Hello Everyone, my name is Muhammad Usman Janjua and my two colleagues on stage are Vladimir Averkin (point) and Sunil Kutty (point). I’ve been asked to introduce this provisioning system solution to the audience today because I faced some unique challenges in the past year or so at work and this topology builder (aka machine factory) solution helped me a lot. So let me walk you through one of those tasks/challenges that I was faced and how this technology came to my rescue.
Last year, we were developing a multi-machine management tool which could target machines locally and remotely, monitor them, deploy/remove components from them, monitor events, services and apply best practice actions to name a few. After ironing out any functional kinks in the first couple milestones, here came the advanced/end-to-end testing phase. I was required to test this tool across a complex multi-domain test topology with varying trusts between domain controllers. I was told that we were to do this validation henceforth for every milestone. This figure presents an approximation of the scenario I was required to validate per milestone. (Explain the figure)
After describing the figure and all the challenges involved ......

So really what I needed was something (or someone) who would take care of setting this all up (correctly configured) and give the ready-made topology to me on which I could test the management tool. This process had to be robust and repeatable as per the requirement. And as we know, machines are best at churning out repeatable algorithmic tasks (avoiding human mistakes). In came the machine factory (topology builder) tool to my rescue.
Personal Experience:
Multi-domain test topology with varying trusts

Topology Builder provided:
1. A mechanism to crisply define my required topology
2. Ability to specify fine details (e.g., password/naming patterns)
3. A highly robust and repeatable process yielding exact output.
4. Productivity increase in orders of scale.
   - A manual error-prone setup of 2-3 days became a 2-hour deliverable
5. A risk free environment for rigorous testing.
   - Based on virtual machines, I could do destructive testing and still be able to recover the state
   - In worst case, an exact new setup was available within 2 hours
6. Independence from a central test environment
Typical application development cycle is represented with the following diagram. First, you implement a piece of functionality by writing some code, then you compile and build your application. Then, you deploy the application, make necessary configurations and do some testing. If some of the tests fail, you debug the problem, fix the code and the cycle repeats again.
Let’s focus on two stages of this cycle – Deploy and Configure. For a simple desktop application, test deployment and configuration is usually a simple and straightforward process. Sometimes, it can be skipped completely when you build and execute your tests in some integrated environment, for example, Visual Studio. For a more complex application they become an important part of the process. And even for a simple desktop application the need for deployment and configuration should not be underestimated.
Suppose you have a large number of tests that cover application functionality well. Is it enough to run those tests on one PC? It can be enough, but only if you plan to have one customer - yourself. And even then it may not be enough. Your PC can be patched, updated, new drivers installed, you can upgrade your OS, add memory, put new CPU. All this can cause your application to stop working. So, if you want to ship a reliable product that works robustly you need to test it with different hardware/software configurations.

Of course, you cannot possibly test all combinations, but you can still make a list of the most representative configurations to test and you will most certainly end up with the test matrix that requires to run tests on more than one computer, and more than one configuration.

If you have written an application that is executed on a single machine, you still need to make sure it
- Executes on different Windows OS versions
- Executes on different architectures, e.g., 32-bit 64-bit
- Scales well to multiple cores but still works on a single core
- Is able to work with minimum required memory and performs well with recommended memory settings
For a distributed application, doing testing on several machines is highly desirable, as not all functionality can be tested using single machine. An easy solution will be to deploy a static set of machines and update application files for each new build. This testing won’t give you full confidence in the quality of your tests since machines can accumulate configuration changes with time and running tests can produce false negatives or, more dangerously, false positives. False positive usually means that you have made some modifications to your test machine to enable certain scenarios and then you forget to take them back. For example you relax some security settings in your browser. Now your test runs fine, but in the production environment your application would fail. Also, the catch with using static machines is that your test code and, sometimes, even your product code has hard-coded dependencies on certain configuration settings, like computer name, IP, subnet, domain name, user credentials, etc. And you won't notice that in your test until you ship. That's why it is extremely important to run tests on dynamic environment, where the machines are newly deployed and configured for each test pass.
Let's recapture some of the main ideas that we have gone through here.

[Reading the slide] As we see it is always a good idea to do dynamic machine deployment and configuration for your testing. One can argue that does not have to be done for each test pass, and can only be done for the final integration pre-release testing. That is true, but for sometimes it is not optional. A good example is testing a component that is part of operating system. In that case, you need to build and deploy new version of OS each time you do the testing. That's where the need for repeatable automated machine deployment and configuration is fairly obvious. You are required to deploy a new build of OS on multiple machines every day, and if you won't automate it, you will probably end up spending 90% of your work time doing this manually.
Now that we have identified the need to redeploy new machines for certain test scenarios, let's see what our provision system which we call Machine Factory can do in that space.

Machine Factory deploys AND configures machines using the data files that we call topology definitions. Here is a typical example of topology definition file that we will use for our demo.

The foundation for the deployment is a virtual hard drive (or VHD as it is better known) which contains a sysprepped OS image. Part of this audience is probably acquainted with the process called sysprepping of the operating system. For the other part I will give a quick description of what sysprepping is. Imagine that you have live virtual machine that is running Windows operating system. There is a system tool that lives in c:\windows\System32\sysprep folder which is called sysprep.exe that you can run on your virtual machine, which will make you OS instance to become an OS template. It will remove the computer name, IP address and other properties from the registry that make this OS to belong to a specific machine instance. At the same time it will retain all data files, installed components, etc. on the disk. At the end of this process, sysprep will automatically shut down your computer and the VHD will become sysprepped. Now, if you start your VM again you will go into Out-Of-the-Box Experience (OOBE), which will prompt you for locale choice, administrator password. So, essentially you will have the OS setup experience, which will be much quicker.
than regular setup. No data files copying, unpacking, etc. You get your OS to set up in minutes. So sysprepping gives you 2 main benefits. One, the OS setup is faster. The other, you can get some of the stuff you need, pre-installed and pre-configured. Like, for example, Microsoft Office can be pre-installed on your OS. That's why enterprise admins love sysprepping so much.

So, ok, getting back to Machine Factory, it basically takes a sysprepped VHD and either makes a virtual machine out of it, using Hyper-V, or deploys physical machine, using VHD Boot. For simplicity, in our demo and examples we will show virtual machine creation, but keep in mind that almost exactly the same process happens for physical machine deployment.

Needless to say, there are numerous tools out there that are doing automated virtual machine deployment, thanks to a simplicity of Hyper-V API that can be used to build those tools. What makes Machine Factory more or less special compared to other similar tools is its ability to do automated configuration of provisioned machines as we can see from the following demo.
Topology Example

```xml
<xml version="1.0" encoding="utf-8">  
<Topology Name="DeploymentTest">    
<Description>A set of server SKUs for Server Manager Deployment testing.</Description>    
</Topology>    
</xml>
```

```xml
<xml version="1.0" encoding="utf-8">  
<Script>    
# Set global variables for configuration tasks.    
$global:fdn = 'mtest.microsoft.com'    
$global:domain = 'mtest'    
$global:user = 'user'    
$global:password = 'Admin01'  
</Script>    
<RunOnFirstLogon>    
<SetNetworkLocationPrivate/>    
<EnableTelnet/>    
<AddToDomainUserEntry DomainName="3fgdn" UserName="1domain\user" Password="$password"/>  
<AddToLocalAdministrators UserName="1domain\user"/>  
<StartLogonAndReboot DomainName="3domain" UserName="user" Password="$password"/>
</RunOnFirstLogon>  
</Topology>
```
Deployment Process
Lets give a quick overview of how automated configuration is done. We use three technologies here - Windows Unattended Setup, Autologon, and RunOnce. Windows Unattended setup allows you to specify the computer name and bypass the initial UI dialogs. It can also invoke commands during deployment and first logon phase. You can specify configuration tasks in unattend.xml in two ways. First is to use proprietary unattend.xml format. Second is to specify command or CMD scripts that you want to execute. We specifically made a choice to use the second one for all custom configurations but you can still use the first approach if you want by direct modification of the unattend.template.xml file that we use for all Machine Factory deployments. We decided to use commands, rather than unattend.xml settings for a number of reasons. First, it is infinitely easier to develop and test the configuration task that can be defined as command-line script. To test the script you just need to run it in CMD console on an already deployed computer and if it works there is a high chance it will work on an unattended deployment as well. Even if it does not work you can put any amount of logging to your script that will help you to troubleshoot. To test unattend.xml you need to actually deploy OS and then rely on a limited logging capabilities of Windows Unattended Setup if the configuration did not work as expected. Note that some tasks configuration tasks can be done before the first logon but some tasks require logon before proceeding, e.g., domain joining or adding user. To make those tasks run automatically we need two things. First, define those tasks in <FirstLogonCommands> section of unattend.xml. Second, define autologon parameters - user name and password as can be seen from the next slide.
Since all configurations tasks are based on scripts, configuration can be of any complexity and scale. If you know how to do your configuration from command-line, everything else becomes easy. Virtually anything that can be invoked from CMD console can become a configuration task. Let’s see how the process of creating a new configuration task looks like.

Creating new configuration task

- Start with a CMD script
- Declare any files that you need to copy
- Check if reboot is expected or required
- Define if the script needs to take parameters.
- Add manifest to `<MachineFactory>\Config\Tasks.xml`
- Put a reference to your task in topology XML.
- Specify parameter values, if needed
Here you can see a typical configuration task. The task requires two tools to be copied to VHD before deployment – sleep.exe and SetNetworkLocationPrivate.exe. During the execution phase we use those tools to set network location to private.
In the end I would like to briefly shed light on the plug-ability and extensibility of this tool. Given the input/output interface is XML-based (where topology definitions and tasks can be defined as xml and scripts), the topology builder presents a highly extensible and pluggable model. Adding custom new tasks and topology definitions is a breeze. Similarly, plugging it into an existing test infrastructure is straightforward.

Explain the figure now.

Taking advantage of this pluggable model, I decided to take the story forward and plugged in the reporting component into the system. This component could consume XML logs generated by the Execution framework, parse results and store in a database. A monitoring service observes the DB changes and sends instantaneous test-pass summary reports.
I’ll quickly summarize the obvious benefits provided by this tool.

**Benefits**

- **Light-weight and Simple**
  - 55 Kilobytes of download size.
  - Copy on any Windows 7 or higher machine, target a hyper-V server and start building topologies.
  - XML-based interfaces and powershell/cmd based tasks
  - Portable: Deploy on laptop and create topologies from any location

- **Cost-effective and Powerful**
  - No proprietary software required, just windows 7 or higher OS.
  - Single machine environments to complex multi-machine topologies

- **Extensible and Pluggable**
  - If a task is scriptable, it will fit in
  - XML interfaces allow easy plugging into existing test infrastructure
Try it yourself

You can download the tool from

http://mfactory.codeplex.com/
References

MICROSOFT CORPORATION. 2012, Windows PowerShell

MICROSOFT CORPORATION. 2012, Windows Management Instrumentation

CodePlex 2009, Windows PowerShell Management Library for Hyper-V
http://pshyperv.codeplex.com

MICROSOFT CORPORATION. 2010, Remote Desktop Connection Manager – RDCMan 2.2

MICROSOFT CORPORATION. 2012, Script Convert-WindowsImage.psi - WIM to VHD for Windows 8!
http://gallery.technet.microsoft.com/scriptcenter/Convert-WindowsImagepsi-01e73a8f

Markin et al. 2010, Test Environment Configuration in a Complex World
Questions?