INTRODUCTION TO DESIRED STATE CONFIGURATION (DSC)

JEFFERY HICKS, MVP

JHICKS@JDHITSOLUTIONS.COM

WHAT IS DSC?

An extension to the PowerShell language	• Uses PowerShell syntax • Create configuration scripts
Create and manage server configuration files	• Use PowerShell language and cmdlets to create and deploy configurations
Ensures servers are always configured the way you need	• A local configuration manager does the heavy lifting

WHY DSC?

- Prevent server configuration "drift"
- Separate configuration from implementation
- "Continuous" server deployment
- Manage servers on-site or in a cloud
- Leverage your existing PowerShell skills

REQUIREMENTS

Requires Windows Management Framework 4.0

- PowerShell 4.0
- CIM DSC Namespace (Root\Microsoft\Windows\DesiredStateConfiguration)
- DSC cmdlets, providers and resources

.NET Framework 4.5

Windows Server 2008 R2 SP1 and later

Windows 7 SP1 and later

REQUIREMENTS

Verify KB2883200 for Windows 8.1 and Windows Server 2012 R2

PowerShell remoting must be enabled

Optional: Public Key Infrastructure for SSL and encryption certificates

DSC ARCHITECTURE

Push Model

- Configurations deployed to servers
- Use Start-DSCConfiguration to deploy

Pull Model

- Servers poll a central server
- HTTP/HTTPS
- SMB
- Use traditional fault tolerance and load balancing

DSC PHASES

Authoring Phase

- Can include imperative and declarative commands
- Create MOF definitions

Staging Phase

- Declarative MOFs staged
- Configuration calculated per node

“Make It So” Phase

- Declarative configurations implemented through imperative providers

MANAGING CONFIGURATIONS

One configuration (i.e. one MOF) per server

Managed by Local Configuration Manager (LCM)

Think modular and plan ahead

Implement the pull model to simplify management

LOCAL CONFIGURATION MANAGER

Use Get-DSCLocalConfigurationManager to check node settings

- ConfigurationMode
  - ApplyOnly
  - ApplyAndMonitor
  - ApplyAndAutoCorrect
- RefreshMode
  - Push
  - Pull
- RefreshFrequencyMins (15min when using Pull)
- ConfigurationModeFrequencyMins (30min)

Set with a configuration

CREATING A CONFIGURATION

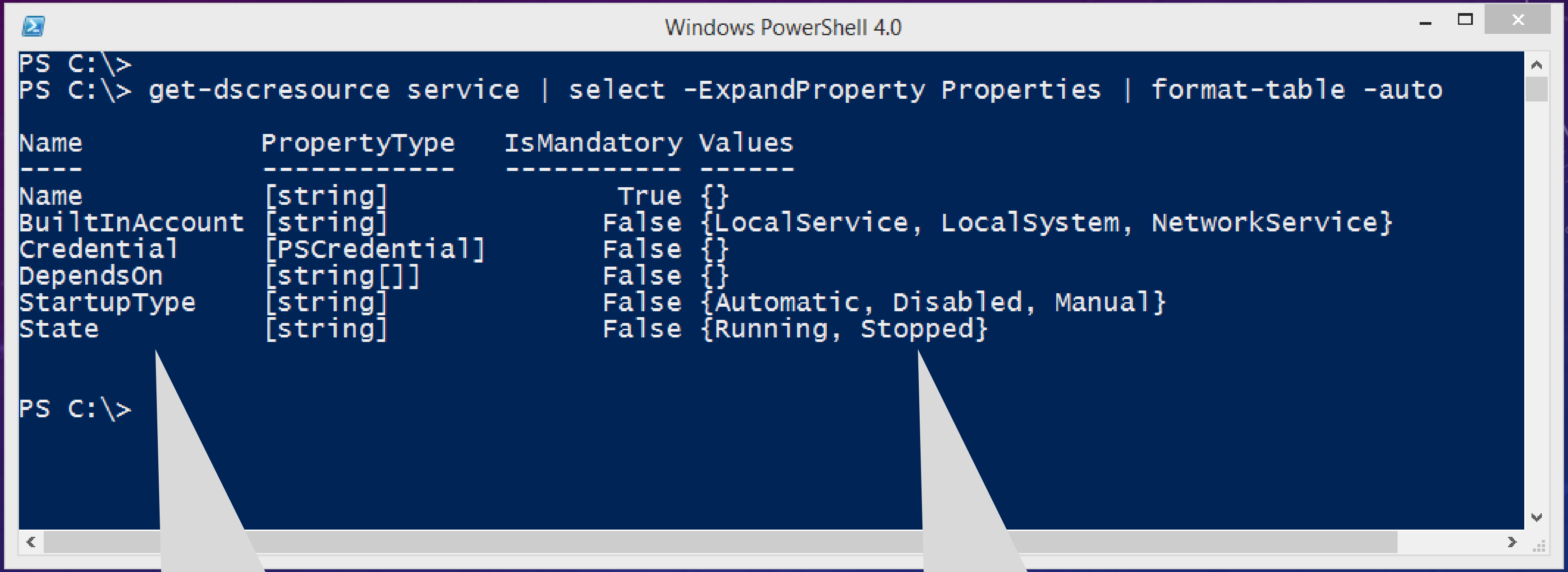
- Defined in a PowerShell script
- Configuration keyword (new command type)
- Define Node
- Configure resources

DSC RESOURCES

- Managed element you define in your configuration
- Core resources shipped “out of the box”
- Additional “experimental” resources shipped from Microsoft
- Community developed resources
- You can write your own

DSC RESOURCES

Provider	**Description**
Archive	Unpacks archive (.zip) files at specific paths on target nodes.
Environment	Manages system environment variables on target nodes.
File	Manages files and directories on target nodes.
Group	Manages local groups on target nodes.
Log	Logs configuration messages.
Package	Installs and manages packages, such as Windows Installer and setup.exe packages, on target nodes.
Registry	Manages registry keys and values on target nodes.
Script	Runs Windows PowerShell script blocks on target nodes.
Service	Manages services on target nodes.
User	Manages local user accounts on target nodes.
Windows Feature	Adds or removes Windows features and roles on target nodes.
Windows Process	Configures Windows processes on target nodes.



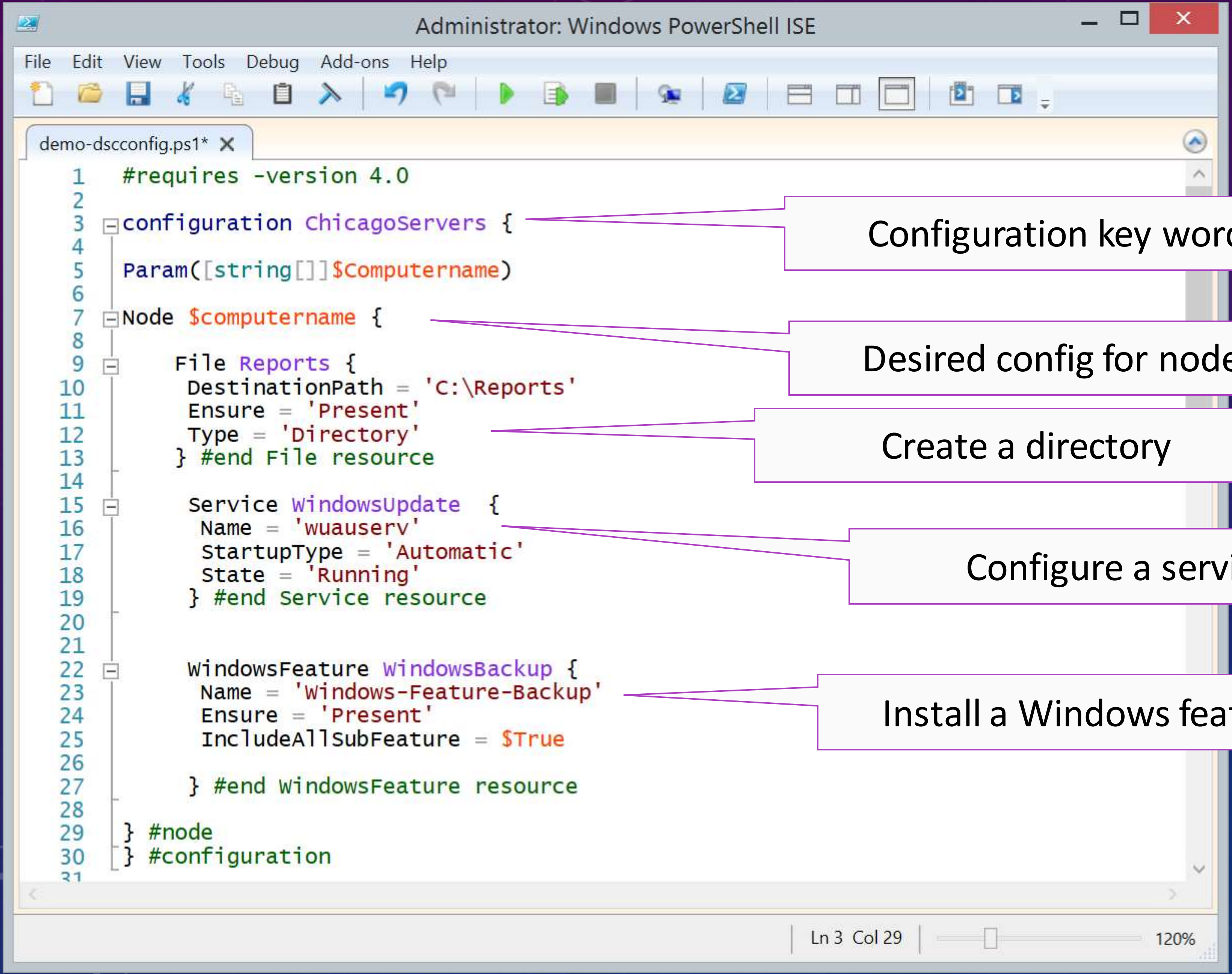
```
PS C:\>
PS C:\> get-dscresource service | select -ExpandProperty Properties | format-table -auto

Name            PropertyType   IsMandatory Values
----            ------------   ----------- ------
Name            [string]              True {}
BuiltInAccount  [string]             False {LocalService, LocalSystem, NetworkService}
Credential      [PSCredential]       False {}
DependsOn       [string[]]           False {}
StartupType     [string]             False {Automatic, Disabled, Manual}
State           [string]             False {Running, Stopped}


PS C:\>
```

Possible Resource settings

Possible Resource values


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
demo-dscconfig.ps1*
Ln 3 Col 29
120%

```
#requires -version 4.0

configuration ChicagoServers {

Param([string[]]$Computername)

Node $computername {

    File Reports {
     DestinationPath = 'C:\Reports'
     Ensure = 'Present'
     Type = 'Directory'
    } #end File resource

     Service WindowsUpdate   {
      Name = 'wuauserv'
      StartupType = 'Automatic'
      State = 'Running'
     } #end Service resource


     WindowsFeature WindowsBackup {
      Name = 'Windows-Feature-Backup'
      Ensure = 'Present'
      IncludeAllSubFeature = $True

     } #end WindowsFeature resource

} #node
} #configuration
```

Configuration key word

Desired config for nodes

Create a directory

Configure a service

Install a Windows feature

DEPLOYING A CONFIGURATION

Define configuration and load into PowerShell

- `PS C:\Scripts> . .\ChicagoCoreConfig.ps1`

Defines a configuration called ‘ChicagoCore’

Invoke the configuration to create MOF

- `PS C:\Scripts> ChicagoCore`

Configuration has hard code node names

Start the configuration on the computer

- `PS C:\Scripts> Start-DscConfiguration -Path .\ChicagoCore`

Configuration pushed to every defined node

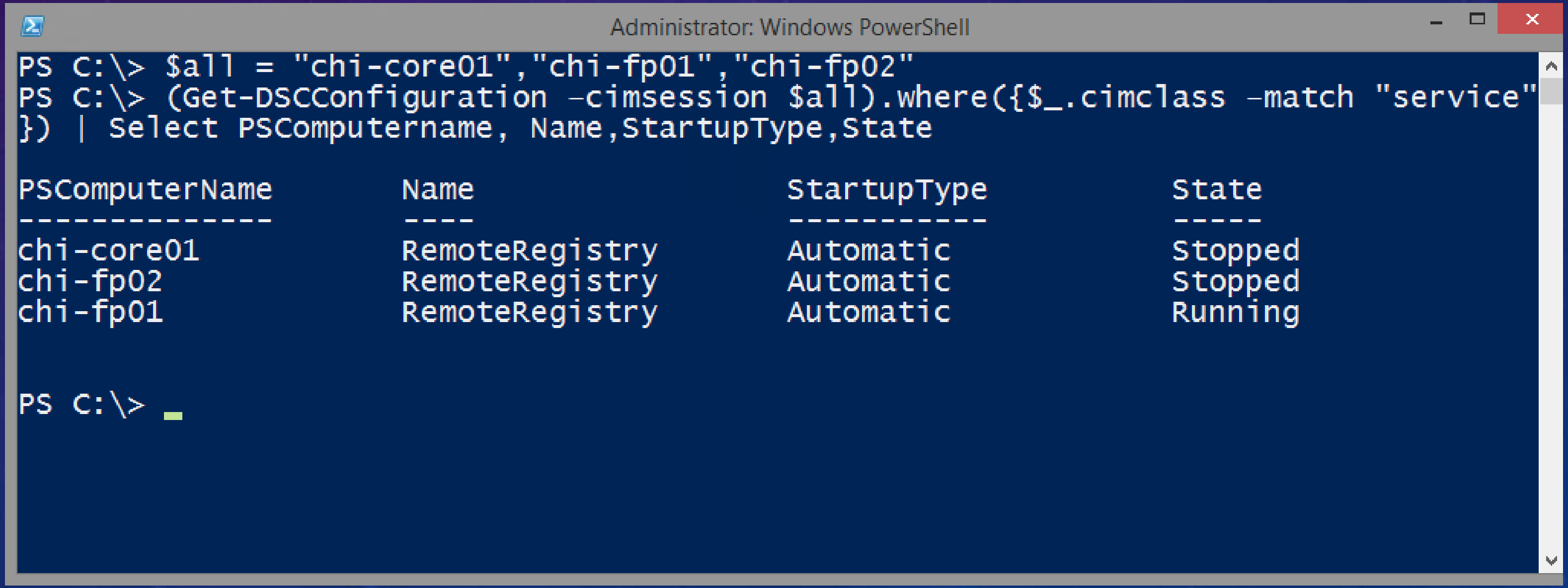
GET DSC CONFIGURATION

Get last applied configuration

- Gets objects for each type of resource
- Use Where-Object to filter for a specific resource or setting
- Use Where-Object to filter on multiple computers

GET DSC CONFIGURATION

```
PS C:\> Get-DSCConfiguration -cimsession CHI-FP02
PS C:\> (Get-DSCConfiguration -cimsession
$all).where({$_.cimclass -match "service"}) | Select
PSComputername, Name,StartupType,State
```

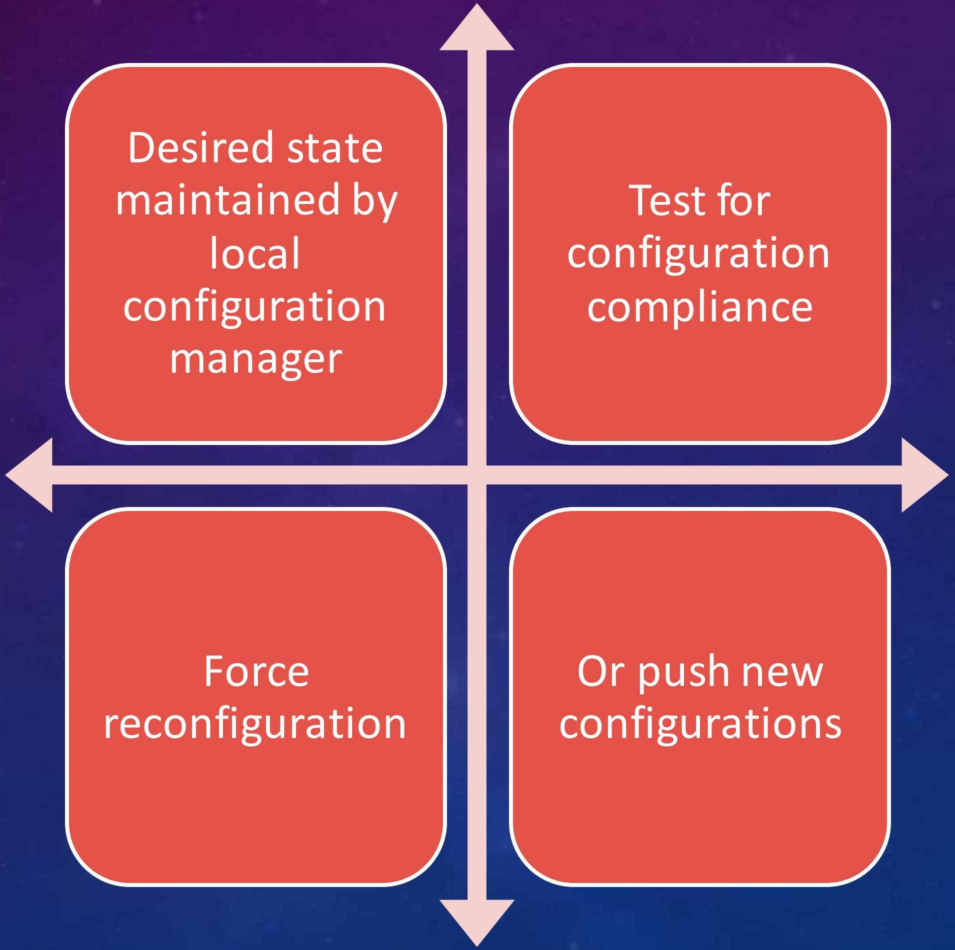


```
PS C:\> $all = "chi-core01","chi-fp01","chi-fp02"
PS C:\> (Get-DSCConfiguration -cimsession $all).where({$_.cimclass -match "service"
}) | Select PSComputername, Name,StartupType,State

PSComputerName       Name                 StartupType          State
--------------       ----                 -----------          -----
chi-core01           RemoteRegistry       Automatic            Stopped
chi-fp02             RemoteRegistry       Automatic            Stopped
chi-fp01             RemoteRegistry       Automatic            Running


PS C:\> _
```

TEST AND RESET A CONFIGURATION


Desired state maintained by local configuration manager
Test for configuration compliance
Force reconfiguration
Or push new configurations

Pull model ensures servers have desired configuration state

TEST CONFIGURATION

- Test-DSCConfiguration returns True or False
- Recommend testing one server at a time
- Use –verbose to see details
- PS C:\> Test-DSCConfiguration –cimsession CHI-FP02

DSC IN ACTION

RESOURCES

- The DSC Book
  - free ebook at http://powershell.org/wp/ebooks
- PowerShell in Depth: An Administrator's Guide 2nd Edition

- DSC Resources on GitHub
  - https://github.com/powershellorg/dsc
- PowerShell Team blog
  - http://blogs.msdn.com/b/powershell/

SUMMARY

DSC requires PowerShell 4.0

DSC leverages your existing PowerShell skills

DSC will become the “norm” for server configuration

Define a server configuration and know that it will always be that way

THANK YOU

http://jdhitsolutions.com/blog

jhicks@jdhitsolutions.com

@JeffHicks

http://plus.google.com/+JefferyHicks