28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Test Environment Configuration in a Complex World

Authors
Liu Hong liuhong@microsoft.com
Edward French edfrench@microsoft.com
Maxim Markin maxim@microsoft.com

Abstract

Imagine you need to reproduce a complex test environment configuration that consists of Windows Server and Client machines and requires several Active Directory Domains, DNS and/or DHCP roles installed on some of the server machines. How will you implement this in a way that is robust across all available versions, editions and languages of Windows Server? With the invention of PowerShell, Windows got command line usability and tools for local and remote management of Windows Server that allows tackling these kinds of problems.
This paper offers an approach that allows capturing and reproducing a complex test configuration in an Xml file and executing a test scenario configuration using PowerShell scripting and PowerShell remoting. This approach may be used to test applications, networking components, or products which interact or run under Microsoft Windows.
In this paper we will examine the problems associated with the current approach, show the new approach, then show how the new approach is more flexible, faster to develop new test scenarios, and more robust.

Biography

Liu Hong is a Lead SDET in Microsoft Platform and Tools division at Microsoft Cooperation. She has worked at Microsoft since May 2006. She holds a Ph.D. degree in Civil Engineering from Clemson University. Before her career at Microsoft, she first worked at Aon Cooperation as a research engineer and afterwards as software developers in various companies, such as Infomove and Applied Inference in Puget Sound areas, and immediately before she joined Microsoft, as an independent contractor for Gray Hills Solutions.

Edward French have worked at Microsoft Corporation for 10 plus years as a Software Developer in Test. Hi is in the Windows Server division testing management products. He has one US patent for software development.

Maxim Markin is a Software Design Engineer in Test in Microsoft. He enjoys working together with Liu and Ed.

1. Introduction

In today's world, system administrators are required to manage more and more servers/machines that are located in a more and more complex environment, various domains and workgroups, locally and remotely. Server management products are therefore required to provide the ability for administrators to deal with the challenges they are facing. While products are made to enable the capability of managing many remote servers and machines, it also brings a new set of challenges for us as software quality assurance engineers to test these products.

One of the most important aspects of the challenge is to capture and reproduce the test environment setup and configuration the test environment in order to execute our tests. The test environment we are configuring consists of multiple computers running Microsoft Windows and the network connecting them. Configuring the networking components like switches and routers is not in the scope of this approach. The actual physical machines may be located locally or remotely and may be physical or virtual machines. They may reside in different domains and workgroups. To properly set up such an environment is a challenging task.

Microsoft Baseline Configuration Analyzer (MBCA) is a good example of multi-machine management application. MBCA is a product which helps maintain optimal system configuration by analyzing the configuration of a computer or set of computers, that are located in a distributed multi-machine environment, against a predefined set of best practices, and then reporting results of the analysis. For MBCA testing, there are five major scenarios that we focused on. They are:

1. MBCA host machine and a remote target machine
2. MBCA host machine and a number of remote target machines
3. MBCA host machine, a remote target machine, and a 2nd hop machine
4. MBCA host machine, a remote target machine, and a remote target machine
5. MBCA host machine, a remote host and remote targets, and 2nd hop machine

For MBCA scenarios, we have basically four major roles, MBCA host machine, remote host, remote target and 2nd hop machine. MBCA host machine hosts the MBCA application. Remote host is needed in case of using MBCA remoting feature to remote to the remote target. Remote target is the machine in which the configuration is to be scanned. The 2nd hop machines are the machines that remote target need to gather part of configuration data from.

Take scenario 5 as an example. Two domains and a workgroup are involved in this topology. Each domain has a single domain controller (DC) and two joined workstations which may have some Server Manager roles installed. Other two machines are joined to a workgroup and also may have some Server Manager server roles installed. The follow diagram captures scenario 5. The MBCA host machine, remote host, remote targets, and the 2nd hop machine may be any of the machines in the diagram.

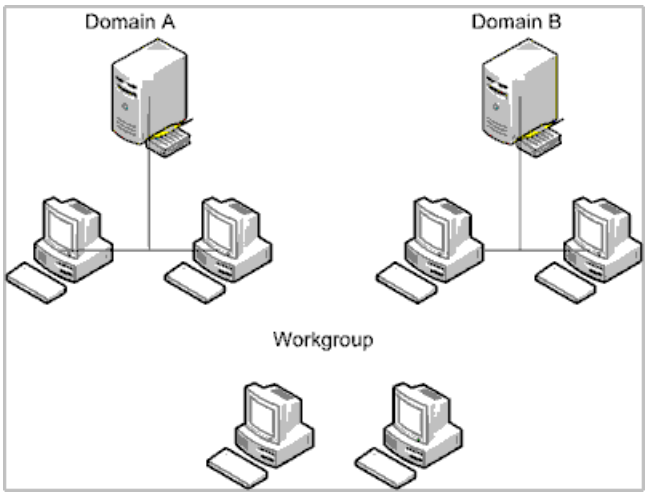

Figure 1 Sample scenario topology

In order to set up this environment, we need to accomplish the following major tasks:

1. Create two domains with one domain controller each.
2. Add two machines to each domain.
3. Create a trust-relationship between the two domains.
4. Add two machines to a workgroup.
5. Enable remoting on all machines.
6. Install MBCA and PowerShell.
7. Install server roles/components on remote machines.

Now we will discuss what we need to do to configure the test environment.

2. What We Used to Do

In single machine testing environments, we could configure a static topology, and then join the machine to be setup for testing to various places in the topology. Even if the topology is complex, the application under testing was only concerned with the single machine. Multiple individual machines could be set up quickly in this static environment.

For the challenges we are facing, we are managing multiple machines configuration where the application under test interacts with applications on other machines in the topology. This means a static set of machines can't be used for testing because each machine in the topology may be changed by the application under test. Traditionally, test machines were configured using a combination of scripting languages, such as VBScript and command line commands. For each step, we created a corresponding script or command. We then wrapped each script or command as a "job". We used these jobs in our test execution environment. For simple test execution, such as single server management, this approach worked fairly well as it was simple and straight forward.

But for the complex test setup which involves multi-machines, with the increase in the number of machines and the complexity in configuration, using this approach for configuration became cumbersome and very time consuming. In our recent MBCA testing, to cover the scenarios we talked about previously, we ended up with close to 100 "jobs". Before test automation execution, all jobs were tested separately as a first step. For each topology, these jobs were then combined together as a workflow for execution of test automation. The integration testing of this workflow creates another challenge. If the workflow failed, lots of trial and error was needed to determine the failure point because of the lack of debug ability of the scripts and commands. Configuring and debugging consumed lots of time and efforts. Needless to say we required many workflows for different scenarios. All these workflows had to be tested thoroughly for our test execution. The cost just increased linearly.

As a result, we found this approach does not scale for testing in complex test environment. Today's test environment setup is no longer single machine, but multi-machines with cross machine configuration requirements, such as MBCA testing. We need our system to scale for the new challenges we are having. Our mission is to find an approach that can reduce the cost of creating, managing and maintaining test execution, and also improve the debugging experience of configuration tasks.

3. New Approach – Integrated solution of XML and PowerShell

The new approach is to capture the test environment in a set of library files, then recreate the environment using Windows PowerShell. This new approach is based on two key technologies, XML and Windows PowerShell. XML is used to describe the configuration/execution requirements, and orchestrate automation execution. PowerShell scripts are used to parse the XML, configure, and set up the test environment.

Windows PowerShell is a command-line shell and scripting language that is designed for system administration and automation. Built on the Microsoft .NET Framework, Windows PowerShell enables IT professionals and developers to control and automate the administration of Windows and applications. Some of its 2.0 features, such as Remoting, Integrated Scripting Environment, and Scripting Debugging make it the ideal technology for the implementation of our approach to configure complex melt-machine test execution environment.

This new approach enables us to configure the test environment for cross-machine testing more efficiently. It provides a general approach of implementing multi-machine configuration in test lab and provides a common vocabulary for exchanging test configurations with external teams.

Below is an example of a PowerShell script that connects to specified machines and creates an object to be consumed by other script methods. This script gathers machine configurations by querying WMI providers.

```
function Discover-Machines ($machineNames){
    # Create script block to be executed on remote machines. The
script block will return
    # an object containing OS settings
    $scriptBlock = {
        $win32Obj = Get-WmiObject Win32_OperatingSystem
        $arch = $env:PROCESSOR_ARCHITECTURE
        Add-Member -InputObject $win32Obj -MemberType NoteProperty -
Name "Architecture" -Value $arch -Force

        $currentCulture =
[System.Globalization.CultureInfo]::CurrentUICulture.Name
        Add-Member -InputObject $win32Obj -MemberType NoteProperty -
Name "CurrentCulture" -Value $currentCulture -Force
        return $win32Obj
    }
    # Initialize the variable to hold the results as an array
    $results = @()
    # Iterate over the machines and populate the results variable
    foreach($machine in $machineNames)  {
        $session = New-PSSession -computername $machine -cred $PSCred
        if (!$?) {
```

```
            throw "Failed to connect to some of the machines.
"+$error[0]
        }
        $result = icm -Session $session -ScriptBlock $scriptBlock
        $results += $result
        Remove-PSSession -Session $session

$result.BuildNumber,$result.ProductType,$result.OperatingSystemSKU,$r
esult.OSLanguage,$result.Architecture)
    }
    $results
}
```

Figure 2 Sample PowerShell script

3.1 Configuration file

As previously mentioned, we capture test configurations in a configuration file in XML format. The configuration file can contain several test configurations or test scenarios. Each test configuration completely describes test scenario by specifying required test resources and scripts/tasks to be executed on each resource. Test resource is a machine described by number parameters: Operating system platform, edition (SKU), language, platform architecture (amd64/x86/ia64), virtual/physical machine. Then the configuration file references scripts that need to be executed on each resource. Scripts are grouped in logical units which are called test roles. A single test role may be applied to a number of resources. A test role may also have a number of parameters specified in the configuration script.

Below is the sample scenario configuration file.

```
  <Scenario Name="TwoMachineTestDemo">
    <Roles>
      <Role Name="DomainController_PDC">
        <Tasks>
          <Task Name="taskPromoteDomain"/>
        </Tasks>
      </Role>
      <Role Name="Domain1Joined">
        <Tasks>
          <Task Name="taskJoinDomain">
            <Parameter
Name="domain">"D"+$script:Machines.Domain1_DomainController</Parameter>
            <Parameter Name="user">$global:LogonUser</Parameter>
            <Parameter Name="password">$global:LogonUserPassword</Parameter>
          </Task>
        </Tasks>
      </Role>
    </Roles>
    <TestSuites>
      <TestSuite Name ="Test1" Harness="TestFramework.exe">
        <Parameter Name="-AnalyzerAssembly">'Tests.dll'</Parameter>
      </TestSuite>
    </TestSuites>
    <Machines>
      <Machine Name="Domain1_DomainController" Platform="WS08-R2"
SKU="Enterprise Server Core Edition" Language="en-US" Architecture="amd64"
ComputerName="">
```

```
        <SetupRole Name="DomainController_PDC" Status=""/>
        <TestSuite Name ="Test1" Order="0"/>
      </Machine>
      <Machine Name="Domain1_MemberServer_1" Platform="W2K3" SKU="Server
Edition" Language="en-US" Architecture="x86" ComputerName="">
        <SetupRole Name="Domain1Joined" Status=""/>
      </Machine>
    </Machines>
  </Scenario>
```

Figure 3 Sample scenario configuration file

The file captures two machine test scenario: one is Domain Controller while another is joined to that domain. Also it defines roles properties of machines that are required of each test roles. And finally it defines which tests need to be run on each resource.

3.2 Single entry point to execute scenarios.

We have a single script to drive overall execution. This script calls helper functions from other scripts. We group helper functions by library scripts, for example we have libMachineSettings.ps1 and libUtilities.ps1 scripts. Library scripts are referenced by the main script or dot-loaded in terms of PowerShell. The main script has six parameters: type of action to perform, list of machines, scenario configuration file name, scenario name, and two parameters used by an internal reporting tool. Below is the execution workflow:

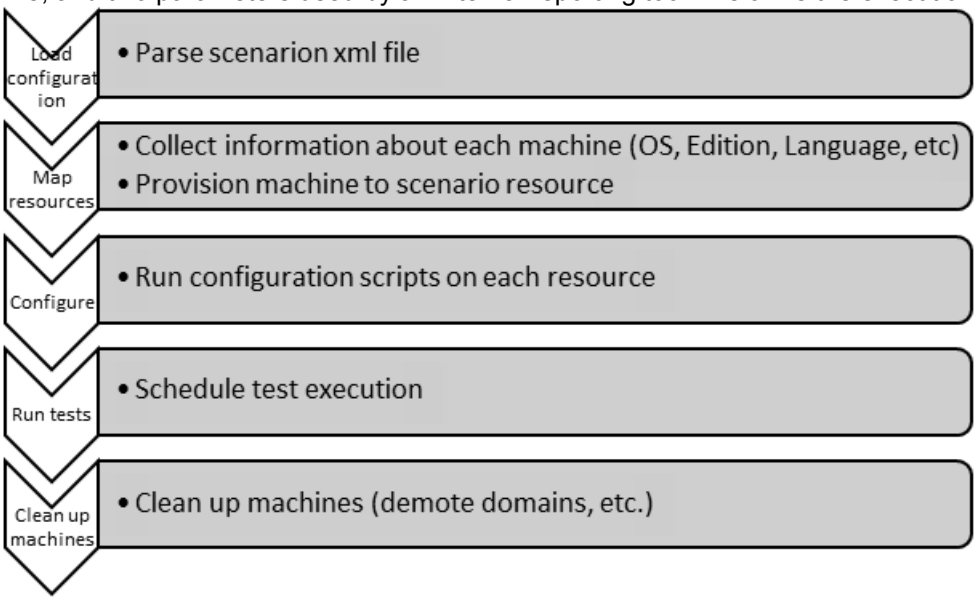

Figure 4 Scenario configuration workflow

3.3 PowerShell remoting is used to configure remote machines

PowerShell Remoting is used to control script execution on remote machines. We store credentials required to connect to remote machines in central repository.

3.4 Solution Structure

Below is the architecture layout of the solution. The solution is built on top of Windows Management Framework that includes PowerShell, Bits and WinRM. The script library of common management tasks contain library of common functions to configure a machine which includes function to join to a domain,

turn on the firewall, change IP address and etc. The top level has test roles scripts which can be applied to a test machine. By combining different test roles we get the test scenario to be executed.

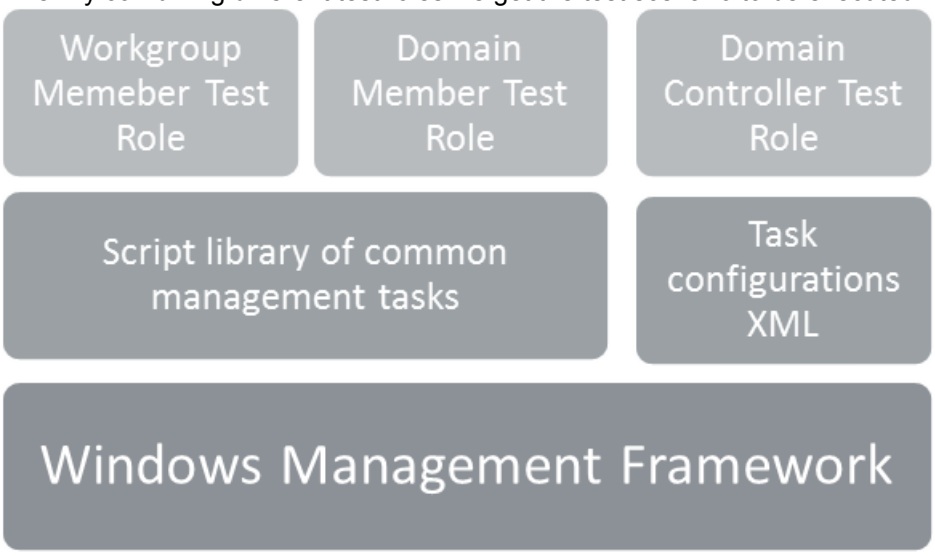

Figure 5 Architecture layout

4. Our Findings

Based on MBCA multi-machine testing, we did a comparison of test cost using the existing approach versus the new approach. As we discussed in the previous section, we have 5 different topologies for MBCA multi-machine test execution. With the assumptions that we have the basic infrastructure in place and relative familiarity with PowerShell programming, our initial finding suggested about 65% percent reduction in costing to set up and execution. The following table lists the detailed data for two approaches:

Work Items	Description	Old Approach (days)	New Approach (days)
VBScript and Command line batch files	Create, test, and debug individual task scripts	26 days (70 jobs)	N/A
PowerShell Script	Create, test, and debug individual task PowerShell scripts	N/A	10.5 (7 scripts)
Master script execution	Create, test, debug the master scripts for each scenario.	20 days (5 scenarios)	0
Write configuration file	Create xml configuration file	N/A	2
Configuration challenges	Test and debug each configuration	10	10

Process overhead	The triaging, bug fixes, communication overhead involved to get a failure in test configuration issue fixed, new test cases added to execution, fixing the bug itself.	20	5
Total		76	27.5

Figure 6 Comparison of time spend on testing

For our future Microsoft Windows testing, with the increased number of machines included in testing and configuration, we will have more topologies and more complex environment. We expect saving will be even greater.

4.1 Summary of Benefits for the New Approach

In contrast to the existing way of test environment configuration and execution, this new approach we proposed here adopted two key technologies, which when combined, bring together many benefits for efficient multi-machine test environment configuration and execution.

First, the requirements are captured in structured storage in XML document. This provides a structured and consistent way of describing configuration and execution requirements. With the help of the schema, we get the validity and integrity of the data stored in the configuration file verified. This reduces lost time when invalid entries, such as spelling errors, are made.

Secondly, PowerShell is used in parsing and programming the requirements. The advantages of using PowerShell in the new approach are:

1. Ease of coding.
2. Remote access is built-in into PowerShell: PS remoting allows remote computers to be easily managed through a SOAP-based web service
3. User experience: the user experience is the same whether you are connected to local or remote host
4. Parallel execution in PowerShell: execution of a script on remote machine does not necessarily block local execution
5. Open protocols: WinRM, the underlying protocol used by PowerShell remoting, is based on WS-Management open standard

In summary, our new test machine configuration approach provides us with the following benefits:

1. Rich debugging support of PowerShell
2. Script storage in source control system
3. Simplified maintenance of configuration scripts
4. Simplify workflow generation: New scenario can be created by generating new XML section, which describes the scenario and its configuration requirements, and combining existing scripts (tasks).
5. The scripts easily can be shared between other test teams.

4.2 Adapting to other test environments

The key to this approach is storing the configuration information in xml files. These xml files may be accessed by other technologies, but we feel PowerShell gives us the easiest tools to work with both the xml files and executing the specialized tasks required. While we're focused on creating environments that revolve around domain controllers, workstations, and network connectivity, the techniques provided by this approach are just as applicable to configuring a single machine environment to test either hardware or software.

The first step in using this approach is to model the required environment in the xml configuration file. We do this by defining a role with properties that define the tasks required to create and configure that role. We next create PowerShell scripts to execute the tasks. These individual task scripts are then executed in a master script which controls the order and on which machines the tasks are executed.

One simple pilot program would be to install an application on several machines, logon to each machine with different user credentials, then execute a set of tests on each machine. In this case, the tasks to be done might include running setup.exe for application, setting some registry values for the application, and then setting auto-logon with the user's credentials. These tasks would be added to each role definition in the configuration xml and a role would be assigned to each machine.

5. Conclusion

As we are dealing with more and more complex multi-machine test environments, the traditional approach of using scripts and commands to configure this environment will not scale. In our new approach, we capture the configuration and execution requirements in an easy to modify configuration file. With the help of PowerShell, this approach becomes a very effective way of configuring complex system and executing tests. PowerShell provides ease of implementation, ease of maintaining and ease of debugging. It provides a good solution to many of the challenges we are facing in order to configure and execute testing for multi-machine scenario effectively.

Glossary

Name	Description	Acronym
Microsoft Baseline Configuration Analyzer	Microsoft application which helps maintain optimal system configuration by analyzing the configuration of a computer or set of computers	MBCA
Active Directory	Active Directory directory service provides the means to manage the identities and relationships that make up network environments	AD
Active Domain Controller	A Windows server that maintains user database and manages security in Active Directory	DC
DNS role	Microsoft Windows role	DNS
DHCP role	Microsoft windows role	DHCP
Server Manager	Windows component for managing servers	
Windows Management Infrastructure	Infrastructure for management data and operations on Windows-based operating systems	WMI

References

Payette, B 2007. Windows PowerShell in Action

Microsoft Technet. Getting Started With Windows PowerShell
http://technet.microsoft.com/en-us/library/ee177003.aspx

Microsoft KB. Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0)
http://support.microsoft.com/kb/968929

Microsoft downloads. Microsoft Baseline Configuration Analyzer (MBCA):
http://www.microsoft.com/downloads/details.aspx?familyid=DB70824D-ABAE-4A92-9AA2-1F43C0FA49B3&displaylang=en