

2013

SMiT: Local System Administration Across Disparate Environments Utilizing the Cloud

Kevin Guyot

Grand Valley State University

Follow this and additional works at: <http://scholarworks.gvsu.edu/cistechlib>

Recommended Citation

Guyot, Kevin, "SMiT: Local System Administration Across Disparate Environments Utilizing the Cloud" (2013). *Technical Library*. Paper 148.

<http://scholarworks.gvsu.edu/cistechlib/148>

This Project is brought to you for free and open access by the School of Computing and Information Systems at ScholarWorks@GVSU. It has been accepted for inclusion in Technical Library by an authorized administrator of ScholarWorks@GVSU. For more information, please contact scholarworks@gvsu.edu.

SMiT: Local System Administration across Disparate Environments Utilizing the Cloud

By
Kevin L. Guyot
April, 2013

SMiT: Local System Administration across Disparate Environments Utilizing the Cloud

By
Kevin L. Guyot

A project submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University
April, 2013

Table of Contents

Abstract	4
Introduction.....	4
Script Management Tool (SMiT).....	5
Related Work.....	5
ManageEngine	6
CruiseControl	6
AutoMate	7
SMiT Architecture	7
Web Client	8
Engines.....	9
Plugins	9
Implementation	10
Architecture: SaaS	10
Web Services: REST	11
Platform: Google App Engine (GAE).....	11
Web Client	11
Web Services.....	12
Engine.....	13
System Evaluation.....	13
Conclusion	14
Bibliography.....	16
Appendices	18
Appendix A: Installing the SMiTEngine	18
Appendix B: Creating a Web Client Plugin UI.....	19
Appendix C: Creating an Engine Plugin	20

Abstract

System administration can be tedious. Most IT departments maintain several (if not several hundred) computers, each of which requires periodic housecleaning: updating of software, clearing of log files, removing old cache files, etc. Compounding the problem is the computing environment itself. Because of the distributed nature of these computers, system administration time is often consumed in repetitive tasks that should be automated. Although current system administration tools exist, they are often centralized, unscalable, unintuitive, or inflexible. To meet the needs of system administrators and IT professionals, we developed the Script Management Tool (SMiT). SMiT is a web-based tool that permits administration of distributed computers from virtually anywhere via a common web browser. SMiT consists of a cloud-based server running on Google App Engine enabling users to intuitively create, manage, and deploy administration scripts. To support local execution of scripts, SMiT provides an execution engine that runs on the organization's local machines and communicates with the server to fetch scripts, execute them, and deliver results back to the server. Because of its distributed asynchronous architecture SMiT is scalable to thousands of machines. SMiT is also extensible to a wide variety of system administration tasks via its plugin architecture.

Introduction

System administration can be tedious. Most IT departments maintain several (if not several hundred) computers, each of which requires periodic housecleaning: updating of software, clearing of log files, removing old cache files, etc. IT professionals are constantly looking to minimize the amount of time spent doing trivial tasks [Dubie, 2009], and because humans are most commonly the cause of error when executing repetitive tasks, it makes sense to use tools to automate the execution of these repetitive actions. These tools often take the form of a set of automated scripts [NetworkAutomation, 2010]. Use of these tools reduces costs, reduces human-induced errors, frees employees to focus on strategic activities, and improves security and control of IT processes [NetworkAutomation, 2010].

Compounding the problem is the computing environment itself. Initially, the number of machines may be small and it is not difficult for an administrator to deploy a set of scripts to all of them. However, over time, as the number of machines increases, the deployment of scripts can become problematic as each machine needs to be visited and updated [Kaseya, 2010]. Also, if there is a bug in a script the updated script will need to be redeployed, further consuming the administrator's valuable time. Furthermore, IT professionals often visit each machine's log to view execution history and script metrics, which is linear in proportion to the number of machines.

Although solutions for distributed system administration have existed for several years, they often lack in one of the following areas: scalability, intuitive user interface (UI) and flexibility.

Scalability is limited due to the system designs forcing isolated or on-premise implementations. The user interfaces of systems in the market typically favor the consumption of gathered or “performed” information rather than the setup and execution of information-gathering or maintenance actions. Finally, tools are often limited to a specific environment or discipline, and cannot span multiple environments or handle the monitoring, building and deploying of applications.

Script Management Tool (SMiT)

To meet the needs of system administrators and IT professionals, we have developed the Script Management Tool (SMiT). SMiT provides a scalable, intuitive and flexible framework to automate infrastructure and application monitoring, system maintenance, and continuous integration. SMiT is an enterprise class, unified, web-based scripting tool that allows an IT professional, without development expertise, to solve everyday enterprise needs. In other words, SMiT, is a tool to allow an IT professional to easily create and execute reusable scripts across a disparate set of machines. SMiT provides an intuitive user interface to easily aggregate autonomous actions into scripts such as copying a file, moving a directory, check disk utilization, cloning a Bitbucket repository, or compiling a java application. These scripts can be aggregated into a project to allow a user to execute the actions manually or on a repeating interval. Script executions can occur anywhere within an organization and within various environments, solving both the scalability and flexibility concerns.

SMiT can solve everyday enterprise needs such as:

- Continuous building and compiling of applications from source control.
- Deployment of an application to disparate machines across an enterprise.
- Monitoring an application by checking it is running, and if it is not, sending an email.
- Monitoring a machine by checking how much disk space is available and sending an email if beyond a tolerance limits.

Related Work

Existing applications in this space typically fit into one of three spaces: infrastructure and application monitoring, continuous integration and system maintenance. We evaluate three applications active in the market today, ManageEngine: an infrastructure and application monitoring tool, CruiseControl: a continuous integration tool, and AutoMate: a system maintenance tool. There are many other tools in the market in the infrastructure and application monitoring, continuous integration and system maintenance spaces as shown in Figure 1, but these tools were chosen as they are illustrative of other tools in their category.



Figure 1 (Related Work)

ManageEngine

ManageEngine, [Manage Engine], is designed to manage mission critical business applications. It is designed around monitoring infrastructures that are required to enable business processes within an enterprise. ManageEngine is very good at monitoring networks, servers, applications and databases remotely with agentless monitors. This agentless monitoring allows ManageEngine to scale quickly across enterprises.

ManageEngine provides visually stunning UIs, however the UI can get overwhelming and confuse users. ManageEngine can confuse users with the number of available dashboards and drilldown capabilities, requiring extensive user training to be used effectively. Not only will training be required to navigate and utilize the UI, training would also be required to understand installation, setup and configuration of the system. A full time staff dedicated to supporting the system will most likely be needed [ENTERPRISE MANAGEMENT ASSOCIATES]. The support staff would be needed to insure the system is operational as well as work with other teams within the enterprise to configure ManageEngine for each specific business process trying to be monitored.

CruiseControl

CruiseControl, [CruiseControl], is an open source continuous integration tool, available on SourceForge, which has a versatile extensible framework that can be used to create a custom continuous build process. CruiseControl offers a vast built-in library of plugins to support a variety of source controls, build technologies, and notifications schemes. It also provides a framework for developers to create and contribute new plugins either for personal use or for the benefit of the CruiseControl community. Each plugin submitted to the community is reviewed by the Administrators and only those accepted are included as part of the CruiseControl project. The CruiseControl web interface provides details of the current and previous builds allowing a user to visualize project status.

One of CruiseControl's weaknesses is scalability, as CruiseControl was originally architected to execute on a single machine. Since CruiseControl's original release an external package has been released to allow distributed builds. This package takes considerable effort to get configured and working and requires users to be familiar with CruiseControl if they expect to

succeed with this more complex arrangement (CruiseControl - Distributed Extensions). CruiseControl has plans to combine this package into the core application but no time frame has been provided (CruiseControl - Distributed Extensions).

Another weakness of CruiseControl is the lack of an intuitive UI for build configuration. While the UI is very good at communicating the status of builds to its users, it is cumbersome to create or modify builds. All build configurations are stored in a complex XML configuration file. This doesn't allow for a user to intuitively change a build script from the UI or to easily access a build script from a disparate location.

AutoMate

AutoMate, [AutoMate], from NetworkAutomation is marketed as a tool that can automate server and desktop tasks as well as system maintenance. AutoMate is based on the concept of scripting tasks without writing code. Script writing is accomplished via a powerful tool called Task Builder. Task Builder provides an easy-to-use, drag-and-drop, intuitive user interface for creating a set of scripts that can be automated via a predefined set of triggers. Triggers watch for specific events or conditions to occur, and then executes the scripts associated to the trigger. Triggers can monitor various events within a system including but not limited to: event logs, performance metrics, SNMP traps, processes or system startup.

AutoMate's major weakness is flexibility, as AutoMate only provides support for Windows based operating systems. It is no longer sufficient to fixate on one platform, as hardware devices are quickly expanding and running varying OSes including: Windows, UNIX, Linux and OS X [Asay, 2010]. It stands to reason as an organization is choosing tools to use within their enterprise; they will seek tools that support the major environments within their organization. As AutoMate has chosen to not directly supporting the major server OSes, organizations may turn to other vendors.

SMiT Architecture

SMiT is comprised of 3 components: a web client, a set of execution engines, and a set of plugins (Figure 2). The SMiT web client allows the user to create, organize, queue and track scripts within a web browser. Since the client is web-based it provides users the ability to access and share scripts with other users across an enterprise. The web client also allows scripts to be queued for execution on an engine, and the execution results to be reviewed.

A SMiT execution engine is resident on every device that must be managed. It is responsible for executing the queued scripts locally on the host machine, and storing an activity log of the script execution for later review by the web client. The execution logs allow a user to centrally monitor a script's performance or impact to an enterprise. SMiT also identifies which scripts failed execution, which scripts are waiting to execute, and which scripts are still executing.

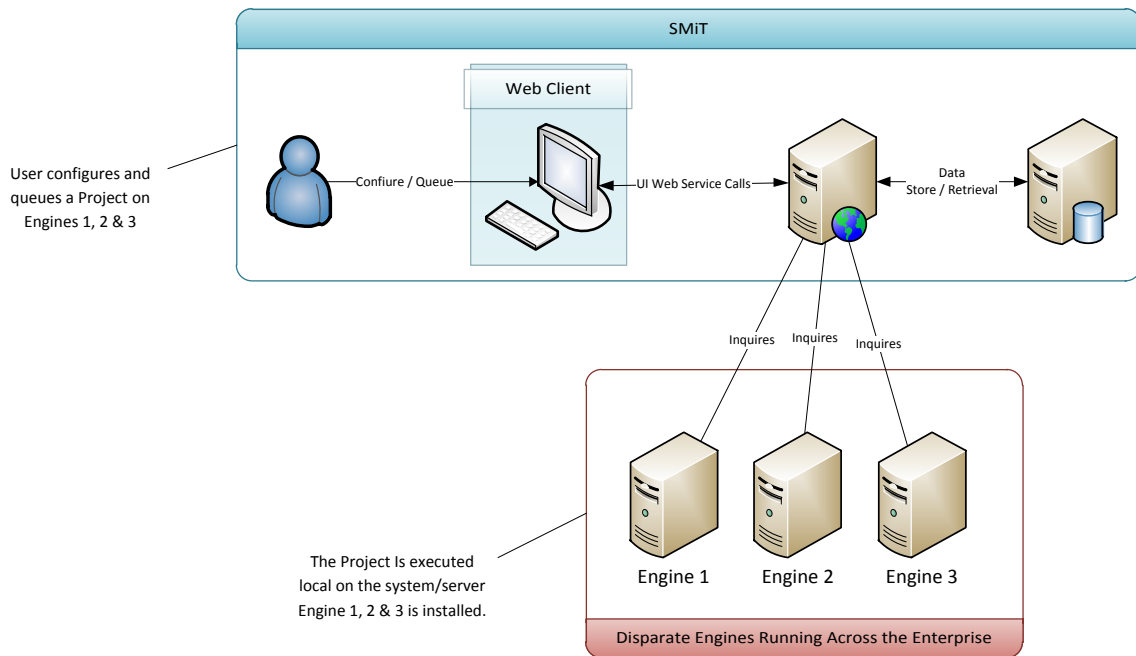


Figure 2 (Overall Architecture)

Finally, SMiT provides an extensible framework to create custom actions, called plugins. Plugins extend SMiT's already available capabilities, enabling the tool to meet an organization's growing needs. Ultimately SMiT should save an individual time in their daily activities, allowing them to focus their time on discovering competitive advantages for their organization.

Web Client

The web client enables users to create, manage, and organize scripts (Figure 3). Scripts are composed of actions by simply dragging and dropping actions from a toolbox to the script window. Scripts can be grouped into a project to allow for either synchronous or asynchronous execution of all the associated actions.

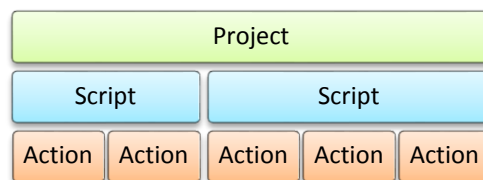


Figure 3 (Script Management)

The web client also allows users to assign execution of a project to a machine by queuing a project on one or more engine(s). The engine(s) execute the project (see next section) and

return the results to the web client. From the web client users are able to see currently executing scripts, previous executions, and detailed logs of the executions.

Engines

There are two aspects of the SMiT engine. The first one is the registration and assignment of an engine within the web client. The second aspect of the engine is the physical instance and the execution performed. To get started, a user needs to register an engine within SMiT web client. When an engine is registered an access token must be provided. The access token is used to authenticate a running instance with SMiT. Once an engine is registered within SMiT it can be assigned to multiple projects, allowing a user to run multiple projects on the same engine.

Physical instances of engines are deployed on disparate systems and are responsible for script execution. Engines inquire for projects to execute from a central data store, utilizing their access token. If the correct access token is provided by the engine, projects that need to be executed are returned. Once an engine has projects to execute, the engine retrieves the scripts for that project from the central data store and executes the scripts locally. Upon completion of each action within a script the engine stores the results in the central data store, including the start and end times of each action executed.

Plugins

The SMiT web client and engine are both extensible utilizing a plugin architecture. The web client allows for the dynamic creation of the user interface for new actions called plugins. This plugin architecture allows the web client to be extended with new autonomous actions without modifying the SMiT web client. Currently only SMiT administrators can add plugins. SMiT administrators are owners of the SMiT platform, having the ability to publish new versions SMiT.

Plugins consist of two parts: a user interface, and the code for execution. A plugin's user interface is input to the web client. The interface is defined in HTML, utilizing a common and well known markup to acquire input parameters from the user that the plugin needs to execute.

Edit Plugin:

Group: File System

Name: Copy File

UI: `<div>
<label>Source:</label> <input name='src'
id='copyfilesrc'>
</div>`

Cancel Edit Plugin

Figure 4 (Plugin UI Editor)

The source code for each is written in Java, implements the interface defined in the SMiT engine API, and stored in an engine's "plugin" folder. When an engine starts it loads and validates the available plugins installed in the plugins folder that have implemented the interface. When the

engine recognizes it has scripts to execute, each action in that script is resolved to a plugin. An instance of the plugin is created and the action is executed.

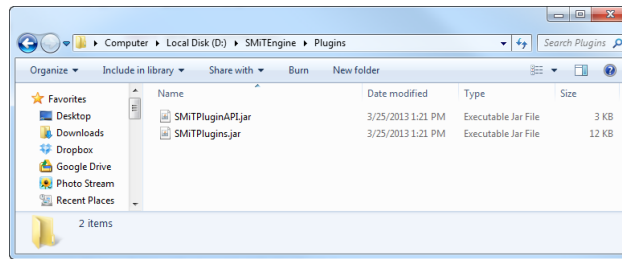


Figure 5 (Engine Plugin Folder)

For example, consider a plugin designed to copy a file. The user interface would prompt for the source and destination files for the copy. The source code within the plugin would accept the source and destination parameters from the engine, and implement the necessary code to perform the file copy operation. When a script called for the file copy operation, the plugin parameters would be sent from the web client to the engine, which would then invoke the plugin code.

Implementation

The major tools in the market are currently designed to be installed on premise within a datacenter of an enterprise [Cerny, 2013]. AutoMate's CEO, Dustin Snell, recently commented on why AutoMate is not currently in the cloud in a ZDNet article stating: "... integration and automation, it's going to be much more often on premise." However, with major corporations actively seeking cloud based Software as a Service, SaaS, Solutions this direction seems short sighted [Cerny, 2013]. To meet the demand of enterprises shifting to cloud solutions, SMiT's Web Client and Engine are architected as a SaaS and utilize RESTful web services, which have been developed on top of Google App Engine, GAE, platform.

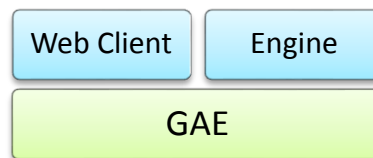


Figure 6 (SMiT Stack)

Architecture: SaaS

The SaaS architecture was chosen for SMiT as it provides convenience, lower costs, less risk and faster deployments for an enterprise [Kenney, 2008]. Convenience is provided to enterprises with access from wherever there is internet connectivity and scalability is provided to an enterprise as they only need to worry about what they want to monitor and not the hardware or resources to support the platform.

Deployments of traditional on premise applications can take years, consume massive resources and yield unsatisfactory results [Zucco, 2009]. SaaS applications typically have lower costs and faster deployments which can be attributed to several factors [Finch, 2006]. First SaaS has a lower cost because an enterprise does not need a team of IT administrators to maintain and support a system. Secondly with SaaS architecture, enterprises do not need to worry about server hardware to support the additional enterprise devices or users, as this is the responsibility of the SaaS provider. Finally SaaS removes the enterprise from the burden of upgrading their software to current supported versions, reducing the cost of resources needed to perform upgrades.

Enterprises risks are reduced when utilizing a SaaS architecture as costs are not paid up front and industry standards for security and data archival are expected and provided [Finch, 2006]. SaaS supports the pay as you go method, where an organization only pays for what an enterprise needs and uses, reducing initial capital expenses.

Web Services: REST

The data feeds to the web UI and Engine for SMiT were design around the Representational State Transfer (REST) architectural style. The REST architectural style helps to separate the user interface from the data storage. Separating the user interface from the data storage improves the portability and scalability of an application [Fielding, 2000]. REST is not a toolkit or a standard but an architecture style that is based on the HTTP protocol, which provides a stateless, cacheable, uniform interface to a set of interconnected named resources [Fielding, 2000], [Rodriguez, 2008]. A set of web services exposed over the Internet implementing the REST architectural style is commonly referred as RESTful Web Services [Rodriguez, 2008].

Platform: Google App Engine (GAE)

SMiT Web user interface and RESTful API was built on top of Google App Engine (GAE), a cloud based platform as a service (PaaS) development environment hosting web applications. The web UI and RESTful web service backend, built on GAE, was chosen for its cost, scalability and robust architecture. GAE's costing model is based on the resources an application consumes; charging for the bandwidth and data storage consumption. GAE allows hosted web applications to run on top of the same systems that Google utilize for their own applications, preventing the maintenance of servers and datacenters. Resources typically spent on system maintenance can now be redirected, allowing enterprises to focus on application features and customers.

Web Client

In addition to HTML and JavaScript, The Web Client, also utilizes other frameworks including: JQuery, Bootstrap from Twitter and DataTables. The JavaScript JQuery library was chosen to simplify HTML document traversal and manipulation, event handling, animation and Ajax requests across multiple browsers. The Bootstrap user interface framework provided a consistent look and feel to SMiT as well as providing a responsive user interface to support both

desktop and mobile devices. Finally DataTables, a JQuery plugin, was leveraged for visualizing data within tables providing search, pagination & sorting capabilities.

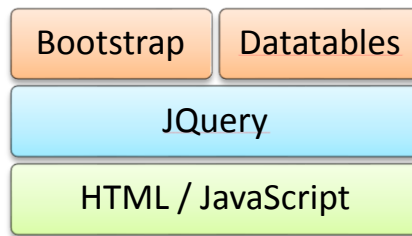


Figure 7 (Web Client Stack)

Users accessing the Web Client are authenticated via the Google authentication services. Utilizing Google authentication services requires users to have a Google account. Once a user is authenticated via Google, SMiT is provided with the email address of the authenticated user. The web client uses the users email address to track what objects within SMiT are created by which users.

Web Services

The RESTful web service backend was written in Java unitizing the Restlet Framework, as the framework is well established and proven by organizations including: Google, LinkedIn, Nasa and Ericsson which are utilizing the framework. SMiT's data is stored in Google's App Engine High Replication Datastore (HRD). Utilizing the HRD allows SMiT to not worry about the distribution, replication, and load balancing of data behind the RESTful web services as well as it allows the application to scale quickly. The HRD is accessed via DataNucleus's Java Data Objects (JDO), which is an abstracted persistence layer, allowing NoSQL data access. The JDO returns the data requested by the client web service call from the HRD. Before the data is returned to the client from the web service call it is converted to JavaScript Object Notation (JSON) so the web client can easily consume the data.

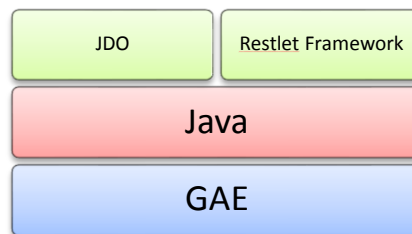


Figure 8 (RESTful Web Service Stack)

The security of SMiT is at the web service layer, making every request to a web service be authenticated via Google authentication services. The Google authentication service provides the application an authenticated user. Since all created objects are associated to an

authenticated user, segmenting data storage based on users email address, SMiT determines if the authenticated user has rights to perform each transaction requested.

Engine

The engine was written in Java, which continues to be a well-known, supported and relevant development language [Almog, 2013], to provide cross platform support without writing the engine more than once. The engine utilizes the Restlet Framework to access the RESTful web services hosted on the GAE, providing access to the scripts to execute locally and logging facilities.

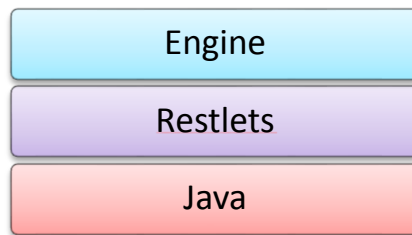


Figure 9 (Engine Stack)

However there were some challenges with engine authentication, due to the fact the engine is not running as an authenticated user. To resolve the authentication concerns the engine provides an access token to the RESTful web services for authentication. Once the token is validated, the requested data is provided to the Engine.

System Evaluation

To evaluate SMiT's scalability and flexibly, multiple SMiT Engines were installed on various machines across the internet. The engines were installed on PCs located behind firewalls and three of the PCs were running Windows and one was running OS X. All four SMiT Engines were asked to execute the same project and observed. The main observation of the scripts running on multiple OSes was the different paths each OSes required to access local files. The limitation of path delimiters hinders the idea of writing scripts once and running them anywhere. However to work around this a user could create multiple projects for different OSes. Future work could include updating plugins that require platform specific delimiters to be platform aware during execution to replace delimiters based on the executing OS.

To evaluate how intuitive SMiT's user interface is to create scripts, a user was asked to access the SMiT web client and create a simple script to copy a file from a one known location to another. The user immediately asked for documentation on how SMiT works. After some simple instructions the user was able to easily create a script and execute the project on one of the existing engines. To help new SMiT users a simple web based walk through explaining to users how to create a script could be provided, reducing confusion of how to create projects and scripts.

To evaluate how intuitive SMiT's user interface is at evaluating execution metrics, the same user was asked to access the SMiT web client to determine when a project was last ran, how long it ran, was the project successful and what did the project execute. The user was able to navigate and find the start time, end time of the last ran project as well as determine successful execution. The user was also able to find a detailed log of the scripts that the project executed on each engine. However, it was noted that having a dashboard to provide aggregated results would have reduced the time to understand the last execution of a project. The dashboard should include the following metrics: Number of failed and successful executions, Last successful execution, last failed execution and Average execution time.

Conclusion

While SMiT meet its goal of being a scalable, flexible and easy to use cloud based script management tool, SMiT has several possibilities for future work in the areas of project execution scheduling, dashboards, scripting and plugin development. The first future area for SMiT is in the scheduling of project execution. Currently projects can be scheduled as repeating jobs occurring at a set frequency. While project execution repetition works well, projects may not need to be executed during a period of predefined time or a specific system state. Henceforth, SMiT would benefit greatly with a more sophisticated *triggering system* for project execution. Project execution could be based on a trigger that monitors for specific conditions to be met. Triggers could then be based on a predefined schedule, system events, application or system log activity, system performance or file system changes.

Another opportunity would be in the logging, trending and display of captured data by plugins in easy to understand dashboards. Performance centric plugins should be able to provide historical information to users in a usable format such as graphs and charts. While it might be useful for a user to drill down to textual logs of captured system performance at a specific moment in time it does make it difficult to see trends in performance. For example seeing the utilization of the CPU in a graph over a period of time would be more useful than a specific moment in time.

Finally, for SMiT to be ready for an enterprise several plugins would need to be created and the engine & plugins need to be self-updating. The current available set of plugins would only be useful for simple application deployment activities. The following list of plugin groups would of a recommended starting point for plugin development:

- Amazon EC2
- Google App Engine
- Source Control
- System Performance
- Database Performance & Execution
- Web Services (SOAP/REST) Execution

Also the extensible plugin framework could be documented for enterprises, or external communities, to develop their own SMiT plugins. As well, SMiT could offer an online library for submitted plugins and a forum or FAQ for script creation & support.

SMiT achieving the goal of providing a scalable, flexible and intuitive SaaS based framework for automation across three facets: infrastructure and application monitoring, continuous integration and system maintenance is unique in the industry. While SMiT has several growth opportunities with the development of plugins, dashboards and project execution triggers before an enterprise might consider adoption, it is one of the first SaaS tools in this market segment. If future work is focused in the outlined areas of future growth, SMiT could quickly become one of the first viable SaaS script management automation tools in the market today.

Bibliography

- Almog, Shai. "Why Java Is More Relevant Than Ever in the Mobile Age." 21 January 2013. *sys-con*. <<http://java.sys-con.com/node/2513729>>.
- Asay, Matt. "The rising importance of cross-platform apps." 18 January 2010. *cnet*. <http://news.cnet.com/8301-13505_3-10436518-16.html>.
- "AutoMate." March 2013. *NetworkAutomation*. <<http://www.networkautomation.com/automate/>>.
- Bamboo*. n.d. March 2013. <<http://www.atlassian.com/software/bamboo/overview>>.
- Cerny, Jeff. "Network Automation: How companies are making the transition to a SaaS model." 1 March 2013. *ZDNet*. 11 March 2013. <<http://www.zdnet.com/network-automation-how-companies-are-making-the-transition-to-a-saas-model-7000012026/>>.
- CruiseControl*. n.d. March 2013. <<http://cruisecontrol.sourceforge.net/>>.
- CruiseControl - Distributed Extensions*. n.d. Web Page. 26 03 2013. <<http://cruisecontrol.sourceforge.net/distributed/index.html>>.
- Dubie, Denise. "IT automation technology dominates Vegas conferences." 21 May 2009. *Networkworld*. <<http://www.networkworld.com/news/2009/052109-interop-forrester-automation.html>>.
- ENTERPRISE MANAGEMENT ASSOCIATES. "ManageEngine: Shaking Industry Complacencies." October 2009. *ManageEngine*. ENTERPRISE MANAGEMENT ASSOCIATES, (EMA). <<http://www.manageengine.com/it360/ema-manageengine.pdf>>.
- Fielding, Roy Thomas. "Architectural Styles and the Design of Network-based Software Architectures." 2000. PH.D. Dissertation University of California, Irvine, CA. <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- "FinalBuilder." March 2013. *VSoft Technologies Pty Ltd*. <<http://www.finalbuilder.com/>>.
- Finch, Curt. "The Benefits of the Software-as-a-Service Model." 2 January 2006. *ComputerWorld*. <http://www.computerworld.com/s/article/107276/The_Benefits_of_the_Software_as_a_Service_Model>.
- Hudson*. n.d. March 2013. <<http://hudson-ci.org/>>.
- Jenkins*. n.d. March 2013. <<http://jenkins-ci.org/>>.

- Kaseya. "Automating IT Systems Management." 2010. *Kaseya*.
<http://www.kaseya.com/download/en-us/white_papers/kaseya.smee.whitepaper.pdf>.
- Kenney, Brad. "Top 12 Benefits of the SaaS Model." 14 May 2008. *IndustryWeek*.
<<http://www.industryweek.com/information-technology/top-12-benefits-saas-model>>.
- Manage Engine*. n.d. 26 03 2013. <<http://www.manageengine.com/>>.
- Network Automation. "Solving Scripting Problems with Technology." August 2010.
NetworkAutomation.
<<http://www.networkautomation.com/documents/4c87c1d291f20840556363.pdf>>.
- Prigge, Matt. *Best of open source in enterprise monitoring*. 10 September 2007.
<<http://www.infoworld.com/d/security-central/best-open-source-in-enterprise-monitoring-107>>.
- Rodriguez, Alex. "RESTful Web services: The basics." 06 November 2008. *IBM*.
<<http://www.ibm.com/developerworks/webservices/library/ws-restful/>>.
- "Why App Engine." 26 June 2012. *Google App Engine*.
<<https://developers.google.com/appengine/whyappengine>>.
- Zucco, Jim. "Benefits of a software as a service model." February 2009. *TechTarget*.
<<http://searchenterprisewan.techtarget.com/feature/Benefits-of-a-software-as-a-service-model>>.

Appendices

Appendix A: Installing the SMiTEngine

1. The SMiT web client can be found at <http://scriptmgmttool.appspot.com/>
2. Create an engine in SMiT (<http://scriptmgmttool.appspot.com/#engines>)
3. Download SMiTEngine.zip and Extract to a folder and update the properties file

```
server=scriptmgmttool.appspot.com
engineId=<engineid>
accessToken=<accesstoken>
refresh=1
```

engine.properties

- a. `<engineid>` - is the id of the engine you created
- b. `<accesstoken>` - is the access token provided when adding an engine

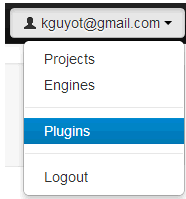
Engines

Id	Title	Server	Access Token	Enabled
68001	Engine 1	Desktop	1234	true
89001	2	Laptop	1234	true

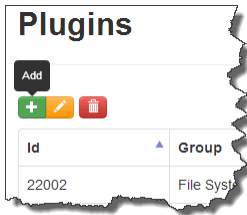
4. Start the engine: `java -jar SMiTEngine.jar` from a console window
 - a. If Launching the Engine to work with a proxy
`java -Dhttp.proxyHost=<host> -Dhttp.proxyPort=<port> -jar SMiTEngine.jar`

Appendix B: Creating a Web Client Plugin UI

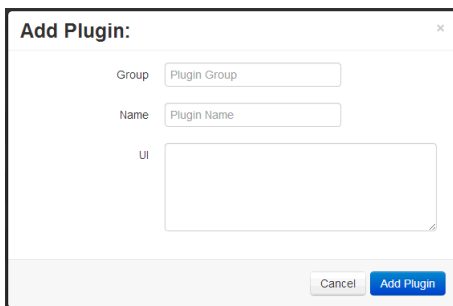
1. Open a Browser and go to the following url: <http://scriptmgmttool.appspot.com/>
2. Login to SMIIT
3. Select Plugins from the user drop down menu



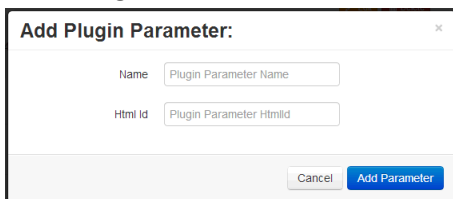
4. Add a new Plugin by pressing the Add button from the tool bar.



5. Provide the Group the plugin belongs to and the Name of the plugin. These names will need to match the Group and Name set in the engine plugin when that is developed. Also provide the UI. The UI needs to be defined in HTML tags and the input boxes need to be assigned an id. Note the id as it will be needed later.

A screenshot of the 'Add Plugin' dialog box. It contains three input fields: 'Group' with the value 'Plugin Group', 'Name' with the value 'Plugin Name', and 'UI' which is an empty text area. At the bottom right, there are 'Cancel' and 'Add Plugin' buttons.

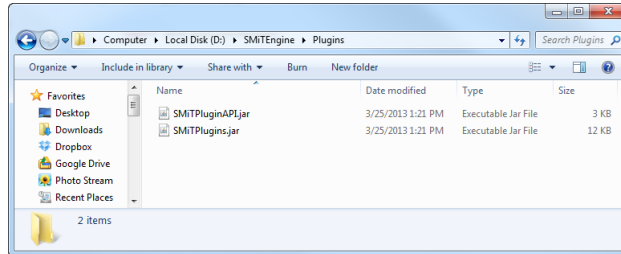
6. Once you have added the plugin select it from the grid and press edit. You should now be able to add parameters to the plug.
7. When adding a parameter the Html id is required and the name referencing that input inside the engine. When the Engine needs access to the value provided in the input box, it maps the html id to the internal name of the parameter, providing the user supplied value to the engine.

A screenshot of the 'Add Plugin Parameter' dialog box. It contains two input fields: 'Name' with the value 'Plugin Parameter Name' and 'Html Id' with the value 'Plugin Parameter Htmlid'. At the bottom right, there are 'Cancel' and 'Add Parameter' buttons.

8. Now that all the parameters are added you can utilize this plugin in scripts. However, until the engine has a plugin created to relate to this Web Plugin no real execution will execute.

Appendix C: Creating an Engine Plugin

1. Create a Java Project
2. Add the SMITPluginAPI.jar to the project as a referenced library.
 - a. The SMITPluginAPI.jar can be found in plugins folder of the Engine.



- b. To download the engine login to the SMiT Web Client and go to the following url: <http://scriptmgmttool.appspot.com/#engines> From there you can download the latest engine
3. Create a new Class and inherit the class from guyot.smit.plugin.Plugin

```
import guyot.smit.plugin.Plugin;

public class CopyFile extends Plugin {
    public CopyFile() {
        setGroup("File System");
        setName("Copy File");
    }

    void Initialize();
    long ExecuteAction();
    void Cleanup();
}
```
 4. Create a constructor and set the Group the plugin belongs too and the Name of the plugin. These are used to uniquely identify the plugin. If the group and name don't match the setup provided in the Web Client the engine will not find the plugin and the scripts will fail.
 5. Override the following methods
 - a. Initialize() – is executed on plugin creation. This allows the plugin writer to setup any objects required for use in the ExecuteAction() method.
 - b. ExecuteAction() – Is main function of the plugin, performing the required steps for the action to complete. This function should return 0 for success and anything else as an error.
 - c. Cleanup() – Is executed when the plugin is done executing allowing the plugin writer to clean up an objects created during Initialize.
 6. Export the Java Project to a Jar file.
 7. Place the Jar file in the Plugins Folder where the SMITEngine.jar is located
 8. If the SMITEngine.jar is running, restart the SMITEngine.jar to load the new Plugin.