later she's done. She knows that her systems are hard at work applying the changes she's specified. What about the systems that are currently offline? No problem. They'll apply the changes the next time they come online. And newly purchased systems? They'll be automatically updated to have the correct configuration.

Stork is the name for a collection of package management tools. Among the features it provides are:

- Central management of packages that should be installed on a distributed set of computers.
- The ability to organize systems into logical groups for ease of management.
- Increased speed and reliability of file transfers by utilizing a variety of efficient and redundant protocols.
- Secure architecture utilizing public-key cryptography for digital signatures.
- Low-upkeep repository not requiring a central repository administrator.
- Space, network bandwidth, and CPU savings in some virtual machine environments.
- Fast and efficient update awareness through the use of a publish/subscribe system for update notification.

In this article we'll focus on how to use the Stork tools for centralized management.

How Stork Works

There are three fundamental components of the Stork architecture:

- Client tools: These are package dependency resolution tools similar to yum and apt. In addition to dependency resolution, they also follow instructions from signed configuration files.
- Management tools: The management tools are the utilities administrators use to create the signed configuration files that act as the instructions to the client tools. The management tools are generally used by an administrator on his or her own computer.
- Repository: A repository is a Web server where configuration files created by the management tools are stored so that the client tools can access them.

Administrators can use one of the management tools to specify which packages should be installed on a node. The management tools create the necessary configuration files, digitally sign them with the user's private key, and can even upload them to a repository.

The client tools, running on each machine, determine which packages to install, upgrade, or remove based upon the contents of the configuration files the administrator created. If the actions require downloading package files from the repository, the client tools also verify that the package files to be installed are themselves trusted according to the configuration files.

Currently, the client and management tools run on UNIX-like operating systems, with active usage including Fedora, Gentoo, and Arch Linux. The package formats supported at this time are RPMs and tarballs, with deb file support in development. The client tools can wrap around any existing package management utilities that are installed on a system, so support can be added for any other package formats a user may desire.

Signatures and Trust

By having the client tools ensure that every configuration file downloaded from the repository is signed by a trusted administrator, the client tools can be certain that no configuration files have been tampered with. This allows unrelated organizations to safely share the use of a single repository: The client tools do not trust files because they are signed by a repository key, but rather because they are signed by an actual administrator's key.

As verifying a digital signature requires knowing in advance the public keys whose signatures should be trusted, this begs the question: How do the client tools know which public keys those are? There are two main approaches to making sure the client tools know which administrator keys to trust. These are not mutually exclusive.

The first approach is for an administrator to directly configure the client tools to trust specific keys. For example, trusted keys can be configured at the time the client tools are installed on each system. After that initial setup, new keys can be securely distributed via packages installed using Stork, for example.

The other approach that can be used is to integrate the client tools with an existing method an organization has to distribute or to answer queries for such keys. For example, PlanetLab [1] makes available the ability for a system to fetch the keys that belong to that system's administrators. Since a module in the client tools makes use of this secure PlanetLab API that provides access to these keys, the client tools can be used on PlanetLab without the need to manually distribute trusted keys to each system.

The client tools can be used in environments with multiple administrators. To accommodate this, the client tools will ensure that, for any given config-

uration file, they are using the most recent version of the file in the repository that has a valid signature of a trusted administrator. Thus, when another administrator uploads a new version of the file in which the groups are defined, the client tools will use the newer one.

Organizing Systems into Groups

An important optimization for managing systems is a group. A group is simply a logical association of multiple systems defined by the administrator of those systems. The fact that groups are purely a logical organization is very important: The systems in a group do not need to be connected in any way and a single system can be a member of multiple groups. Groups exist for the sole purpose of simplifying the job of the administrator, who may want multiple systems to have some of the same packages installed.

Stork has three types of groups: simple, composite, and query result.

Simple groups allow users to manage a collection of systems as if they were a single system. One can declare that machines 1, 2, and 3 are part of group G and can then simply say that package X should be installed on group G. This would result in each of machines 1, 2, and 3 installing package X. There is no limit to how many systems can be in a group.

Composite groups are created by performing an operation on existing groups. The supported operations for combined groups are UNION, INTERSECT, COMPLEMENT, and DIFFERENCE. Composite groups do not have systems directly added. Instead, the systems in the group are chosen by the operation and the membership of the other groups. For example, combined group H may be defined as the UNION of group I and group J and will contain all of the systems appearing in either group I or group J. Groups I and J can be any types of groups, including composite groups.

Query result groups are computed based on some property of the network. PlanetLab is a major user base of Stork and therefore the management tools provide support for building groups from the result of CoMon [2] queries. For example, a group to refer to all nodes with more than 1 GB of free disk space could be created using the CoMon query `select='gbfree > 1'`. Queries are evaluated at the time of group creation. Administrators who wish to have query result groups that automatically update can reevaluate the query result groups periodically via a cron job.

Management Tools

There are two main management tools for administering systems that are running the client tools: a graphical tool (a.k.a. the GUI) and a command-line tool called storkutil. These tools make it easy for administrators to create the signed configuration files that need to be uploaded to the repository. Both tools are cross-platform and require that python and openssl be installed. Let's first take a look at using the GUI to manage systems.

The information that the GUI asks for when it is first started is a username and password that will be used to authenticate with the repository, as well as the location of a private key (with optional password). The private key is used to sign configuration files. The corresponding public key is what the client tools will use to verify that the signatures on downloaded configuration files are legitimate.

After the GUI verifies the repository login information and validity of the private key, the user can begin configuring the groups and desired package

installation actions. The GUI retrieves the latest configuration files from the repository and displays any groups and package installation actions that were previously defined. To add a new group, click on the "add group" button and give the group a name. Next, proceed to either add nodes to form a simple group, make this a combined group by defining it as a union or intersection of other groups, or make it a dynamic group by setting a network property such as a CoMon query.

After the new group is created, it important to define package management actions for the group. That is, define packages that should be installed, upgraded, or removed for all of the nodes of the group. To install or upgrade a package, either specify the name of the package (e.g., "gnupg") and let the client tools, when they run on the systems, attempt to find a gnupg package file that is trusted or provide a gnupg package file that is stored locally. When providing a package file that is stored locally, the GUI will take care of adding trust of this specific package file to the configuration files in addition to uploading the package file to the repository so that it is available.

If any changes are made in the GUI, an icon shows that the local state is out of sync with the repository. One can then sync with the repository, which means that the new configuration files and any newly added package files will be uploaded to the repository.

The GUI is the most convenient tool for day-to-day management of systems using the client tools. However, some situations require command-line tools that can perform these tasks. All of the same functionality for configuration file modification and generation that is done by the GUI can be done using the command-line tool storkutil.py.

Here's an example using storkutil.py to add a new group with the name EXAMPLE_NODES with one system, nodeA.example.com, in the group:

```
storkutil.py pacgroups include EXAMPLE_NODES nodeA.example.com
```

Then, to add more systems to the same group:

```
storkutil.py pacgroups include EXAMPLE_NODES nodeB.example.com \
nodeC.example.com nodeD.example.com nodeE.example.com
```

Finally, to say that all of the systems in the new group should install gnupg:

```
storkutil.py pacpackages group EXAMPLE_NODES install gnupg
```

The command-line tool storkutil.py will have taken care of generating the configuration files as well as signing them with the private key. The one thing that the GUI does that storkutil.py doesn't is upload the files to the repository for us. Uploading the files to the repository is something that can be scripted, if desired (for example, using curl).

After the configuration files are generated and uploaded to the repository using either the GUI or the command-line tools, the client tools running on any of the nodes will retrieve these newest files and take any actions necessary based on them.

Where Stork Can Help

Stork's design makes it quite flexible in terms of the roles it can perform for an administrator. However, its primary focus, and that for which it is optimized, is centralized management of many systems.

It is common for administrators to have a base configuration for all of their systems. Various systems may then have specialized software needs depending upon the additional job requirements of the people using those systems

or the services they provide. For example, everybody in the finance department may need a database program that others in the organization do not. Further, all of the secretaries and executives may require a specific calendar management program. Stork makes it easy to manage the software installed for different groups of users. You can even allow a system to be in multiple groups, thus allowing the finance department's secretary to have both the database and the calendar management software installed.

Stork's system of secure file transfer, fast and efficient propagation of changes, and ability to use tarballs as packages make it very useful not only as a package management system but also as a configuration management system. For example, suppose an administrator wants to install a set of custom scripts that cron will run periodically on all of his systems to monitor disk usage. The administrator can package these scripts as a tarball and use the management tools to install the tarball on all machines. The fact that Stork manages tarballs in a similar way to RPM packages gives the administrator considerable control and flexibility without the overhead that can be involved in using traditional package formats to meet these needs. The tarball packages can be easily checked for installation, removed cleanly, or even used to satisfy dependencies.

Incorporating package file security decisions made by other users is simple with Stork's user trust system. This trust system allows an administrator to include, in real time, another user's repository-published list of untrusted packages into his or her own list. This can be very useful in many organizations with a separate IT department that focuses on security. Although this department may publish lists of packages that should not be installed because of known security problems, traditionally it can be hard for system administrators to keep up with the latest changes to such published lists. However, by being able to automatically include the other department's package trust decisions, no further work is needed to stay current with the list of packages to avoid, which can consist of packages marked as untrustworthy by name, version, or hash. With this setup, the system administrators can know that Stork will automatically prevent these untrustworthy packages from being installed.

Other Solutions

There are other tools available for performing centralized management, but none are intended to work the same way as Stork. These other systems are generally configuration management systems that can be used for some degree of package management.

One method used for configuration management is remote command execution. The tools that provide the ability to remotely execute commands on a set of nodes all utilize multiple secure shell connections that originate from the administrator's own computer. Of these, many are oriented toward specific networks, such as PlanetLab. One such system is pssh [3], which provides parallel versions of the openssh tools, including versions of ssh and scp. Using pssh, an administrator could execute a package management command on all of their active systems simultaneously. Major disadvantages with this approach include having to manually repeat the process for new systems brought online or for systems that were previously down and have been brought back online; limited groups functionality (done by utilizing files that contain lists of hosts), but having no functionality similar to Stork's composite groups; and an administrator having to make potentially hundreds or thousands of shell connections directly from the administrator's own computer in order to install, upgrade, or remove packages.

Other solutions that provide similar functionality, all using this same approach, include:

- Plush [4], which includes a graphical interface (see page 32).
- pShell [5], a command-line-only and PlanetLab-specific tool.
- PlMan [6], which includes a graphical interface and the ability to find PlanetLab nodes by CoMon queries.

Although package management on large numbers of systems would be easier using these tools compared with having to manually access each system to do so, none of them focuses on providing efficient and easy-to-use centralized package management. Using them for this purpose would be far from ideal, as none of them handles common cases such as nodes that are down at the time a command is issued or automation of setting up new nodes that come online.

A more robust way to perform configuration management is to use a powerful configuration management system such as Cfengine [7]. Configuration management systems usually involve a central server that provides configuration information to all nodes, where this configuration information is often in a format or language specific to the system. These systems provide considerable general-purpose system administration functionality, such as ensuring that certain files on all nodes are the same, configuring backups, and instructing nodes to perform basic package management actions such as installing a specific package.

Configuration management systems, however, often lack the specialized functionality for secure, centralized package management that Stork provides. For example, Stork can be used securely even when an administrator does not run his or her own repository but instead uses a shared repository. As no actions are taken by the Stork client tools unless the signatures of configuration files are verified as having been created by an administrator, the security of the nodes using the repository does not depend on the security of the repository itself.

Stork is not intended to be a configuration management system, just as tools such as Cfengine are not intended to provide the package management functionality that Stork does. In general, Stork provides a high degree of flexibility and security for centralized management, with the focus being on package management. The package management functionality of Stork, which includes very easy-to-use and lightweight packaging by means of tarball packages, allows it to function as a configuration management system in many cases.

Conclusion

For administrators responsible for multiple systems, centralized package management such as that provided by Stork can save time as well as prevent frustration and mistakes. An administrator need only define once the package management actions the systems should perform. Using Stork alleviates much of the burden of administering large numbers of systems. For more information or to download Stork, please visit the Stork Web site at http://www.cs.arizona.edu/stork.

REFERENCES

[1] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating System Support for Planetary-Scale Network Services," *Proceedings of NSDI '04* (USENIX Association, 2004).

[2] K. Park and V.S. Pai, "CoMon: A Mostly-Scalable Monitoring System for PlanetLab," *ACM Operating Systems Review*, 40(1) (January 2006): http://www.cs.princeton.edu/nsg/papers/comon_osr_06/comon.pdf.

[3] pssh: http://www.theether.org/pssh/.

[4] J. Albrecht, C. Tuttle, A.C. Snoeren, and A. Vahdat, "PlanetLab Application Management Using Plush," *ACM Operating Systems Review*, 40(1) (January 2006): http://www.cs.williams.edu/~jeannie/papers/plush-osr06.pdf.

[5] "pShell: An Interactive Shell for Managing Planetlab Slices": http://www.cs.mcgill.ca/~anrl/projects/pShell/.

[6] "Planetary Scale Control Plane": http://www.cs.washington.edu/research/networking/cplane/.

[7] M. Burgess, "Cfengine: A Site Configuration Engine," *USENIX Computing Systems*, 8(3), 1995.