

Test Environment Configuration in a Complex World

Maxim Markins, Edward French, Liu Hong,
Muhammad Usman
Microsoft Corporation
October, 2010

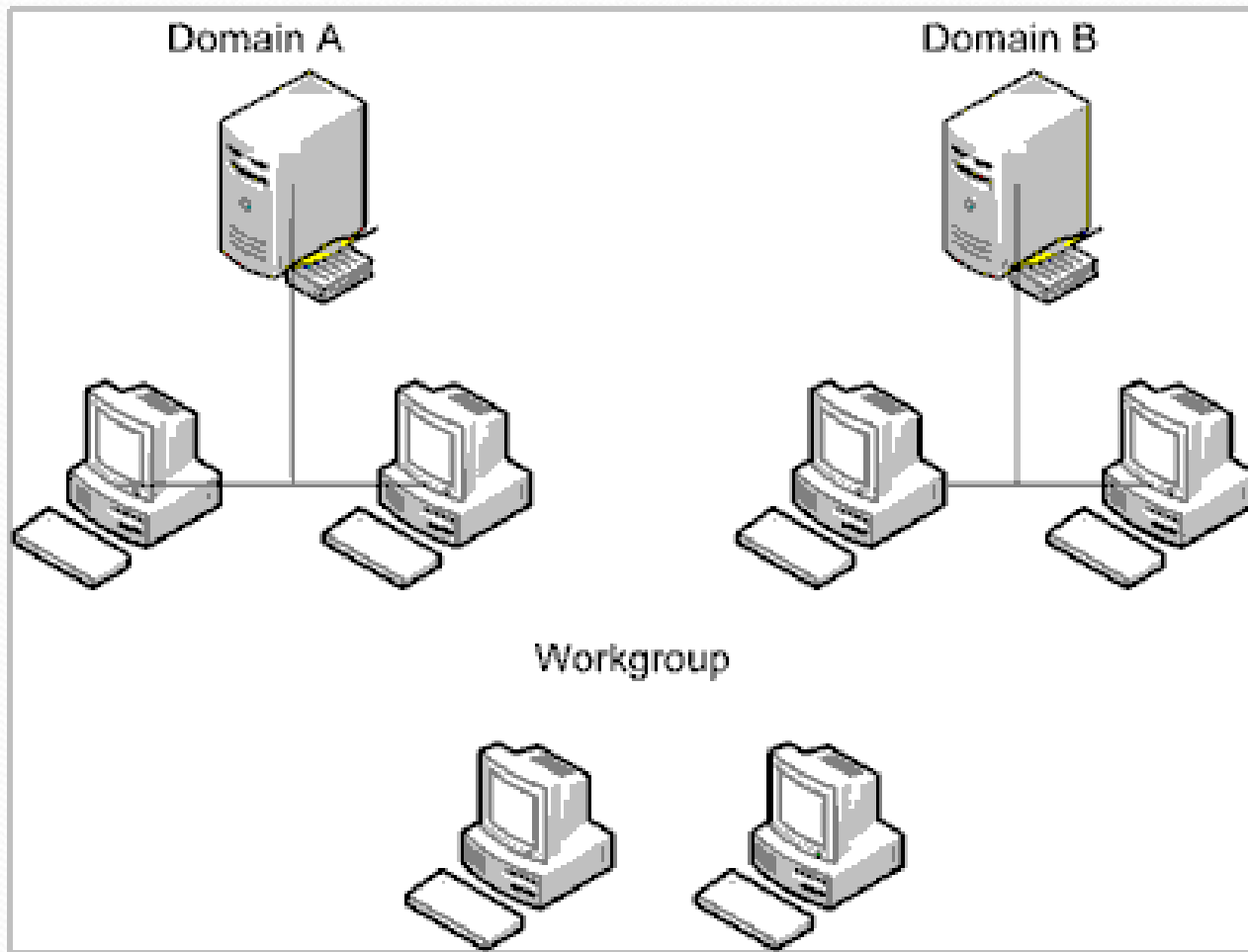
Challenges

- IT Pros need to manage many machines
- Server management products need to target multiple machines
- We need to configure complex test environment on a daily basis

A Testing Example

- Microsoft Baseline Configuration Analyzer (MBCA)
 - Scan configuration of local or remote systems
 - Analyze the configuration against predefined rules
 - Help maintain optimal system configurations
- Some test scenarios
 - MBCA host and a number of remote target machines
 - MBCA host, a remote host and remote targets, and 2nd hop machines

Testing Configuration



Configuration Requirements

- Machine requirements
 - MBCA host machine
 - Remote host
 - Remote target(s)
 - 2nd hop machine(s)
- Test Setup
 - Domain controller promotion
 - Join machine to domain
 - Enable remoting
 - Etc.

Old Approach

- Implementation and execution
 - Used VBScript or command line commands for test setups
 - Wrapped each script or command as a “job”
- Is hard to debug and maintain
- Will not scale in multiple machine testing
- Became very complex for testing against multiple machines
 - Over 100 “jobs” for MBCA 2.0 scenarios

New Approach

- Based on XML and Windows PowerShell 2.0
- XML used to specify configuration requirements
- PowerShell used to parse requirements, test setup and execute test
- PowerShell script drives the whole testing workflow

Solution Architecture

Workgroup
Member Test
Role

Domain
Member Test
Role

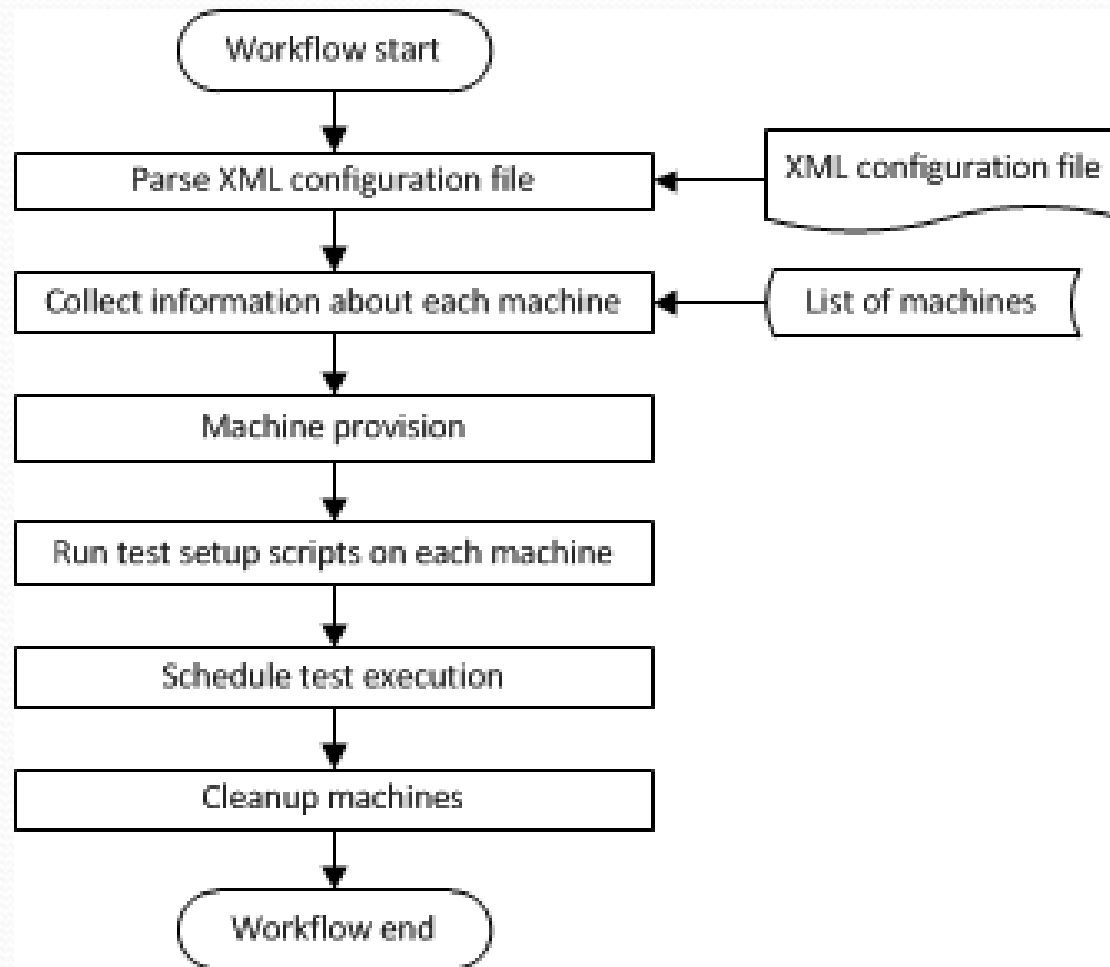
Domain
Controller Test
Role

Script library of common
management tasks

Task
configurations
XML

Windows Management Framework

Testing Workflow



Sample 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>
  <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>
  <TestSuites>
    <TestSuite Name ="Test1" Harness="TestFramework.exe"><Parameter Name="-AnalyzerAssembly">
      'Tests.dll'</Parameter></TestSuite>
  </TestSuites>
</Scenario>
```

Sample Script – Discover Machines

```
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,$result.OperatingSystemSKU,$result.OSLanguage,$result.Architecture)
    }
    $results
}
```

Sample Script - DCPromo

```
function Promote-Domain(  
    [int] $DomainLevel = $global:DomainLevel,  
    [int] $ForestLevel = $global:ForestLevel,  
    [string] $NewDomain = $global:NewDomain,  
    [string] $DomainNetBiosName = $global:DomainNetBiosName,  
    [string] $NewDomainDNSName = $global:NewDomainDNSName,  
    [string] $RebootOnCompletion = $global:RebootOnCompletion,  
    [string] $ReplicaOrNewDomain = $global:ReplicaOrNewDomain,  
    [string] $InstallDNS = $global:InstallDNS,  
    [string] $ConfirmGc = $global:ConfirmGc,  
    [string] $CreateDNSDelegation = $global:CreateDNSDelegation,  
    [string] $Password = $global:Password,  
    [string] $SafeModeAdminPassword = $global:SafeModeAdminPassword,  
    [string] $LogonUser = $global:LogonUser,  
    [string] $LogonUserPassword = $global:LogonUserPassword)  
{  
    # Set autologon to domain builtin Administrator  
    AutoLogon-UserName $global:LogonUser $global:DomainNetBiosName $global:LogonUserPassword 9999  
    LogStringToFile "Running Promote-Domain in $global:TestAutomationPath"  
    $CmdLine = "-Unattend -DomainLevel:" + $DomainLevel + " -ForestLevel:" + $ForestLevel + " -NewDomain:" + $NewDomain + " -  
        DomainNetBiosName:" + $DomainNetBiosName + " -NewDomainDNSName:" + $NewDomainDNSName + " -Password:" +  
        $Password + " -SafeModeAdminPassword:" + $SafeModeAdminPassword + " -RebootOnCompletion:" + $RebootOnCompletion + "  
        -ReplicaOrNewDomain:" + $ReplicaOrNewDomain + " -InstallDNS:" + $InstallDNS + " -ConfirmGc:" + $ConfirmGc + " -  
        CreateDNSDelegation:" + $CreateDNSDelegation
```

Sample Script – DCPromo (continued)

```
LogStringToFile $CmdLine
```

```
$Result = & dcpromo.exe -Unattend -DomainLevel:$DomainLevel -ForestLevel:$ForestLevel -NewDomain:$NewDomain -  
DomainNetBiosName:$DomainNetBiosName -NewDomainDNSName:$NewDomainDNSName -Password:$Password -  
SafeModeAdminPassword:$SafeModeAdminPassword -RebootOnCompletion:$RebootOnCompletion -  
ReplicaOrNewDomain:$ReplicaOrNewDomain -InstallDNS:$InstallDNS -ConfirmGc:$ConfirmGc -  
CreateDNSDelegation:$CreateDNSDelegation
```

```
$ExitCode = $LastExitCode
```

```
LogStringToFile "Message: $Result.\nLast exit code: $ExitCode" $ExitCode
```

```
if ($ExitCode -ne 2)  
{  
    return $global:Failed  
}  
else  
{  
    return $global:OK  
}  
}
```

Benefits

- Test configuration requirements easily captured in XML
- Lots of advantages of PowerShell:
 - Easy to implement
 - Remote access is built-in into PowerShell
 - Parallel execution in PowerShell
 - Free!
- Simplified adding new test scenarios
- Easy sharing of scripts with other test teams

Summary of Results

- Reduced cost in implementation
- Reduced cost in debugging
- Created easy to maintain scripts
- Enabled fast test configuration and setup
- We use in production!

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>
- About Windows Remote Management: [http://msdn.microsoft.com/en-use/library/aad84291\(VS.85\).aspx](http://msdn.microsoft.com/en-use/library/aad84291(VS.85).aspx)
- Windows Management Instrumentation: [http://msdn.microsoft.com/en-us/library/aa394582\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394582(VS.85).aspx)



Questions?