Automating Computing Infrastructure Configuration with Emphasis on System-Level Design and Integration

Shaun M. Lynch, Ph.D.
Department of Mathematics and Computer Science
University of Wisconsin-Superior
Superior, WI 54880
slynch@uwsuper.edu

Abstract

As computing and server platforms add new capabilities and increase functionality, the gap between the systems perspective and component configuration grows wider and more difficult to span. Improvements come at the cost of making a considerable number of detailed settings through a command line or graphical user interface. The ability to make a system operational relies heavily on technical knowledge and skills needed to understand the configuration and operation of specific components. This creates a substantial barrier to those unfamiliar with the design and construction of computing infrastructures and hinders the ability to delegate responsibility to other individuals. This paper explores the challenge of managing system setup and configuration and presents a prototype application that places emphasis on system-level design and integration while automating computer and network settings.

1 Introduction

As computing and server platforms add new capabilities and increase functionality, the gap between the systems perspective and component configuration grows wider and more difficult to span. Improvements come at the cost of making a considerable number of detailed settings through a command line or graphical user interface. Although advances have been made to enhance the user experience and automate deployment, the underlying process has changed little over time. The ability to make a system operational relies heavily on technical knowledge and skills needed to understand the configuration and operation of specific components. This creates a substantial barrier to those unfamiliar with the design and construction of computing infrastructures and hinders the ability to delegate responsibility to other individuals.

The reality of this situation became clear to the author after a two year effort to design, construct, and implement a computing infrastructure to support the academic programs of the Department of Mathematics and Computer Science at that University of Wisconsin-Superior. The centralized server and network infrastructure used by the department relies heavily on virtualization to host academic servers and applications needed for classes, scholarship, and independent study. The infrastructure serves over 200 students and ten faculty and staff members per semester and supports 50 workstations in two advanced computing laboratories and nearly 20 computers used for prototyping, training, lab support, and system management.

This paper explores the challenge of managing system setup and configuration and presents a prototype application that places emphasis on system-level design and integration while automating computer and network settings. The discussion begins with a brief overview of the background and circumstances leading to this situation with special emphasis on system administration processes. Next, the paper explores the problem in depth using a demonstration system and quantifying key measures. Finally, an automated configuration process is presented along with the design of the prototype application and measures of performance.

2 Background and Circumstances

The summer of 2012 saw the final efforts to install the department's academic server and network infrastructure. The core system consisted of two computers hosting domain and internet protocol address management services; two computers configured as a failover cluster that provides a virtualization platform for operational and academic servers; and a storage area network (SAN) that connects an iSCSI disk array and other shared storage devices. The system was made operational at the end of August in time for classes beginning Fall Semester and activities continued throughout the months of September and October to build an effective Active Directory structure, configure and deploy group policy, add accounts, and tailor user rights and privileges.

2.1 Setbacks

Although the system was successfully brought online in a timely manner, shortfalls in department staffing, graduation of key students involved in the initial design, and lab assistant turnover led to a situation where all the work needed to fulfill the system became the responsibility of the author. Steps had been taken early in the process to avoid this situation to include communication with administrative authorities, budget requests for support personnel, involvement of student lab assistants, *et cetera*. State budget cutbacks for higher education compounded the problems impacting a number of areas ranging from staffing decisions to funds available to support technology investments.

Further attempts to cultivate a pool of individuals capable of supporting the system in the following months proved difficult. Training student lab assistants became increasingly problematic due to turnover, limited student work hours, and the inability to hire work-study students outside the academic school year. Even a call to department colleagues to help co-administrator the system was met with tepidness given their unfamiliarity with the design and amount of work needed to learn the system.

A pattern emerged as the circumstances began to unfold. First, system complexity and steep learning curves created a barrier to entry for those not familiar with the infrastructure. Second, the system did not lend itself to being partitioned into smaller, more manageable subsystems. Third, configuration concepts were difficult to document and communicate making it difficult for those trying to learn the system thus increasing the time needed to acquire the necessary skills.

As the quandary mounted to develop a pool of individuals capable of supporting the department's computing infrastructure, the author began to explore the possibility that difficulties encountered may be symptomatic of problems in the underlying development process. Specifically, the process did not facilitate the transfer of configuration knowledge from the development phase to the operations phase so that the system could be sustained long-term for a small, resource constrained organization prone to shortfalls. Another way would have to be found to work around the personnel and fiscal constraints of the organization.

2.2 Observations

Preparations to bring the core computing infrastructure online brought to light several challenges. To begin with, building a system from the ground up requires a large number of settings to install and configure operating systems, domain services, and virtualization hosts. Some settings are repetitive and must be made uniformly for each computer (e.g. time zone, update policy, etc.). Other settings share a common procedure but are tailored to the particular hardware or capability a computer has (e.g. networking, storage, multipath IO, iSCSI initiators, etc.). While other settings are unique to a particular service (e.g. AD DS, DNS, DHCP, Hyper-V, failover clustering, etc.) and are made to enable or customize functionality or configure a best practice.

Documenting system configuration is very difficult particularly if the intended audience is unfamiliar with the system. Ideally, documentation would include rationale and context in which settings are made, the sequence and mode which settings are entered, limitations and alternatives, and references to additional resources. However, documenting all of these facets becomes increasingly burdensome as the number of system settings increase and quickly expands beyond the capabilities of a resource constrained organization. In this instance, it became infeasible to document to a suitable depth the hundreds of steps needed to configure the system given the limited resources.

As key personnel turnover, tacit knowledge and the experienced gained by participating in the development process cannot be easily replicated or substituted. Although documentation can capture explicit knowledge, step-by-step procedures do not necessarily communicate the underlying design concepts and principles discovered by those participating in the development and support processes. Tacit knowledge and experience are particularly important when resolving configuration issues, diagnosing problems, or adding new functionality to a complex system containing multiple subsystems that interact in non-trivial ways.

Errors and omissions are inescapable given the numerous steps and settings needed to properly configure the system. Keying errors are unavoidable and steps often get overlooked when trying to repeat the process from memory. Some procedural aides such as cue sheets, screen shots, and command lists are useful, but also compete for cognitive resources when entering settings in the system.

Entering settings manually takes a long time. Based on the author's experience, a complete system build takes between six and eight hours. A portion of the time is waiting for an activity to complete such as promoting a domain controller. However, a lot of time is spent navigating the various windows, dialogs, and wizards needed to enter settings.

3 The Challenge Illustrated

A demonstration system was prepared to highlight the difficulties of configuring even a modest computing infrastructure. The system is based on model being considered to update the department's academic computing platform. However, the system would also be a suitable candidate for an infrastructure used in a modern day midsized organization.

3.1 Setup

The example system shown in Figure 1 is provided for demonstration purposes. All the basic components are present including domain and internet protocol address management services, a virtualization platform employing failover clustering, an active directory structure consisting of 18 organizational units, and thirteen predefined group policy objects ready for import. Settings that allow the system to adhere to best practices have been included plus a customization deemed useful in the current deployment.

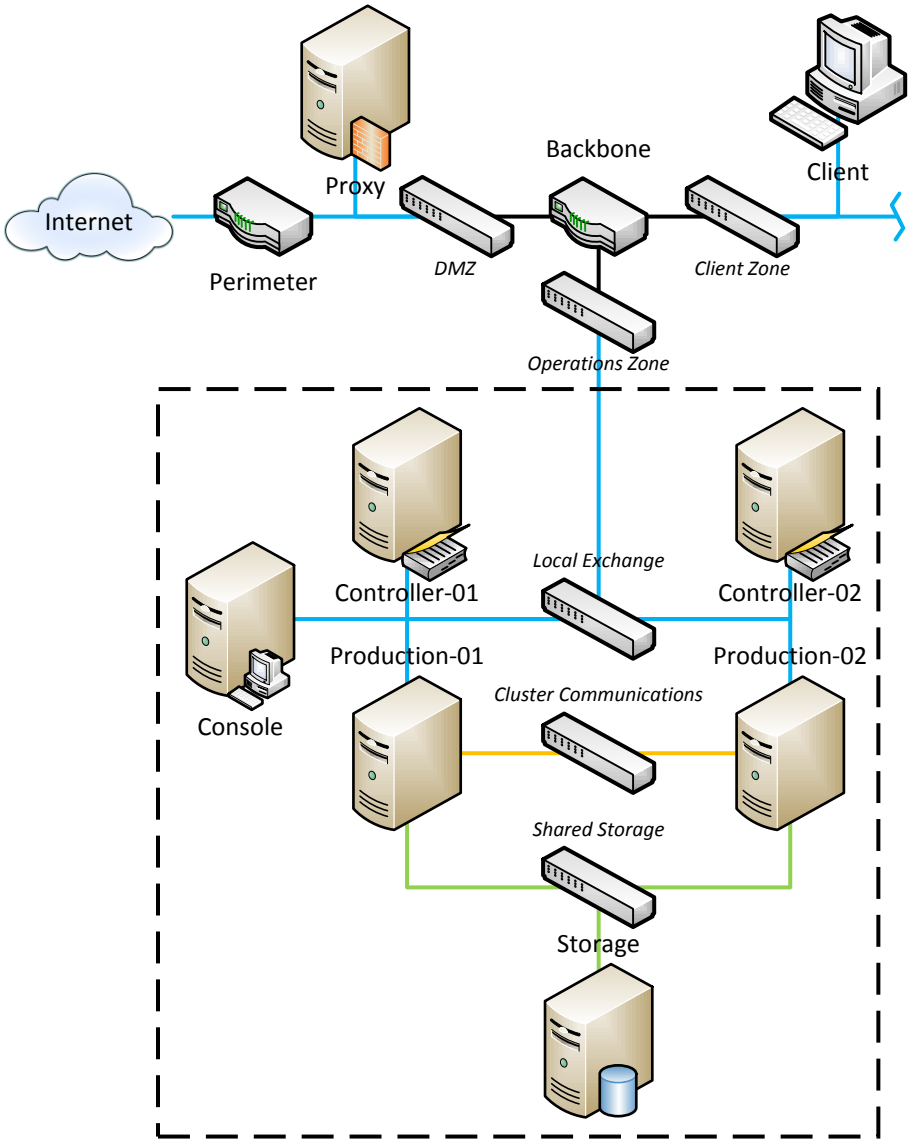

Figure 1: Overall system and core infrastructure used for demonstration

The diagram illustrates a multi-zoned network connected to a centralized router (backbone) partitioned into demilitarized, client, and operations zones. Specifically, the demilitarized zone (DMZ) is the portion of the network that connects to the internet and contains internet-facing devices such as web proxies, email servers, and web servers. The client zone is an internal network that would normally connect employee workstations in a firm or a computer lab in an academic institution. The operations zone is an internal network that connects the domain infrastructure, virtualization cluster, and shared storage used to host back-office applications for the organization.

For purposes of this study, the system is hosted in a virtual environment using VMware Workstation 9. The perimeter router, backbone router, and web proxy server are treated as network appliances and simulated using virtual machines running Ubuntu 12.10 x64 Server. The DMZ, Client Zone, Operations Zone/Local Exchange, Cluster

Communication, and Shared Storage switches are represented with virtual switches hosted by the VMware Workstation application. The client workstation shown in the diagram and is represented with a virtual machine running Windows 8 x64 Enterprise. The entire system exists within an isolated domain called *sandbox.net* and access the internet through a proxy server located in the DMZ.

The region enclosed by the black dashed line identifies the boundaries of the demonstration system. The primary functional components includes: domain services using Microsoft Active Directory and Domain Services (AD DS), internet protocol address services such as Domain Name Service (DNS) and Dynamic Host Configuration Protocol (DHCP), a virtualization platform employing a Microsoft Hyper-V failover cluster, and an iSCSI disk array in a separate storage area network (SAN). The functionality is distributed across six different computers:

- Two computers named `CONTROLLER-01` and `CONTROLLER-02` respectively, arranged in redundant configuration host the domain and internet protocol address services
- Two computers named `PRODUCTION-01` and `PRODUCTION-02` respectively, configured as a two-node failover cluster hosts the virtualization platform
- One computer named `STORAGE` configured as a storage server hosts the iSCSI disk array located in a storage area network
- One computer named `CONSOLE` that provides the management interface to the system

Connectivity within the operations zone is provided by three distinct local networks organized in the following manner:

- Local Exchange—Internal IPv4 and IPv6 network connecting the service ports on the domain controllers; virtual machine ports on the production cluster; and management ports on the production cluster, iSCSI disk array, and management console
- Cluster Communication—Private IPv6 network connecting nodes in the production cluster enabling cluster health, cluster shared volume, and live migration traffic
- Shared Storage—Private IPv6 storage area network connecting nodes in the production cluster to the iSCSI disk array

A common procedure was used to build the system from new computer (OS installed) to fully operational infrastructure. Activities include:

1. Configure computer `CONTROLLER-01`
2. Promote `CONTROLLER-01` to domain controller in new Active Directory forest
3. Setup core domain settings
4. Setup core domain administrators
5. Configure computer `CONTROLLER-02`
6. Promote `CONTROLLER-02` to domain controller in existing forest
7. Configure computer `CONSOLE`
8. Configure computer `STORAGE`

9. Configure computer `PRODUCTION-01`
10. Configure computer `PRODUCTION-02`
11. Setup DNS Service
12. Setup DHCP Service
13. Setup iSCSI Target
14. Setup Multipath IO (MPIO)
15. Setup iSCSI Initiators
16. Setup Hyper-V
17. Setup Failover Cluster (`PRODUCTION` cluster)
18. Setup Active Directory
19. Setup Group Policy

Each step encapsulates the major activities needed to configure the operating system on each computer and prepare the domain and internet protocol address services, virtualization platform, and shared storage components. It should be noted that the actual activity grouping will vary depending on the manner in which the system is built.

3.2 Results of a Manual Build

The system was built manually using the configuration and procedure previously described. Each computer in the system was cloned from a single virtual machine template with a fresh copy of Windows Server 2012 Enterprise (GUI), VMware tools, and the most recent updates installed. Prior to cloning, the template operating system was generalized and prepared for an out-of-box-experience using Microsoft's sysprep.exe utility. Once cloned, additional network ports and disk storage was added to specific virtual machines.

Activities were tracked using Microsoft's Step Recorder during the build process. Step recorder captures each action such as a mouse click, textbox entry, or command line statement that invokes a change of state. The only steps exempt from being recorded were those needed to start, restart, or shutdown the computer since that would impact the functionality of the step recorder program. Table 1 itemizes the steps needed to complete the activities described by the common procedure.

The complete process takes an experienced person (the author in this case) between six and seven hours and requires approximately 2,400 steps to build the system.[1] Priority was given to settings that could be made using the graphical user interface, and then settings made using command line statements. All actions were recorded in attempt to replicate a typical configuration process including navigation problems, adjusting windows to view content, fixing erroneous entries, and going back to complete missed steps.

[1] The author acknowledges that the results of this one demonstration is not a reflection of a particular operating system or vendor but to highlight the hierarchy of systems, subsystems, and components present in technology based information systems. Many factors influence the steps needed to configure a system such as available documentation, support, and training aids. It is arguable that an infrastructure built using other technology be Unix/Linux, VMware, etc. experience similar challenges.

Activity	Steps
Configure computer CONTROLLER-01	122
Promote CONTROLLER-01 to domain controller in new Active Directory forest	14
Setup core domain settings	44
Setup core domain administrators	63
Configure computer CONTROLLER-02	125
Promote CONTROLLER-02 to domain controller in existing forest	17
Configure computer CONSOLE	138
Configure computer STORAGE	229
Configure computer PRODUCTION-01	241
Configure computer PRODUCTION-02	228
Setup DNS and DHCP Services on CONTROLLER-01	193
Setup DNS and DHCP Services on CONTROLLER-02	188
Setup iSCSI Target on STORAGE	57
Setup MPIO on PRODUCTION-01	15
Setup MPIO on PRODUCTION-02	15
Setup iSCSI Initiator and Hyper-V on PRODUCTION-01	113
Setup iSCSI Initiator and Hyper-V on PRODUCTION-02	88
Setup Failover Cluster (PRODUCTION cluster)	81
Setup Active Directory	145
Setup Group Policy	303
Total	2,419

Table 1: Summary of manual build procedure

A parts-explosion is a convenient metaphor to view the results of Table 1 whereby each activity encapsulates a certain number of steps to complete and can be measured in terms of fan-out. In this particular instance, the average fan-out is 1:121 with a maximum of 1:303 and a minimum of 1:14. To put this into perspective, consider the following analogy: If the computing infrastructure represents an automobile, and the functional components such as domain and internet protocol address services, virtualization platform, and shared storage correspond to the automobiles drivetrain, suspension, and chassis, then acquiring an automobile would be like receiving a do-it-yourself kit that must be put together bolt-by-bolt.

A relatively flat hierarchy with a high fan-out ratio contributes to steep learning curves and the difficulties of delegating support responsibilities. In this situation, an individual unfamiliar with the system would have to perform an average of 121 steps to complete an activity.

4 Automated Configuration

In many ways, system configuration is well suited for automation. Duplicate settings made across computers, repeatable procedures that tailor computer hardware, and derived settings can often be directly managed through automated systems. Using software to automate a process has additional benefits, such as capturing and embedding tacit knowledge, self-documentation, incorporating best practices, default and custom setting management, and pre-deployment testing. In addition, automation software can reduce the fan-out of hierarchy by encapsulating manual steps. Other design concepts important in the development of an automated system include: a common interface, scalable configuration files, the ability to manage default and custom settings, and a flexible scripting method.

4.1 Configuration File

The configuration file contains all the settings and sequence of tasks needed to configure the system. XML was selected as the file format due to its flexibility and programming tools available to read and manipulate contents. A key consideration was the ability to present the data in a hierarchical structure, yet access the data in practically any manner deemed suitable to the task (i.e. sequential or random access, collections, etc.).

The file consists of two sections. First, the computer section identifies each computer in the system and serves as container element for individual service settings. For example, the computer named `CONTROLLER-01` contains DNS and DHCP settings since hosts these two servers in addition to being a domain controller. The type of service determines whether settings depend on sequence. Second, the task section enumerates the activities needed to configure the system. Tasks may include global settings along with settings declared within each computer element and are executed in sequence.

The sample configuration file is shown in Figure 2 and illustrates the hierarchal structure of the data. The file contains five *Computer* elements and a *Tasks* element. The computer element named `CONTROLLER-01` is expanded to show the *DnsServer* and *DhcpServer* elements that contain computer specific settings drawn upon by the *SetupDnsService* and *SetupDhcpService* elements in the tasks section. The file structure is particularly flexible and can be scaled across multiple files.

A special form of the configuration is used to set the operating system properties and configure computer hardware. In this case, only one computer element is defined and settings are executed in sequential order except installing Windows Features, running Windows Update, adding the computer to a domain or workgroup or renaming it, and restarting the computer.

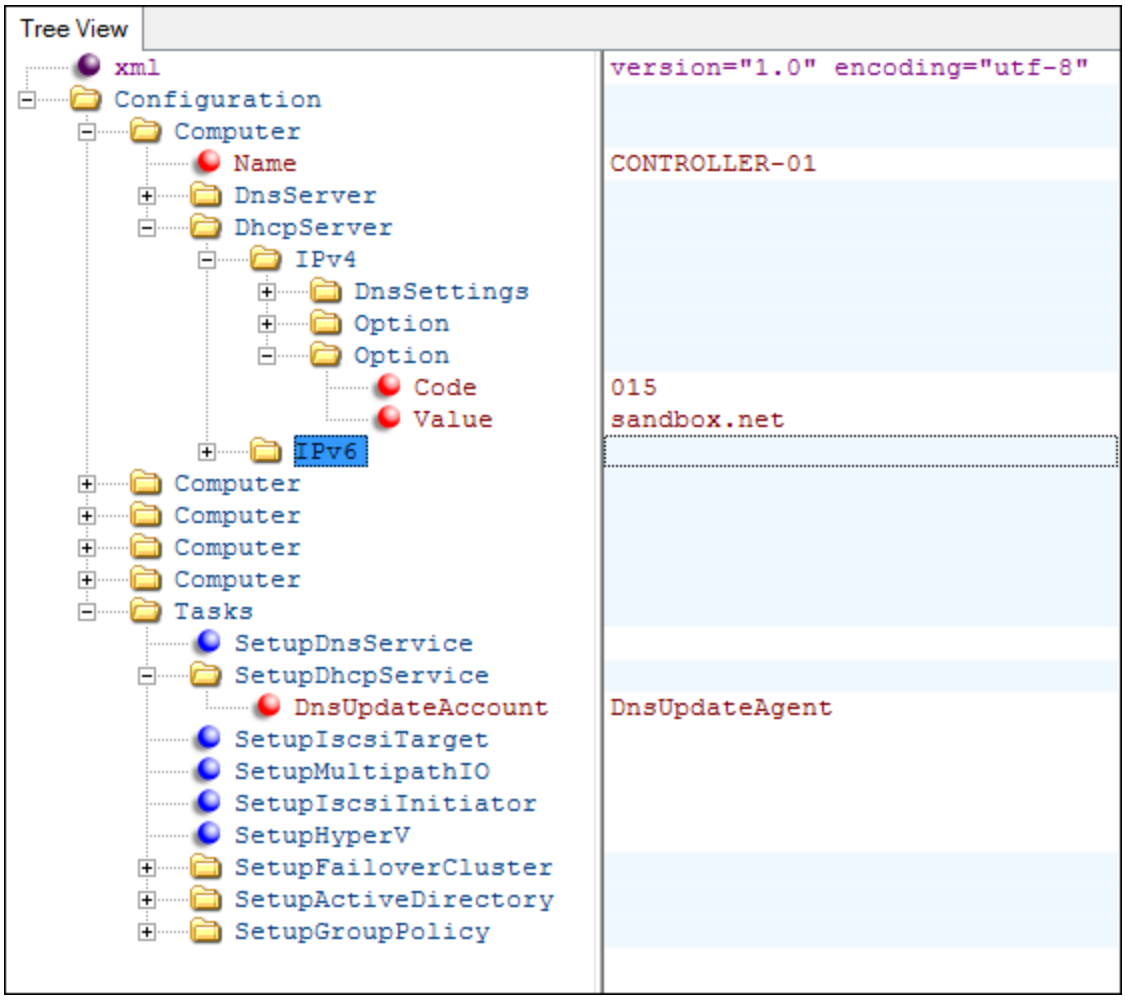

Figure 2: Hierarchical structure of sample configuration file

4.2 Application

The prototype application used to automate system configuration consists of a collection of PowerShell scripted commands called cmdlets. Unlike a traditional application where all the functionality is embedded in a single executable, cmdlets perform particular tasks and can be combined in various ways to form a complete process using pipelining.

Microsoft PowerShell is designed to administer computers running the Windows operating systems and provides a consistent scripting environment built on the .NET Framework. Version 3.0 was released with Windows Server 2012 and greatly expands the number of cmdlets available, replacing many of the legacy commands often used to administer systems. Organizing the problem space as a set of tasks allowed the application to be created in a highly modular fashion where new components could be added with ease.

The principle tool in the application is the *Write-Script* cmdlet. The purpose of the cmdlet is to read the configuration file and create a script object that includes all the statements needed to make the required settings along with comments to enhance readability. The script object can be saved, displayed, or pipelined to another cmdlet. Several options are available for subsequent pipeline stages including *Invoke-Expression*, *Invoke-Script*, and *Format-Script*.

Invoke-Expression is a native PowerShell cmdlet that executes a string expression directly. Although very effective at running a script, control does not return until the script completes. Invoke-Script is a cmdlet included in the application that allows more control over the execution of the script, provides visual progress feedback, and incorporates several debugging features to manage exceptions should they occur. Format-Script is a cmdlet that generates the text representation of the script and is useful for reviewing script contents or screening the script prior to execution.

Consider the following example. Figure 3 lists the configuration file for DNS services on `CONTROLLER-01` and `CONTROLLER-02`. The *DnsServer* element defines individual computer settings while the *SetupDnsService* element indicates the task to perform.

SetupDnsService.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>

    <Computer Name="CONTROLLER-01">
        <DnsServer>
            <NetworkAdapter Name="LE">
                <NameServer>172.16.3.11</NameServer>
                <NameServer>127.0.0.1</NameServer>

                <NameServer>fd8d:4989:2b1a:3::11</NameServer>
                <NameServer>::1</NameServer>
            </NetworkAdapter>

            <Zone Name="3.16.172.in-addr.arpa" />
            <Zone Name="3.0.0.0.a.1.b.2.9.8.9.4.d.8.d.f.ip6.arpa" />
            <Scavenging ApplyOnAllZones="*" />
        </DnsServer>
    </Computer>

    <Computer Name="CONTROLLER-02">
        <DnsServer>
            <Scavenging ApplyOnAllZones="*" />
        </DnsServer>
    </Computer>

    <Tasks>
        <SetupDnsService />
    </Tasks>

</Configuration>
```

Figure 3: DNS service configuration

Figure 4 shows the command line statement (highlighted) used to prepare the script. In this case, the *Write-Script* cmdlet reads the contents of the configuration file *SetupDnsService.xml* and creates a script object. The script object is then piped to the *Format-Script* cmdlet that formats and renders script statements to the console.

```
PS> Write-Script SetupDnsService.xml | Format-Script

Import-Module 'SystemConfiguration'

# DNS Service Setup
# #################

# Confirming Windows feature installation
Confirm-WindowsFeature DNS -ComputerName 'CONTROLLER-01', 'CONTROLLER-02'

# Configuring server 'CONTROLLER-01'
Invoke-Command 'CONTROLLER-01' {
    Get-NetAdapter 'LE' | Set-DnsClientServerAddress `
        -ServerAddresses 172.16.3.11, 127.0.0.1, fd8d:4989:2b1a:3::11, ::1
    Set-DnsServerScavenging -ScavengingInterval 7.00:00:00 -ScavengingState $true `
        -ApplyOnAllZones:$true
    Add-DnsServerPrimaryZone -Name '3.16.172.in-addr.arpa' -ReplicationScope Domain
    Add-DnsServerPrimaryZone -Name '3.0.0.0.a.1.b.2.9.8.9.4.d.8.d.f.ip6.arpa' `
        -ReplicationScope Domain
}

# Configuring server 'CONTROLLER-02'
Invoke-Command 'CONTROLLER-02' {
    Set-DnsServerScavenging -ScavengingInterval 7.00:00:00 -ScavengingState $true `
        -ApplyOnAllZones:$true
}
```

Figure 4: DNS service script

There are a few points worth noting about the script. First, the script is quite readable. Comments are created by the DNS service configurator and inserted automatically. Also, PowerShell cmdlets and associated parameters can be rather wordy thus taking longer to enter manually. However, the wordiness tends to enhance the scripts readability when generated by the configurator. Second, the script serves as a configuration management document recording the settings made. Third, the script employs default parameters as needed (see the argument used for the scavenging interval) thus reducing the number of settings that need to be entered in the configuration file. Fourth, the script provides a starting point if problems arise or changes are needed at a future date offering support personnel the ability to drill-down into the details of the specific configuration.

4.3 Results of an Automated Build

The automated build parallels the manual setup described in the previous section. Windows Server 2012 Enterprise was again used as the base operating system, however, both domain controllers and the storage server employed the core version and the two production computers used Windows Hyper-V Server 2102 (core version, virtualization

platform only). Other than the default interface and availability of certain roles and features in Hyper-V Server, there is no difference in functionality.

Tracking the automated process is fundamentally different given the implementation. In its most compact form, the entire build can be achieved with less than two dozen commands of which ten are used to setup the automation environment upon startup. Of the remaining statements, two are used to promote the domain controllers and nine write and execute the configuration scripts. As an alternative, Table 2 itemizes the number of script statements generated to accomplish the activity.

Activity	Stmts
Configure computer CONTROLLER-01	14
Promote CONTROLLER-01 to domain controller in new Active Directory forest	1
Setup core domain settings	7
Setup core domain administrators	5
Configure computer CONTROLLER-02	14
Promote CONTROLLER-02 to domain controller in existing forest	1
Configure computer CONSOLE	14
Configure computer STORAGE	38
Configure computer PRODUCTION-01	53
Configure computer PRODUCTION-02	53
Setup DNS Service	5
Setup DHCP Service	23
Setup iSCSI Target	11
Setup MPIO	10
Setup iSCSI Initiator	16
Setup Hyper-V	2
Setup Failover Cluster (PRODUCTION cluster)	7
Setup Active Directory	21
Setup Group Policy	28
Total	323

Table 2: Summary of automated build procedure

Using the automated process, the entire build was completed in one hour. The fan-out at the system-level is 1:22 corresponding to each statement executed. The average statement fan-out per activity is 1:17 with a maximum of 1:53 and a minimum of 1:2. The significant reduction in fan-out ratios attests to the efficiency scripted commands can attain by eliminating the need to navigate various user interfaces.

Although automated and manual processes achieve the same objective, a direct comparison is difficult since the approaches have significant differences. Factors like the need to prepare and debug a configuration file or the overhead associated with embedding tacit knowledge in code are not included in the immediate results. Nonetheless, the ability to configure a system with two dozen commands is considerably more appealing than manually going through 2,400 steps to achieve the same outcome.

5 Summary and Future Work

Automation brings a number of benefits to system configuration such as repeatability, consistency, and the ability to optimize a process. However, the key advantage is the ability to manage the hierarchy of subsystems, components, and settings much more effectively. Individuals can concentrate on system configuration without having to navigate various interfaces, search for suitable commands, or decipher cryptic syntaxes. This eases the learning curve and provides a structure for support personnel in academic organizations that experience high turnover or constrained resources.

This paper identifies a current challenge system administrators face deploying computing infrastructures and proposes an approach that employs automation to manage the system configuration. The prototype application presented allows settings to be organized in a systematic manner and deployed efficiently. Restructuring the problem opens new opportunities to create tools to streamline the process. Potential areas to pursue include deriving settings from known dependencies, creating an interface that allows users to visualize subsystems and associated settings, and incorporating best practices logic and pre-deployment testing.