PowerShell Desired State Configuration (DSC) in the Enterprise

by

Derek Foos, Nathan Deems, Kyle Klingler

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Derek Foos, Nathan Deems, ~~Kyle~~ Klingler 4/17/17

4/17/17

Brian Verkamp 4/17/17

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2017

Table of Contents

List of Illustrations

Abstract

Modern businesses rely on computer systems for their daily operations. Specific configurations allow these systems to function properly so that work can be completed. Our project demonstrated how PowerShell Desired State Configuration (DSC) could be used to manage those configurations and ensure that they remained in the desired state by preventing configuration drift. For our project, we built several virtual machines in UC's Sandbox virtual environment that represented systems that might be found in a small business. These included systems like file servers, web servers, and Linux machines. We then wrote configurations that defined attributes on these systems and leveraged PowerShell DSC to deploy and manage the configurations on those systems using a central repository. In doing so, we demonstrated how PowerShell DSC could be used to prevent configuration drift and keep those systems running and their services available.

1. Problem Statement

1.1 Introduction

Most small businesses do not have the time nor resources to setup, deploy, and maintain device configurations throughout the business along with all their other job responsibilities. As cloud technology continues to grow and become more complex, the need for configuration management tools is increasing. Device configurations may inadvertently change over time and potentially cause issues with systems and the services they provide. This project explored PowerShell DSC (desired state configuration) as a solution to this problem. PowerShell DSC allows administrators to deploy and then maintain device configurations using PowerShell DSC configurations on a platform that is built into Windows. PowerShell DSC uses declarative configurations that can be either pushed out or pulled in to different targets, allowing for more ease-of-use and flexibility. Our goal with this project was to create a network representing that of a small business environment in the UC IT Sandbox, and then integrate PowerShell DSC for our configuration management needs. We wrote configuration scripts to deploy and manage our configurations to make our environment as efficient and easy to use as possible.

1.2 Project Description

We created a virtual environment representing systems that could be found in a small business environment. This was created and implemented in the UC IT Sandbox. Once the network had been developed, we then setup a PowerShell Desired State Configuration pull server. This allowed clients to pull configurations from the server and then effectively manage the configurations on those systems. Using PowerShell DSC, we demonstrated how it could keep device configuration up to date, how this system could be highly scalable, and prevent unnecessary IT work.

1.3 Problem

As technology continues to grow and user's reliance on technology continues to expand exponentially, most businesses are having to manage an increasing amount of computer systems. Some of these systems are physical units and others are virtualized or cloud based. It is common for some businesses to have hundreds, if not thousands of these systems to manage. Configuring and reconfiguring an environment with so many systems is difficult and this difficulty is the basis for our project. We believe the solution to this issue is using PowerShell DSC to manage device configurations.

1.4 User Profile

The primary user of our project is a systems administrator. This would be the person within a business that is responsible for managing the configurations of user computers and business servers. A systems administrator who plans to implement PowerShell DSC should have a deep understanding of the Windows operating system. If they plan to manage Linux machines as well, they should also be comfortable with Linux and its many variants. Knowledge of scripting languages is also important. Proficiency in PowerShell is key. Having used technologies like Puppet or Chef would also be helpful for any systems administrator trying to learn how to implement PowerShell DSC.

See Table 1 below for a detailed user profile.

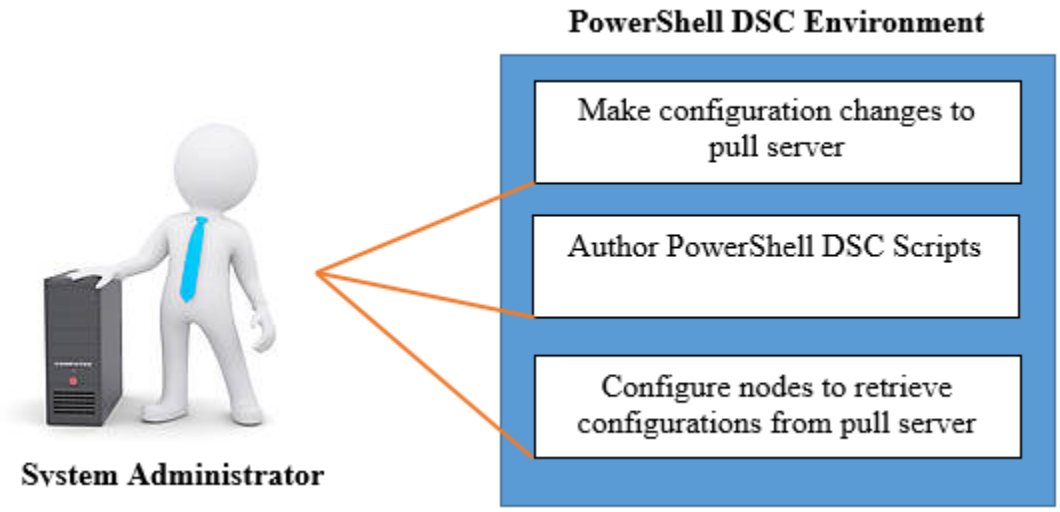
Table 1 - User Profile Form

User Profile Form
Application: PowerShell Desired State Configuration (DSC)
Potential Users: System Administrators
Software and Interface Experience: The user should have extensive knowledge of Windows and Linux operating systems (both server and client's OS's) and be familiar with configuring these OS's both through the GUI and via the command prompt/terminal. Users should also have basic knowledge of PowerShell or another scripting language.
Experience with Similar Applications: Both Chef and Puppet provide similar services. Familiarity with group policy and system administration. Knowledge of CMD or another scripting language is a plus.
Task Experience: Administrators can create PowerShell DSC scripts to quickly perform system configurations.
Frequency of Use: Most of the work will be with determining what to automate and creating the initial PowerShell scripts. Once the initial scripts are created they can be setup to run automictically, thus limited changes will be required other than if additional tasks need automated or if existing tasks need modified.
Key Interface Design Requirements that the Profile Suggests: Administrators can create PowerShell DSC scripts to automate time consuming and or repetitive tasks.

1.5 Use Case Diagram

Our use case diagram, Figure 1 below, shows our single use case, in which a system administrator is responsible for implementing and maintaining PowerShell DSC in a small business environment.

Figure 1 - Use Case Diagram



2. Project Management

2.1 Budget

Table 2 below represents the real-world cost that it would take to perform this project. The total came out to be $27,654. We performed this project for $0 since we donated our time to this project and we utilized the resources provided to us through the School of Information Technology and the University of Cincinnati.

Table 2 - Project Budget

No.	Item	Unit(s)	Price/Unit	Total
	Networking			
1	Labor (Setup VMs, network, configurations etc.)	90	$85.00	$7650.00
	Software			
2	Labor (script creation)	150	$85.00	$12750.00
3	Windows Server 2012 R2	6	$1209.00	$7254.00
4	CentOS	2	$0.00	$0.00
	Supplies			
6	Other	0	$0.00	$0.00
Subtotal				$27654.00
Cost to Group				$0.00

2.2 Objectives/Deliverables

At the completion of our project, we had built a replica business environment in the UC IT Sandbox that contained various severs, such as web servers, file servers, a domain controller, and other hosts. Various configurations on these machines were managed by a PowerShell DSC pull server that machines could query for their correct configuration.

The first system that we completed was a PowerShell DSC pull server that was responsible for storing configurations and delivering them to hosts who requested them. This allowed configurations to be centrally managed.

The second system was a file server managed by PowerShell DSC. This file server had a second drive that was formatted and shared via SMB (server message block) on the network. The formatting of the drive and the existence of the share were managed by PowerShell DSC. The file server also had a high-level folder hierarchy and specific domain user permissions for those folders also managed by PowerShell DSC.

The third system in our environment was a web server hosting three static websites. PowerShell DSC ensured that the IIS Windows feature was installed, that the IIS console was installed, that the default IIS website was stopped, that three new company websites were created and running on different ports, and that the new company websites used different directories and index.html files also managed by PowerShell DSC. The PowerShell DSC configuration also configured a web

application within each of the three websites using different directories and index.html files managed by DSC, and then configured two web app pools that the web applications belonged to.

The fourth system in our environment was a Linux machine hosting a Samba file share. PowerShell DSC was used to install Samba, configure a share, set up user permissions, configure firewall settings, and start both the firewall and Samba services.

The fifth system we created was another Linux machine with a LAMP (Linux, Apache, MySQL, PHP) environment. MariaDB was used in place of MySQL and PowerShell DSC was used to ensure that these applications were installed, that an index.html file with our abstract on it existed and was being hosted by Apache, that a PHP config site was running at test.php and that these services remained started.

The sixth system in our environment was an Active Directory server. PowerShell DSC was used to install Active Directory (AD) and its management tools, create a new forest and domain, create AD users, and create groups and organizational units that might be used to organize a small AD environment.

The final system in our demo environment was a print server. PowerShell DSC was used to ensure that the print services role was installed and ready to be configured.

A detailed timeline of when our project goals were completed are detailed in our Gantt chart and project timeline that are listed below.

2.3 Gantt Chart and Project Timeline

The below Gantt Charts, Table 3 and Table 4, are our projected timelines for both Fall 2016 and Spring 2017. We have more detailed timelines listed below in Table 5 and Table 6, theses tables show our weekly objectives and any deliverables that we worked on during a week.

Table 3 - Fall 2016 Gantt Chart

First Semester

Week of...	18-Sep	25-Sep	2-Oct	9-Oct	16-Oct	23-Oct	30-Oct	6-Nov	13-Nov	20-Nov	27-Nov	4-Dec	11-Dec	18-Dec	25-Dec
Sandbox access															
Research small business environments															
Research PowerShell DSC															
Identify network/security requirements															
Network Implementation															
Identify Powershell DSC attributes for each system															
Create and implement some basic scripts															
Testing/tweaking demo network and cofigurations															
End of year wrap up															
Holiday break															

Table 4 - Spring 2017 Gantt Chart

Second Semester

Week of...	1-Jan	8-Jan	15-Jan	22-Jan	29-Jan	5-Feb	12-Feb	19-Feb	26-Feb	5-Mar	12-Mar	19-Mar	26-Mar	2-Apr	9-Apr
Holiday break															
Regroup and review current progess and next steps															
Divide deliverables up among group members															
Setup additional Windows servers (AD, print, & IIS)															
Setup Linux Servers (Samba, MySQL, and Apache)															
Create configurations for new Windows servers															
Create configurations for new Linux servers															
Review configurations and setup															
Finalize environment and configurations															
Prepare for Tech Expo															
IT Expo															

Table 5 - Fall 2016 Weekly Timeline

Week	Tasks
18-Sept	• Request sandbox access
25-Sept	• Start researching small business environments • Start figuring out what VMs are needed for our virtual network
2-Oct	• Continue researching small business environments • Start researching PowerShell DSC and its capabilities
9-Oct	• Continue researching PowerShell DSC and its capabilities
16-Oct	• Identifying networking and security requirements • Look into best practices

23-Oct	• Continue identifying networking and security requirements • Continue looking into best practices
30-Oct	• Start implementing virtual network and VMs in the IT sandbox • Continue identifying networking and security requirements • Continue looking into best practices
6-Nov	• Continue setup of server and client VM's • Start identifying PowerShell DSC attributes for each system
13-Nov	• Finalize setup of server and client VM's • Continue identifying PowerShell DSC attributes for each system • Start creating test configurations
20-Nov	• Continue identifying PowerShell DSC attributes for each system • Continue creating test configurations • Continue testing test configurations
27-Nov	• Continue creating test configurations • Continue testing test configurations
4-Dec	• End of year group wrap up
11-Dec	• Holiday break
18-Dec	• Holiday break
25-Dec	• Holiday break

Table 6 - Spring 2017 Weekly Timeline

Week	**Tasks**
1-Jan	• Holiday break
8-Jan	• Verify sandbox setup still works • Regroup and review current progress and next steps
15-Jan	• Meet with academic advisor to discussion project status and deliverables • Divide deliverables up among group members
22-Jan	• Setup additional Windows servers (AD, print, second IIS server) • Setup Linux Servers (Samba, MySQL, and Apache)
29-Jan	• Start creating configurations for new Windows servers
5-Feb	• Start creating configurations for Linux servers • Continue creating Windows server configurations
12-Feb	• Continue creating Linux server configuration • Start reviewing current setup and configurations
19-Feb	• Continue reviewing current setup and configurations. • Make any changes if needed
26-Feb	• Continue reviewing current setup and configurations • Make any changes if needed
5-Mar	• Start finalizing environment and configurations
12-Mar	• Continue finalizing environment and configurations • Start preparing for Tech Expo
19-Mar	• Continue preparing for Tech Expo
26-Mar	• Continue preparing for Tech Expo

2-Apr	• Continue preparing for Tech Expo • Have project completed and ready to present
9-Apr	• Finalize all project work, complete any final touches • Setup and prepare for IT Expo

3. Technical Elements

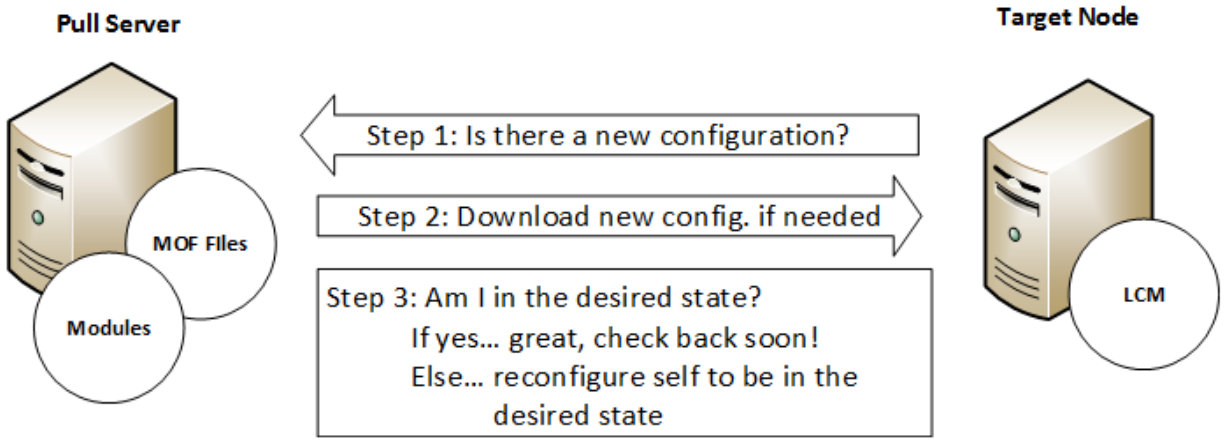
3.1 Technical Elements

The key technical element of our project was PowerShell DSC itself. It was the framework that allowed us to declare configurations and deploy them to machines. The UC IT Sandbox was also paramount to the success of our project because it gave us all of our resources like computing power, operating systems, and network to build out a replica business environment to apply our PowerShell DSC configurations to.

3.2 Technical Approach/Methodologies

The general approach we took to this project was trial and error. Our group was very unfamiliar with PowerShell DSC so we first had to do a lot of research around it to understand the framework and best practices. Once we identified best practices and felt comfortable with how PowerShell DSC should be implemented, we started building a series of systems with managed configurations. It was decided that we would use a pull server to deliver configurations to our managed systems. This allowed hosts to contact the pull server at a set interval and check to make sure it was configured as it was supposed to be (See Figure 2 below). By having this central repository working, adding systems, and scaling the environment became easy. With the pull server running and some configurations already managed, our next steps were to write additional scripts that automated the configuration of various types of systems that were housed in the Sandbox (our college's infrastructure as a service environment).

Figure 2- PowerShell DSC Workflow



3.3 Testing Overview

Testing is an important part of any project. It is a way to show that you are meeting project requirements and that you can provide the deliverables that the project seeks to implement. We defined our project requirements as:

1. A functioning PowerShell DSC pull server

    a. It can communicate with other network devices

    b. Devices can successfully pull configurations from it

    c. Multiple configurations can be deployed from a single server

2. Scalable automated configurations

    a. Configurations can be applied to multiple systems

    b. Configurations are applied automatically

c. Devices can automatically reapply their defined configuration

3. A system compatible with a mixed Windows and Linux environment

a. Active Directory

b. IIS server

c. File server

d. Print server

e. Apache server

f. MySQL server

g. Samba server

In Table 7 below, we outlined the tests we performed to determine that the project was meeting the requirements we listed above. We listed the actions we performed while testing. Each action is linked to a requirement and has an expected outcome. If the action matches the expected outcome, then it passes the test. If it does not, then it fails.

Table 7- Test Cases

Req. #	Input/Action	Expected Outcome	Actual Outcome	Pass/ Fail	Explanation If Failed	Date
1	Verify Pull Server web service is responding	See XML webpage	See XML webpage	P		10/1
1a	Verify Pull Server can communicate with network	Successfully ping client device	Ping client device	P		10/1
1b & 3	Test pull Windows configuration	Configuration deployed to host	Configuration setup name & IP	P		10/14
1b & 3	Test pull Linux configuration	Configuration deployed to host		P		4/07
3b	Test IIS configuration	Configuration deployed to IIS Server	Configuration installed IIS role, deployed website	P		11/2
2a & 2c	Delete IIS role	IIS reinstalls at scheduled time	IIS reinstalls, setups website	P		11/2
3c	Test File Server configuration	Configuration deployed to File Server	Configuration enables disk & shares network share	P		10/27
2a & 2c	Delete network share	Share is recreated at scheduled time	Share is recreated	P		10/27
3d	Test Print Server configuration	Configures Print server	Installs print server roll	P		3/17
3a	Test AD configuration	Configures AD server	Installs/configures installed AD role	P		3/15
2a & 3b	Test IIS websites	Websites can be reached	Websites are available	P		4/09
3g	Test Samba configuration	Configuration deployed to Samba server	Errors out	F	Failing to apply due to firewall rules	2/24
3e	Test Apache configuration	Configuration applied to Apache server	Installs and configures Apache. Creates website	P		2/25
3f	Test MariaDB configuration	Configuration applied to MariaDB server	Installs and configures MariaDB	P		2/29

3.4 Testing Conclusion

As the testing report shows, we met the majority of our requirements and ran many successful tests. The only deliverable we did not fulfill was the Samba server configuration. By following the testing plan, we ensured that we didn't overlook anything and that our environment was configured and operating as intended.

3.5 Problems Encountered

Working with a brand-new technology that none of our team had any prior knowledge of was a great challenge and learning experience. As with learning any new piece of technology, there comes various problems along the way. PowerShell DSC brings forth many features and options, which brings out difficulties in selecting what we wanted to do with this project.

1. Initial setup - The first issue we ran into, was how to setup and deploy PowerShell DSC. There are many ways to implement PowerShell DSC and some requirements that must be met like having a system with at least Windows Management Framework (WMF) 4.0 (Zerger 2014).

2. Push vs. Pull modes - Desired states can either be pushed out from the server onto targets, or pulled from the server by clients (Team 2013). Each method ends in the same result but requires a different process. It was difficult to decipher which route we wanted to take with our environment so it required us to research and learn about both modes. Ultimately, we decided on a pull server so we could set up an initial server and add more clients as we go to pull from it.

3. Configuration iterations – Every time a change was made to a configuration, it had to be recompiled, redeployed, and then evaluated through logs to ensure it did not fail. This process was very slow which caused some trial and error situations in development to take significant time.

4. Samba failed credentials- We were unable to find a way to successfully setup Samba credentials for an SMB share through a DSC configuration. We are unsure if this is even possible to configure through PowerShell DSC.

4. Conclusion

This project has allowed us to learn about PowerShell Desired State Configuration and how to implement it in a small business environment to manage system configurations. We learned about best practices like using a pull server so that configurations are centrally managed. We've also learned how to write configurations and then use our pull server to deliver them to hosts. By doing so, we demonstrated how a variety of systems could easily be managed with PowerShell DSC which has the potential to save systems administrators time and money.

Bibliography

Team, PowerShell. *Push and Pull Configuration Modes.* November 26, 2013. https://blogs.msdn.microsoft.com/powershell/2013/11/26/push-and-pull-configuration-modes/ (accessed 27 September, 2016).

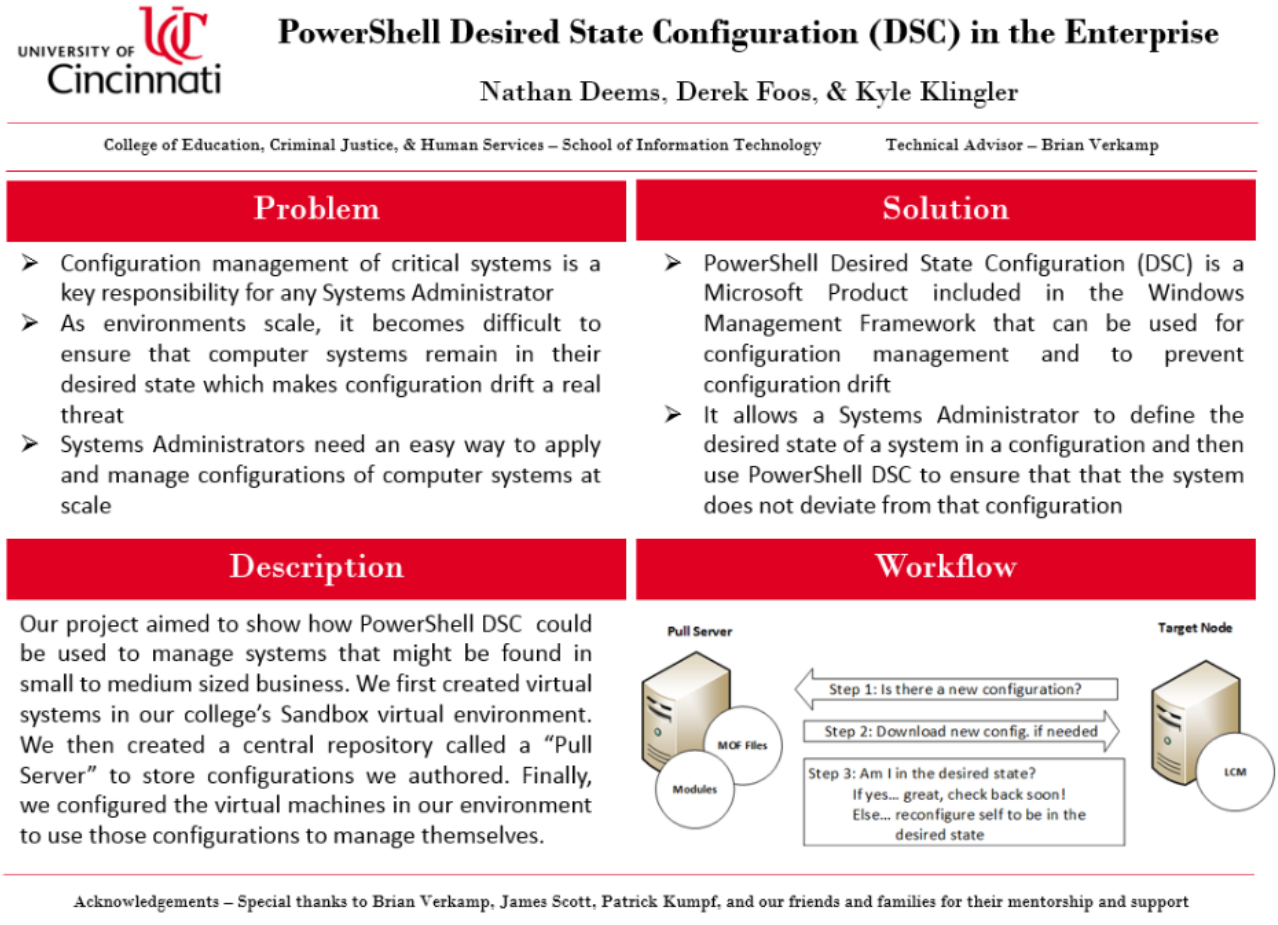
Zerger, Pete. *Day 1: Intro to PowerShell DSC and Configuring Your First Pull Server.* August 2, 2014. http://www.systemcentercentral.com/day-1-intro-to-powershell-dsc-and-configuring-your-first-pull-server/ (accessed October 10, 2016).

Appendix

Acronyms and Definitions

DSC – Desired State Configuration
IT - Information Technology
LAMP – Linux, Apache, MySQL, PHP
PHP- Hypertext Preprocessor
SMB- Server Message Block
UC – University of Cincinnati
WMF – Windows Management Framework

Expo Poster



Configuration Script Example