Universal Script Wrapper - An Innovative Solution to Manage Endpoints in Large and Heterogeneous Environment

Sai Zeng, Shang Guo, Fred Wu, Constantin Adam, Long Wang
IBM T. J. Watson Research Center

Cashchakanithara Venugopal, Rajeev Puri, Ramesh Palakodeti
IBM Global Technology Services

***Abstract*—Endpoint management is a key function for data center management and cloud management. Today's practice to manage endpoints for the enterprise is labor intensive, tedious and error prone. In this paper, we present Universal Script Wrapper, an innovative solution which provides a unique solution to allow users to manage a group of endpoints as if logged in to one endpoint. It harvests proven scripts and makes them available to automate management tasks on endpoints without modification. To reduce risk, it provides a mechanism to guard against intrusive commands and scripts that could cause massive damage to the infrastructure.**

Keywords—multiple endpoint management; task automation; data center management, cloud management.

I. INTRODUCTION

Outsourcing of IT service management empowers enterprises to focus on their core business while service providers ensure smooth functioning of enterprises' mission-critical IT systems and infrastructure. From data center to cloud, from desktop to servers, an IT service provider continuously monitors, operates and controls an enterprise's IT assets to reduce costs and risks and increase uptime, striving to meet service level commitments. An IT infrastructure typically consists of a heterogeneous collection of elements such as servers, virtual machines, logical partitions, hypervisors, routers, switches, storage devices, operating systems, middleware, applications, and services. In our context, an endpoint is defined as any piece of hardware or software in a platform that is capable of connecting to a network, obtaining an IP address on a network, transmitting data through a network, or processing code for a network.

The cost of managing of endpoints in a modern distributed and complex IT infrastructure is often a significant portion of an enterprise's Total Cost of Ownership (TCO). In spite of the trend of endpoints to proliferate in number, complexity, and security risk exposure, enterprises and service providers are continuously challenged to control management cost, reduce risk, and meet service level commitments.

Highly efficient endpoint management tools are designed to use fewer and highly skilled people to manage more endpoints. Tool vendors are motivated to create and embed advanced technologies into their products to increase the number of endpoints per management product instance, thereby increasing the endpoint/FTE ratio. In parallel, enterprises outsource the regular management tasks to the personnel with low skill and cheaper labor rate. Deploy such tools for highly skilled administrators, and still can empower low skill personnel to perform their management tasks raises a new challenge for enterprises.

For companies who manage complex and heterogeneous IT infrastructures require diversified management functions at various scales. Often a commercial product may supply needed functions, but might not be able to integrate with business practices, or integration has an unacceptable cost. This is one of the key reasons why one still finds labor intensive activities in the IT management business. For common, labor intensive management tasks, administrators usually build small automation tools or scripts. Those tools or scripts are very cost effective, in general developed and used locally, and manually invoked on a single endpoint. They can save some manual effort, but the productivity gain is limited.

Finally, managing multiple endpoints as if managing a single endpoint is on every administrator's wish list. This may not be so helpful when dealing with issues and problems which are limited to specific endpoints. For regular maintenance tasks such as health checking, patch update, performance monitoring etc, this new approach can easily achieve productivity gains at the scale of the number of endpoints. For instance, to modify a standard set of users for a group of like servers, instead of separately running commands or scripts on each server, an administrator wants to run commands or scripts against multiple servers simultaneously and view the collective results from all endpoints for validation. We have seen many commercial products in the marketplace that manage multiple endpoints through a single portal, such as the VMWare suite [1] for virtualization management, but they are still far from the goal of managing multiple endpoints as if managing one.

In this paper, we present a innovative solution, called Universal Script Wrapper (USW), which is developed based on Tivoli Endpoint Manager (TEM) [2]. TEM provides capabilities to manage the security and compliance of servers, desktops, mobile devices, etc. The critical characteristic of TEM that we leverage is that it provides a framework to allow a user to wrap, distribute, and invoke commands and automation scripts the user has developed onto any endpoints for which the user has authorization. USW enhances TEM functions with three key features. It provides a unique user interface or dashboard which simulates an administrator's experience of direct interaction with an endpoint through a command window. In this user interface, an administrator can issue commands and invoke scripts for a set of endpoints simultaneously, and standard output of command/script invocation is collectively displayed in the user interface. The experience of such a user interface represents the principle we advocated previously, namely "managing multiple endpoints as if managing one". A potential adverse effect of this principle is that it magnifies the impact of damage if an administrator mistakenly issues disruptive commands or scripts which affect an entire set of endpoints. To prevent damage, USW provides a validation feature to check commands against a whitelist, prohibiting any commands which are not explicitly permitted. Commands not in the whitelist include those that are intrusive, such as shutdown and reboot. Finally, USW takes advantage of TEM's capability to distribute a file to endpoints. We developed a framework which can automatically distribute proven scripts from a central location to selected endpoints. An administrator can execute the scripts as if typing at a command line, and execution results will be displayed upon completion on the endpoints. The details of USW are described in sections II, III, and IV. In section V, we illustrate the efficacy of deploying this solution, and related research and future research directions are highlighted in section VI and VII.

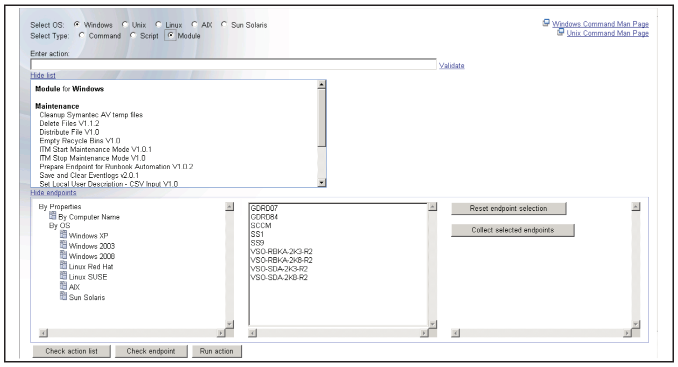
II. ONE USER INTERFACE TO MANAGE MULTIPLE ENDPOINTS

The most common way of managing endpoints is that the system administrator manually logs into the troubled machine, executes commands/scripts, and collects and consolidates outputs for troubleshooting or remediation. When dealing with large numbers of endpoints, this practice is time consuming, tedious, and error-prone.

USW, a dashboard of the solution for managing servers is built on the console of TEM. A TEM agent runs continuously on each client and establishes a secure communication channel with TEM server. The dashboard provides a user interface that accepts native shell commands, native scripts or tool-specific scripts on multiple endpoints in parallel and collects the results from them.

Over the years, design and usability have become increasingly important to business. The IT industry is now realizing that how something looks and feels is as important as the features that it provides. To create a simple but powerful user interface, we keep a strong focus on the end users. An ethnographic field study on system administration tools conducted by IBM Research [3] shows CLIs are the system administration tools preferred over any GUIs by those most familiar with commands because CLIs are faster, more reliable, more robust, and more accurate. However for those who are less experienced, the easy-of-use GUIs seem the only way to support their work. We wanted to develop a tool that benefits both groups of users without sacrificing the effectiveness of either one.

We designed a unified GUI (shown in Fig. 1.) for running commands and scripts across all windows and unix platform managed endpoints. The interface first lets the user select OS platform and action type - whether it is a single-line command, execution of a native script, execution of a module (a module is a collection of TEM action scripts). Once platform and action type are selected, a list of available action items is displayed. These action items were approved and stored in role-specific and account-specific white lists. The white lists are determined by user's roles and the account on which the user is working. The white lists ensure that commands/scripts are safe to run and the user is authorized to run them. The next step is to select the applicable target endpoints on which the actions are to execute. All action items and endpoints that the user can see in the interface are determined by the embedded Privilege Manager and Control List.


Select OS: [✓ Windows] Unix Linux AIX Sun Solaris
Select Type: Command Script [✓ Module]
Enter action:

Module for **Windows**

Maintenance
Cleanup Symantec AV temp files
Delete Files V1.1.2
Distribute File V1.0
Empty Recycle Bins V1.0
ITM Start Maintenance Mode V1.0.1
ITM Stop Maintenance Mode V1.0
Prepare Endpoint for Runbook Automation V1.0.2
Save and Clear Eventlogs v2.0.1
Set Local User Description - CSV Input V1.0

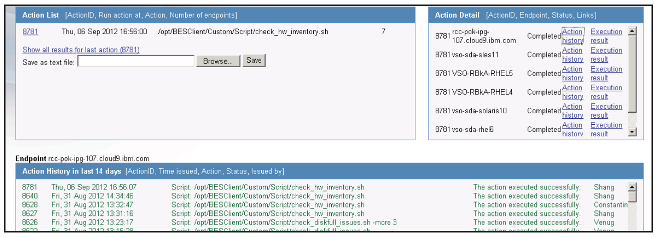
By Properties
By Computer Name
By OS
Windows XP
Windows 2003
Windows 2008
Linux Red Hat
Linux SUSE
AIX
Sun Solaris

GDRD07
GDRD84
SCCM
SS1
SS9
VSO-RBKA-2K3-R2
VSO-RBKA-2K8-R2
VSO-SDA-2K3-R2
VSO-SDA-2K8-R2

Fig. 1. Dashboard User Interface for Command/Script Selection and Endpoint Selection

After the action is issued and sent to multiple selected endpoints via the dashboard, the framework collects the execution results from multiple endpoints and displays them in the same dashboard for the user to view. The interface provides different views of the execution results according to the user's preference, i.e. the execution result from a single endpoint vs. consolidated results from multiple endpoints. In addition to the execution results, the dashboard can show the "Action History" for each endpoint in the last 14 days (shown in Fig. 2.) so the user knows what kind of actions have been performed recently on the endpoint, which can help to diagnose the current situation.

The interface allows the user to concatenate or pipe commands or scripts as they normally do on the command-line. All commands or scripts are validated before they are sent to the endpoints. In order to prevent typing errors when entering commands or scripts in the input field, we provide a copy/paste feature in the GUI so the user can click the command or the script from the list without even touching the keyboard. At the same time, when an intrusive script is selected, a warning message alerts the user.



Action List [ActionID, Run action at, Action, Number of endpoints]

8781	Thu, 06 Sep 2012 16:56:00	/opt/BESClient/Custom/Script/check_hw_inventory.sh	7

Show all results for last action (8781)

Save as text file:

Action Detail [ActionID, Endpoint, Status, Links]

8781	rcc-pok-ipg-107.cloud9.ibm.com	Completed	Action history	Execution result
8781	vso-sda-sles11	Completed	Action history	Execution result
8781	VSO-RBkA-RHEL5	Completed	Action history	Execution result
8781	VSO-RBkA-RHEL4	Completed	Action history	Execution result
8781	vso-sda-solaris10	Completed	Action history	Execution result
8781	vso-sda-rhel6	Completed	Action history	Execution result

Endpoint rcc-pok-ipg-107.cloud9.ibm.com

Action History in last 14 days [ActionID, Time issued, Action, Status, Issued by]

8781	Thu, 06 Sep 2012 16:56:07	Script: /opt/BESClient/Custom/Script/check_hw_inventory.sh	The action executed successfully.	Shang
8640	Fri, 31 Aug 2012 14:34:46	Script: /opt/BESClient/Custom/Script/check_hw_inventory.sh	The action executed successfully.	Shang
8628	Fri, 31 Aug 2012 13:32:47	Script: /opt/BESClient/Custom/Script/check_hw_inventory.sh	The action executed successfully.	Constantin
8627	Fri, 31 Aug 2012 13:31:16	Script: /opt/BESClient/Custom/Script/check_hw_inventory.sh	The action executed successfully.	Shang
8626	Fri, 31 Aug 2012 13:23:17	Script: /opt/BESClient/Custom/Script/check_diskfull_issues.sh -more 3	The action executed successfully.	Venug

Fig. 2. Dashboard User Interface for Action Status and History

III. Harvest proven scripts

In a large IT service management outsourcing organization managing multiple data centers, it is common practice that individual system administrators develop and use their own scripts and tools. They may share some of these scripts that they have found most useful with others in their local department or physical proximity, but still it is likely that across the organization, the same type of work is being done with many different ad hoc tools in an uncontrolled manner. As a result there is little consistency in the quality or format of the outcomes, and there is much duplication of work to develop and maintain these scripts. Furthermore, each administrator or group of administrators may set up its own repository of scripts, each with its own management overhead.

A tool such as the USW based on a distributed management and execution framework has the potential to standardize scripting tools and centralize their management. An important goal of such a system is to minimize the time and effort overhead in the process of taking a locally developed script and "publishing" it as a sanctioned tool. The USW contributes to this goal by eliminating the need to migrate or adapt a native script into a form that is consumable by the framework. For example, the standard way of adding capability to TEM is by creating a new "fixlet" and embedding the script inside the fixlet. A TEM fixlet is essentially a script written in a TEM language, which can accept embedding of OS/shell scripts. However, typically some modifications inside the OS/shell script must be made to ensure that the script will be properly interpreted by TEM and distributed to the endpoints intact. The Universal Script Wrapper avoids this chore by using a single generic TEM fixlet that downloads the unaltered script for execution.

In addition to simplifying the mechanical aspects of publishing new scripts, we are implementing a process for script management. We will set up a simple hierarchy with account, regional, and global levels. A script will initially be used informally by a single administrator, who could later offer it for use across his account(s). If it is found to be widely useful, then the administrator could offer it for use in all accounts in his region. Finally, it could be offered globally. At each promotion step there will be a review of the script's functionality, quality, and range of applicability, plus a comparison with other scripts already available in the proposed scope. In some cases, additional development will be undertaken to merge the script with a similar one.

Our use of the TEM framework for script distribution also reduces network traffic. Each time an administrator initiates a script execution on a set of endpoints, each client checks for existence of that script locally. If the script exists locally, the client agent computes its digital signature and compares it against a value passed in with the execution request. If the signatures do not match or the file is not found, then the client downloads the script through the TEM relay framework. This procedure protects against execution of an altered local copy of the script, yet eliminates the download in the vast majority of cases.

IV. Commands/scripts Validation

The validation feature, or validator, verifies that the user input of a command line complies with a set of policies. Policies are defined for each version of each operating system running in the data center. They specify which commands and or scripts are allowed for particular user groups, and also which options and parameters are allowed for each command or script. A command line is distributed to the endpoints for execution only after it passes the validation check.

The role of the policies enforced by the validator is to reduce the risk of a non-expert user running incorrectly an intrusive action (e.g. restrict the options for '*/sbin/fdisk*' to '*–l*' and '*–u*' in order to only view, not change the disk partition tables), or launching an action that consumes significant resources on a large number of nodes (e.g. exclude the option '*–c*' for '*/bin/netstat*', to avoid repeatedly printing network statistics for each node every second).

A command/script can have one of the following parameter policies: no parameter restrictions; run only with a subset of available parameters; don't run with certain parameters; run only without parameters; or run with any parameters, but only if a specific, required option is invoked.

Validation can be performed as well for composite commands, such as '*dsquery user ou=Marketing, dc=Microsoft, dc=com | dsmod group "cn=Marketing Staff, ou=Marketing, dc=microsoft, dc=com" -addmbr & shutdown*'. If part of a command fails the validation (e.g. *shutdown* is not in the white list), the composite command will be rejected and none of it will execute on the endpoints.

As there are sophisticated rules and requirements for different commands, we apply a language grammar to precisely and flexibly define the command lines that are acceptable by the validator. Part of the Backus-Normal Form (BNF) notations [11] of the language grammar are presented in the table below (for Linux/Unix; similar specifications are defined for Windows) to parse and validate single or composite commands. Each command line input is tokenized into a series of commands; and each command and its set of parameters are checked against the rules described in the white list.

TABLE I. BNF GRAMMAR FOR USER-ENTERED COMMAND LINE IN LINUX

```
{}* zero or more repetitions; {}+ one or more repetitions; | or

cmdline::=cmdstmt {{separator}+ cmdstmt}*
cmdstmt::=cmdnores|cmdnoarg|cmdres
separator::=\||\|\||&|&&|;|)|(
cmdnores::=cmd {{option}* {userinput}*}*
cmdnoarg::=cmd
cmdres::=cmdonly|cmdexcept|cmdrequire
cmdonly::=cmd {onlyoption {userinput}*}+
cmdexcept::=cmd {exceptoption {userinput}*}+
cmdrequire::=cmd {nonrequireoption {userinput}*}*
   {requireoption {userinput}* {nonrequireoption {userinput}*}*}+
cmd::=text|.cmd1|/cmd1|cmdprefix cmd
cmdprefix::=/bin/sh|/bin/shell|/usr/bin/perl|...
cmd1::=text|.cmd1|/cmd1
onlyoption::=option
exceptoption::=option
nonrequireoption::=option
requireoption::=option
option::=-text|--text
userinput::=text|/userinput|.userinput
text::={letter|digit|_|"}text1
text1::={letter|digit|_|.|-|/|"|=|:}*
letter::=a|b|...|z|A|...|Z
digit::=0|1|...|9
```

V. SUMMARY

In this paper, we have presented Universal Script Wrapper, - an innovative solution to allow system administrators to manage multiple endpoints as if managing a single endpoint; this is the key to the productivity gain. Besides supporting execution of standard operating system commands, we also enable importing and distributing scripts onto the endpoints, and invocation of the scripts via a command line. To mitigate the potential risk of damage to mutiple endpoints, we developed a validator to prevent the invocation of dangerous commands on the endpoints. At present, the USW has been deployed in a large number of accounts, and the feedback from the system administrator community is very positive. We are in the process of collecting data to measure productivity gains, and we will report on the efficacy of solution in a future publication.

VI. RELATED WORK

Several open source projects are addressing the problems of configuration management, multi-node deployment, and ad-hoc task execution. CFEngine [4], Puppet [5], and Chef [6] are the most commonly used tools, deployed on all widely available platforms. Recent interesting developments include Salt [7], which adds live monitoring capabilities, and Ansible [8], which emphasizes simplicity and ease of deployment.

A methodology for configuration maintenance based on the promise theory was initially developed by Mark Burgess [9], and CFEngine is an implementation of these concepts. Promises are fundamental components that provide a declarative interface to the managed resources, and act as convergent, idempotent operators that move these resources toward the intended state specified by the user.

Puppet and Chef, at their core, work like CFEngine. One way in which Puppet is different is that it uses a centralized mechanism to deliver policies, as opposed to CFEngine and Chef, where policies are copied and evaluated on the edges. One of Chef's main differentiating points (and best features) is that, it allows separating a node's name from its functionality, while in CFEngine and Puppet classes are matched based on machine identifiers (hostnames, FQDN, IP addresses or equivalent).

Salt is a remote server management and configuration tool that uses ZeroMQ [10] to exchange information between servers (masters) and clients (minions), as opposed to the REST API used by CFEngine, Puppet, and Chef. ZeroMQ is an embeddable networking library that has an asynchronous I/O model and allows connecting sockets N-to-N with patterns like fan-out, pub-sub, task distribution, or request-reply. By adding secure communication capabilities to ZeroMQ, Salt is capable to capitalize on ZeroMQ's high performance and retrieve live data about the system state. Ansible is a solution that puts the emphasis on simplicity. It uses SSH as the default transport, and does not require any additional remote software to be installed on the managed machines. Both Salt and Ansible are implemented in Python; both Salt and Ansible modules work over JSON and can be written in any language.

VII. FUTURE RESEARCH

Many business and technology challenges motivate us to do further research. One research direction we envision is to improve the interactivity of our solution. At present, we support only commands which do not require any additional user input after command issuance due to the fact that TEM action executions are asynchronous, and the long response time is not acceptable for user interaction. In the future, we want to extend our solution to support interactive behavior of standard commands. Another research direction is to build a generic validator that supports commands that do not have a predefined grammar. Instead of modifying validator code to support new grammars, the goal is to modify only the grammar policies.

REFERENCES

[1] VMWare Suite, http://www.vmware.com

[2] Tivoli Endpoint Manager, http://www-01.ibm.com/software /tivoli/solutions/endpoint/?s_pkg=bfmw

[3] E. Haber and J. Bailey, "Design Guidelines for System Administration Tools Developed through Ethnographic Field Studies," Proceedings of the 2007 symposium on Computer human interaction for the management of information technology, pp. 1

[4] CFEngine - Distributed Configuration Management. http://cfengine.com/

[5] Puppet Labs: IT Automation Software for System Administrators. http://puppetlabs.com/

[6] Chef | Opscode. http://www.opscode.com/chef/

[7] Salt Stack. http://saltstack.org/

[8] Ansible - SSH-Based Configuration Management & Deployment. http://ansible.github.com/

[9] M. Burgess, Cfengine: a site configuration engine, USENIX Computing systems, Vol. 8, No. 3 1995

[10] The Intelligent Transport Layer – zeromq. http://zeromq.org/

[11] D. Grune, C. J.H. Jacobs, Parsing Techniques: A Practical Guide. Springer, 2nd edition, 2007