



Windows PowerShell Desired State Configuration (DSC) on the AWS Cloud: Quick Start Reference Deployment

Mike Pfeiffer

September 2014

Last updated: March 2015 ([revisions](#))

Table of Contents

What We'll Cover	3
Architecture Overview	4
A Very Brief Overview of Windows PowerShell DSC	5
How We'll Use Windows PowerShell DSC on AWS.....	7
DSC in Pull Mode.....	7
DSC in Push Mode.....	7
Things That Won't be Handled by DSC	7
Bootstrapping with AWS CloudFormation.....	8
AWS CloudFormation Metadata and cfn-init.....	8
Deployment with a Pull Server Infrastructure	9
DSC Configuration Script Overview.....	9
Bootstrapping the PowerShell DSC Pull Server.....	10
Bootstrapping Client Instances	12
The Configuration Script	14
Deploy the Pull Mode Stack.....	21
Testing Configuration Drift Resistance	22
Deployment with a Push Server Infrastructure	22
Deploy the Push Mode Stack	23
Additional Resources	24
Send Us Your Feedback.....	25
Document Revisions.....	25

What We'll Cover

This Quick Start Reference Deployment includes architectural considerations and configurations used to build a highly available Windows PowerShell Desired State Configuration (PowerShell DSC) “pull server” environment on the Amazon Web Services (AWS) cloud. We discuss how to use the necessary AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Load Balancing (Amazon ELB) and Amazon Virtual Private Cloud (Amazon VPC) together with PowerShell DSC to deploy a highly available enterprise application across separate AWS Availability Zones.

The intent with this guide is to give you a point of reference for implementing your own configuration management solution using the PowerShell DSC platform, and to provide an understanding of the following key topics:

- How to use AWS CloudFormation and PowerShell DSC to bootstrap your servers and applications from scratch
- How to deploy a highly available PowerShell DSC pull server environment using AWS resources
- How to make sure that your instances are resilient to configuration drift once your application stack has been deployed

This guide explores the deployment and management of an internal enterprise web application infrastructure running on the AWS cloud. You can use the patterns in this guide to deploy your own application stack in a similar fashion. You can also deploy automatically the environment outlined in this guide in order to test a fully configured PowerShell DSC pull server infrastructure.

To launch the PowerShell DSC Pull Server infrastructure AWS CloudFormation template into the US West (Oregon) region, [launch the Quick Start](#). This stack takes approximately one hour to create.

Note

You are responsible for all the cost related to your use of any AWS services used while running this Quick Start Reference Deployment. The cost for creating and running the template with default settings is approximately \$3.50 an hour. See the pricing pages of the AWS services you will be using for full details.

After deploying this Quick Start with the default input parameters, you will have built the following PowerShell DSC pull server environment on the AWS cloud:

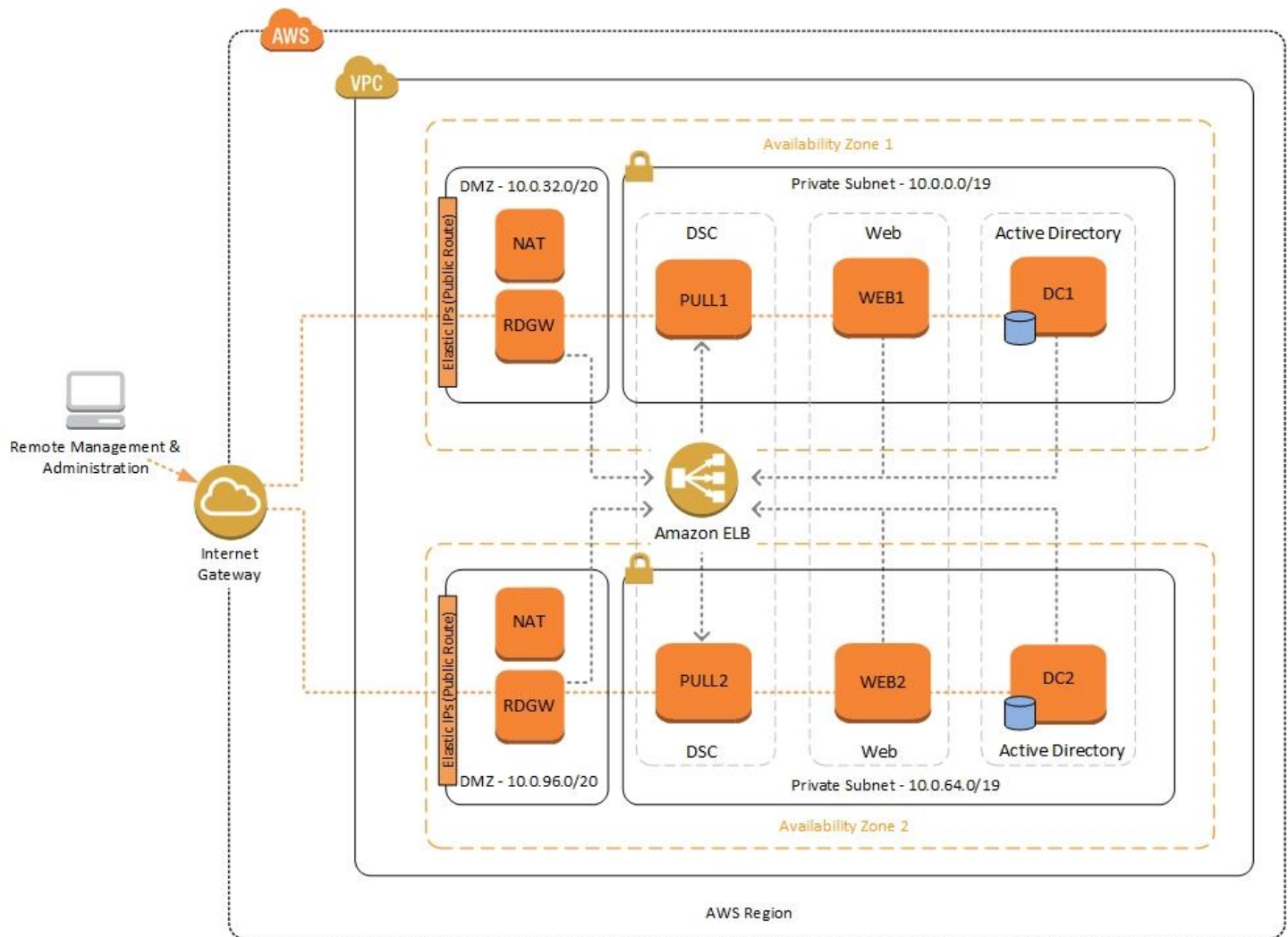


Figure 1: Highly Available PowerShell DSC Pull Server Infrastructure on AWS

Architecture Overview

The core AWS components used by this reference include the following AWS services:

- [Amazon VPC](#) – the Amazon Virtual Private Cloud service lets you provision a private, isolated section of the AWS cloud where you can launch AWS services and other resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.
- [Amazon EC2](#) – the Amazon Elastic Compute Cloud service allows you to launch virtual machine instances with a variety of operating systems. You can choose from existing Amazon Machine Images (AMIs) or import your own virtual machine images.
- [Amazon ELB](#) – the Amazon Elastic Load Balancing service automatically distributes incoming traffic across multiple Amazon EC2 instances. It enables you to achieve greater levels of fault tolerance in your applications.

- [Amazon S3](#) – the Amazon Simple Storage Service provides highly durable and available cloud storage for a variety of content, ranging from web applications to media files. It allows you to offload your entire storage infrastructure onto the cloud.

When deploying a Windows-based environment on the AWS cloud, we recommend an architecture that supports the following requirements:

- Critical workloads should be placed in a minimum of two Availability Zones to provide high availability.
- Internal application servers and other non-Internet facing servers should be placed in private subnets to prevent direct access to these instances from the Internet.
- Remote Desktop Gateways should be deployed into public subnets in each Availability Zone for remote administration. Other components, such as reverse proxy servers, can also be placed into these public subnets if needed.

A Very Brief Overview of Windows PowerShell DSC

If you are new to PowerShell DSC, we highly recommend that you consult the additional resources at the end of this guide for a deeper look at the topic. For now, we'll quickly cover what PowerShell DSC is and how it works.

PowerShell DSC was introduced in Windows Management Framework 4.0. It provides a configuration management platform native to Windows Server 2012 R2 and Windows 8.1, and available to Windows Server 2008 R2, Windows 7, and Linux. PowerShell DSC allows you to express the desired state of your systems using declarative language syntax instead of configuring servers with complex imperative scripts. If you've worked with configuration management tools like Chef or Puppet, you'll notice that PowerShell DSC provides a familiar framework.

When using DSC to apply a desired configuration for a system, you create a configuration script with PowerShell that explains what the system should look like. You then generate a Management Object Format (MOF) file using that configuration script, which is then pushed or pulled by a "node" to apply the desired state. PowerShell DSC uses vendor-neutral MOF files to enable cross-platform management, and "nodes" refer to either Windows or Linux systems.

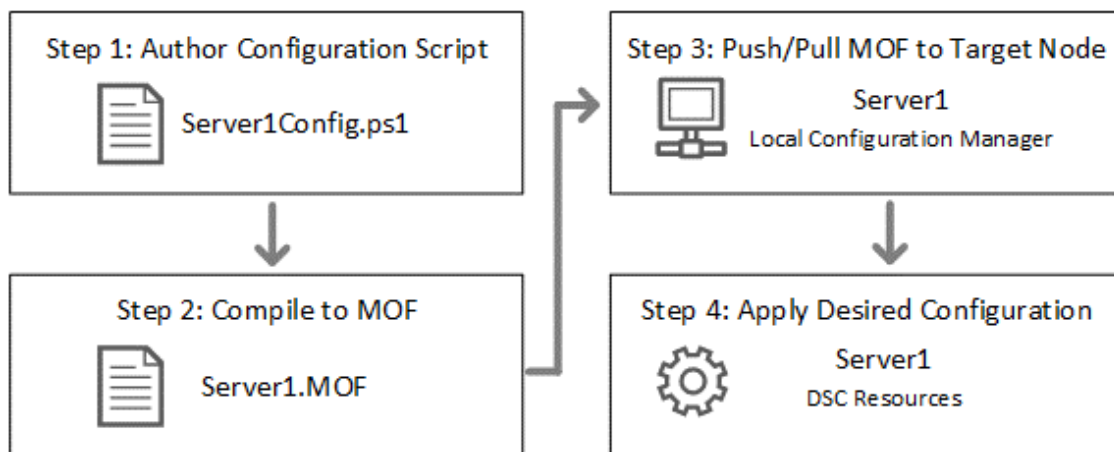


Figure 2: High Level DSC Architecture

Windows systems running Windows Management Framework 4.0 or later include an engine called the Local Configuration Manager, which acts as a DSC client. The Local Configuration Manager calls the DSC resources required by the configuration defined in the MOF files. These DSC resources perform the work of applying the desired configuration.

```
1 Configuration MyService {
2     Node Server1 {
3         Service bits {
4             Name = 'bits'
5             State = 'running'
6         }
7     }
8 }
9
10 MyService
11 Start-DscConfiguration -Path .\MyService -Wait -Verbose
```

Figure 3: Basic DSC Configuration Script

Figure 3 shows an example of a very simple DSC configuration script that can be used to push a desired configuration to a computer.

- **Line 1** – We use the Configuration keyword to define a name (MyService) for the configuration.
- **Line 2** – The node keyword is used to define the desired state for a server named “Server1.”
- **Lines 3 through 6** – We’re creating an instance of the Service resource called “bits.” As you can see, within the resource, we’re declaring that the service actually named “bits” should be in a running state.
- **Line 10** – The configuration is executed, which generates a MOF file called Server1.mof in a folder called MyService.
- **Line 11** – The Start-DscConfiguration cmdlet is used to push the MOF file in the MyService folder to the computer Server1. When doing this interactively, it’s useful to use the -Wait and -Verbose parameters to get detailed information

As we will see later in this guide, the configuration scripts used to deploy the reference architecture will include several resources for each server in the topology. Some of those will be native to the operating system, some of them will be additional resources provided by Microsoft, and others will be custom written resources to fill in the gaps.

Note

Keep in mind that this guide is not a complete tutorial on DSC. It’s simply a reference architecture for how systems can be deployed and configured in tandem with DSC and AWS CloudFormation. For a complete understanding of the mechanics of DSC pull servers and writing custom DSC resources, we highly recommend consulting the supplemental reading at the end of this guide.

How We'll Use Windows PowerShell DSC on AWS

PowerShell DSC clients can pull their configurations from a server or have their configurations pushed to them either locally or from a remote system. In this guide, we've provided two AWS CloudFormation templates that deploy the reference environment using both the pull and push models. In this section, we'll discuss the differences between these two methods and how they will be used to deploy our stack on AWS.

DSC in Pull Mode

To deploy our reference architecture using DSC in pull mode, we'll use AWS CloudFormation to create the Amazon VPC and required network elements. Then we'll launch servers to act as DSC pull servers. On these servers we'll install a web service which will allow the Local Configuration Manager on client nodes to pull their configuration via HTTPS. To do this, we'll use a master configuration script to generate MOF files for each node in our deployment. As each server is bootstrapped by AWS CloudFormation, we'll configure the Local Configuration Manager to retrieve its configuration from the pull server and the desired state will be applied.

Once the stack is built successfully, we'll have a distributed enterprise application deployment that maintains its desired state and is resistant to configuration drift.

DSC in Push Mode

Our second example will deploy the same reference architecture in DSC push mode. In this scenario, there will be no requirement for DSC pull servers, and each server in the environment will "push" a configuration document to itself.

Again, we'll utilize AWS CloudFormation to orchestrate the build process. After creating the Amazon VPC and required network elements, each server instance will be launched and bootstrapped. Since there will be no pull servers in this scenario, each instance will download its own configuration script, generate the MOF document, and apply the desired state using the Start-DSCConfiguration cmdlet.

Things That Won't be Handled by DSC

Most Windows-specific tasks will be handled by PowerShell DSC. However, there are a few things that we'll do using helper scripts called from AWS CloudFormation `cf-init` in order to start the bootstrapping process.

- **Renaming the Computer** – We'll simply use the `Rename-Computer` cmdlet before invoking a DSC push or pull operation. This allows us to use each server's hostname when using the "node" keyword in the DSC configuration scripts.
- **Installing Certificates** – For the sake of demonstration, we'll utilize self-signed certificates to secure the DSC pull server's HTTPS endpoint and to encrypt credentials when using DSC resources that require authentication. Self-signed certificates will be installed on each instance in the environment to support these scenarios. For production environments, we recommend utilizing an internal PKI or commercial SSL certificate provider.
- **Downloading Configurations and Modules** – Configuration scripts and DSC resource modules will be downloaded from a public Amazon S3 bucket. The MOF files will be generated on the fly during the bootstrapping process. This does not apply to servers configured in pull mode, which will obtain their configurations and any required modules from the pull server.

Bootstrapping with AWS CloudFormation

AWS CloudFormation allows you to define a set of resources needed to run an application in the form of a declarative JSON (JavaScript Object Notation) template. The resources within the template can include Amazon EC2 instances, Elastic Load Balancers, and more. AWS CloudFormation takes care of provisioning all of your AWS resources, and there are a number of techniques that can be used to bootstrap applications running on your Amazon EC2 instances. The instances in this Quick Start Reference Deployment are partially bootstrapped using the AWS CloudFormation helper process called [cfn-init](#), which allows us to download scripts and define initialization commands. Once the instance is up and running, we'll rely on PowerShell DSC for the Windows and application-specific configuration.

AWS CloudFormation Metadata and cfn-init

You can attach metadata to any resource in your AWS CloudFormation template. The `cfn-init` helper script retrieves and interprets the resource metadata to create and download files, start services, and define commands that should be executed on the instance. Figure 4 shows a simple example of the techniques used in this Quick Start Reference Deployment to bootstrap instances using AWS CloudFormation.

```

35     "Ec2Instance" : {
36         "Type" : "AWS::EC2::Instance",
37         "Metadata" : {
38             "AWS::CloudFormation::Init" : {
39                 "configSets" : {
40                     "config" : [
41                         "rename",
42                         "bootstrapDSC"
43                     ]
44                 },
45                 "rename" :{
46                     "a-rename-computer" : {
47                         "command" : "powershell.exe -Command Rename-Computer -NewName Server1 -Restart",
48                         "waitAfterCompletion" : "forever"
49                     }
50                 },
51                 "bootstrapDSC" : {
52                     "a-setpullmode" : {
53                         "command" : "powershell.exe -Command c:\\cfn\\scripts\\SetPullmode.ps1",
54                         "waitAfterCompletion" : "0"
55                     }
56                 }
57             }
58         },
59         "Properties" : { ...
74     },
75 }

```

Figure 4: Simplified Example of Bootstrapping a DSC Client Instance using AWS CloudFormation

When `cfn-init` runs on the instance, it looks for resource metadata rooted in the [AWS::CloudFormation::Init](#) metadata key. The metadata is organized into config keys, which you can optionally group into configsets. Using configsets allows you to organize a sequence of commands into individual groups, and `cfn-init` will process each configset in order. Please note the following aspects of this process as depicted in Figure 4:

- **Lines 41 and 42** – Two configsets are defined: one that will rename and reboot the computer, and another to bootstrap the DSC client upon rebooting.
- **Line 47** – The single PowerShell command in the “rename” configset which renames and reboots the computer. Keep in mind that we can have numerous sections within each configset. Each can create its own set of files, start services, or run commands.
- **Line 48** – The `waitAfterCompletion` key specifies how long to wait (in seconds) after a command has finished in case the command causes a reboot. The default value is 60 seconds and a value of "forever" directs cfn-init to exit and resume only after the reboot is complete.
- **Line 53** – The `SetPullMode.ps1` script runs after the reboot to bootstrap the PowerShell DSC client.

Bootstrapping applications with AWS CloudFormation is a broad topic, and there are several other techniques that can be useful depending on your requirements. We recommend that you review [Bootstrapping Applications via AWS CloudFormation](#) for more details.

Deployment with a Pull Server Infrastructure

In this section, we will cover the process of using a single PowerShell DSC configuration script along with an AWS CloudFormation template to deploy our sample architecture.

DSC Configuration Script Overview

Our pull server infrastructure uses a single DSC configuration script that applies to all of the servers in the deployment. The configuration script ensures the systems implement the following changes:

- Create the Active Directory Forest and Domain and build a Domain Controller in the first Availability Zone
- Configure the Active Directory Site Topology
- Join each node to the domain
- Promote another server to a Domain Controller in the second Availability Zone
- Install Remote Desktop Gateway Services on the Remote Desktop Gateways in public subnets
- Deploy IIS and our sample web page on servers in each Availability Zone

Many of these tasks involved in creating this infrastructure use DSC resources that are not native to the operating system. Microsoft has made a number of additional and experimental DSC resources available for download in “DSC Resource Kit Waves.” We’ve used several of these to configure the state of the systems in this architecture. Additionally, we’ve created custom DSC resources in order to configure certain aspects of the environment that are not currently supported by Microsoft’s provided DSC resources.

To help ensure that these DSC resources will always be available, we’ve saved copies of them in an Amazon S3 bucket. This prevents our automated deployment template from being broken if the links suddenly change on the Internet. It also provides the ability to roll back to prior versions of code since the Amazon S3 bucket has versioning enabled.

Bootstrapping the PowerShell DSC Pull Server

The bootstrapping sequence for the pull server lays the foundation for building the rest of the environment. As depicted in Figure 1, each client node accesses load balanced pull servers through Elastic Load Balancing. The bootstrapping process for the pull server includes the following:

- **IAM Role** – The pull server launches with an IAM role, allowing the instance to call the DescribeLoadBalancers and DescribeInstances actions. This process allows the pull server to determine the DNS name for the Elastic Load Balancer and to query tags set on each Amazon EC2 instance in the stack.
- **Setup** – Downloads all of the required components, such as the pull server configuration script, the master configuration script, the DSC resource modules, and other helper scripts. These file downloads are completed by the AWS CloudFormation files resource.
- **Self-Signed Certificate** – The pull server creates a self-signed certificate using a helper script. The DNS name of the internal Elastic Load Balancer is obtained from the Get-ELBLoadBalancer cmdlet. The DNS name is used as the common name on the self-signed certificate. This allows client nodes to pull their configurations through the load balanced endpoint using secure HTTPS connections.
- **Bootstrapping the DSC Web Service** – The pull server runs the CreatePullServer.ps1 configuration script, which outputs a MOF file for the pull server. The settings are then applied locally to create the DSC web service listening on TCP port 8080, which is configured to use the self-signed certificate thumbprint created previously to secure the web service.
- **Generating Configurations** – The pull server executes the master configuration script, which produces a MOF file for each server in the environment. Each file must be renamed to the associated nodes ConfigurationID, which is represented as a globally unique identifier (GUID). Each instance is tagged in Amazon EC2 with a GUID using the AWS CloudFormation template. The pull server is able to obtain these ConfigurationIDs, match them with each node in the topology, and rename and checksum the file.

Pull Server Configuration Script

Figure 5 shows the code for CreatePullServer.ps1, which is the configuration script used to create the pull server.

```

1 Configuration CreatePullServer {
2   param(
3     [string[]]$Computersname
4   )
5
6   Import-DscResource -ModuleName xPSDesiredStateConfiguration
7
8   Node $Computersname {
9     WindowsFeature DSCServiceFeature {
10      Ensure = "Present"
11      Name = "DSC-Service"
12    }
13
14    xDSCWebService PSDSCPullServer {
15      Ensure = "Present"
16      EndpointName = "PSDSCPullServer"
17      Port = 8080
18      PhysicalPath = "$env:SystemDrive\inetpub\wwwroot\PSDSCPullServer"
19      CertificateThumbPrint = (Get-ChildItem Cert:\LocalMachine\My)[0].Thumbprint
20      ModulePath = "$env:ProgramFiles\WindowsPowerShell\DscService\Modules"
21      ConfigurationPath = "$env:ProgramFiles\WindowsPowerShell\DscService\Configuration"
22      State = "Started"
23      DependsOn = "[WindowsFeature]DSCServiceFeature"
24    }
25  }
26 }
27
28 CreatePullServer -Computersname $env:COMPUTERNAME -OutputPath c:\DSC
29 Start-DscConfiguration -Path c:\DSC -Wait

```

Figure 5: Pull Server Configuration Script

The CreatePullServer.ps1 configuration script depends on the xPSDesiredConfiguration resource module. This is a non-native module that can be obtained from Microsoft. The AWS CloudFormation template is configured to download this module from Amazon S3 as a .zip file, which it then unpacks into `$env:ProgramFiles\WindowsPowerShell\DscService\Modules` on the pull server.

A few points of interest about this configuration script:

- **Line 9** – We declare the DSCServiceFeature, ensuring that the DSC-Service WindowsFeature is present and installed on the server.
- **Line 19** – Notice that the value of the certificate thumbprint is retrieved from the machine's local certificate store. This is because we previously generated and installed the self-signed certificate during the bootstrapping process.
- **Line 23** – The PSDSCPullServer resource uses the DependsOn attribute to ensure that the DSC-Service is first installed before attempting to configure the DSC Web Service.
- **Line 29** – The call to Start-DscConfiguration tells us that this is a DSC “push” operation. The pull server is pushing this configuration to itself. Other nodes in the environment will be configured in pull mode.

The CreatePullServer.ps1 script is called on the pull server using AWS CloudFormation.

```
"bootstrapPullSvc" : {  
  "commands" : {  
    "a-create-pullserver" : {  
      "command" : "powershell.exe -Command c:\\cfn\\scripts\\CreatePullServer.ps1",  
      "waitAfterCompletion" : "0"  
    },  
  },  
}
```

Figure 6: Running the CreatePullServer.ps1 Configuration Script on Pull1 using AWS CloudFormation

Remember, there are two pull servers in this environment to provide high availability. Each pull server instance is bootstrapped using the steps outlined here.

After you've deployed your environment, you'll need to make sure that downloadable content (MOF files, resource modules, and checksums) are kept up to date on both pull servers. This can be done as a procedure of your deployment process or by using a file synchronization service to keep the modules and configuration directories on pull servers in sync.

Bootstrapping Client Instances

The bootstrapping process for the each client instance includes the following:

- **IAM Role** – Each server launches with an IAM role, allowing the instance to call the DescribeLoadBalancers and DescribeInstances actions. This process allows the server to determine the DNS name for the Elastic Load Balancer and to query tags set on each Amazon EC2 instance in the stack.
- **Setup** -- Downloads helper scripts from Amazon S3. These file downloads are completed by the AWS CloudFormation files resource.
- **Certificates** – In addition to connecting to the correct DNS name, clients must also trust the certificate installed on the pull server. The self-signed certificate is downloaded from the pull server and installed locally. Keep in mind this is for demonstration purposes, and an enterprise PKI solution is likely the best method for doing this in production.
- **Configuring the LCM** – The Local Configuration Manager is then configured with the HTTPS endpoint that should be used as the pull server. In this case, the endpoint will be the Elastic Load Balancer. Additionally, the ConfigurationID for the node is set. Again, each Amazon EC2 instance is tagged with a unique ConfigurationID, which can also be obtained from Amazon EC2.

Client Bootstrap Configuration Script

Figure 7 shows the code for SetPullMode.ps1, which is the configuration script used to configure the client.

```

1 param(
2     [string]$Instance,
3     [string]$Region,
4     [string]$VpcId
5 )
6
7 $guid = (Get-EC2Instance -Filter @{name='tag:Name';values=$Instance} -Region $Region)[0].Instances.tags.where{$_ .key -eq 'guid'}.value
8 $pullServer = Get-ELBLoadBalancer -Region $Region | Where-Object { $_.VpcId -eq $VpcId } | select -ExpandProperty DnsName
9
10 Configuration SetPullMode {
11     Node $env:COMPUTERNAME {
12         LocalConfigurationManager {
13             ConfigurationMode = 'ApplyAndAutoCorrect'
14             ConfigurationID = $guid
15             CertificateId = (Get-ChildItem Cert:\LocalMachine\My | Where-Object { $_.Subject -eq "CN=$pullServer" })[0].Thumbprint
16             RefreshMode = 'Pull'
17             ConfigurationModeFrequencyMins = 30
18             RefreshFrequencyMins = 15
19             RebootNodeIfNeeded = $true
20             DownloadManagerName = 'WebDownloadManager'
21             DownloadManagerCustomData = @{
22                 ServerUrl = "https://$( $pullServer ):8080/PSDSCPullServer.svc"
23                 AllowUnsecureConnection = 'false'
24             }
25         }
26     }
27 }
28
29 SetPullMode
30 Set-DscLocalConfigurationManager -ComputerName $env:COMPUTERNAME -Path .\SetPullMode.ps1
31

```

Figure 7: DSC Client Configuration Script

A few points of interest about this configuration script:

- **Line 7** – The GUID assigned to the instance is retrieved from the Amazon EC2 “guid” tag and stored in a variable.
- **Line 8** – We store the DNS name of the Elastic Load Balancer in a variable called \$PullServer.
- **Line 13** – The Local Configuration Manager ConfigurationMode is set to ApplyAndAutoCorrect. This setting helps ensure that modifications and configuration drift issues are corrected and that the system remains in the desired state.
- **Line 14** – The ConfigurationID for the client node is set using the value of the “guid” tag on the Amazon EC2 instance.
- **Line 15** – The CertificateID is set to the thumbprint of the self-signed certificate that was obtained from the pull server. In addition to using the self-signed certificate on the pull server to secure HTTPs connections, it’s also used for encryption. Defining CertificateID allows the client node to decrypt credentials in the MOF documents.
- **Line 22** – the ServerUrl is configured to use the \$PullServer variable, which is set to the DNS name of the Elastic Load Balancer.

The SetPullMode.ps1 script is called on each instance from AWS CloudFormation. The Instance and Region parameter values are passed in at runtime.

```
"c-set-dsc-pull-mode" : {
  "command" : {
    "Fn::Join" : [
      "",
      [
        "powershell.exe -Command c:\\cfn\\scripts\\SetPullMode.ps1 -Instance DC1",
        "-Region ",
        {
          "Ref" : "AWS::Region"
        }
      ]
    ]
  },
  "waitAfterCompletion" : "0"
}
```

Figure 8: Running the SetPullMode.ps1 Configuration Script using AWS CloudFormation

After the pull mode has been set on the instance, a pull operation is invoked manually. This is done by calling the Update-DscConfiguration cmdlet that was released with the November 2014 update rollup for Windows Server 2012 R2 (KB3000850).

To view the Local Configuration Manager settings (meta-configuration) for a node, you can use the Get-DscLocalConfigurationManager cmdlet.

The Configuration Script

Now that we understand how the pull servers and client instances are bootstrapped, let's take a closer look at the configuration script that is responsible for defining the state of each server in the environment.

It's important to keep in mind that in this Quick Start Reference Deployment, the pull server is used as a "build server" meaning that it downloads and runs the configuration script, generating MOFs for all the servers in the environment. This means that any additional resources need to be downloaded and extracted into *\$env:ProgramFiles\WindowsPowerShell\Modules* on the pull server during the bootstrapping process.

Let's take a look at the structure of the configuration script pictured in the following figure. Several code blocks are collapsed and will be covered in greater detail in the following sections of this guide.

```

1 param(
2     [string]$DomainDNSName,
3     [string]$DomainNetBiosName,
4     [string]$AdminPassword,
5     [string]$ADServer1PrivateIp,
6     [string]$ADServer2PrivateIp,
7     [string]$PrivateSubnet1CIDR,
8     [string]$PrivateSubnet2CIDR,
9     [string]$DMZ1CIDR,
10    [string]$DMZ2CIDR,
11    [string]$Region,
12    [string]$VpcId
13 )
14
15 #Get the FQDN of the Load Balancer
16 $PullServer = Get-ELBLoadBalancer -Region $Region | Where-Object {$_.VpcId -eq $VpcId} | select -ExpandProperty DnsName
17
18 #Helper functions
19 Import-Module $pscriptroot\Get-EC2InstanceGuid.psm1
20 Import-Module $pscriptroot\IPHelper.psm1
21
22 #Node Configuration Settings
23 $ConfigurationData = @{...}
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61 #Credentials used for creating & joining the AD Domain
62 $Pass = ConvertTo-SecureString $AdminPassword -AsPlainText -Force
63 $Credential = New-Object System.Management.Automation.PSCredential -ArgumentList "$DomainNetBiosName\administrator", $Pass
64
65 #Master Configuration for all nodes in deployment
66 Configuration ServerBase {...}
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193 #Compile and rename the MOF files
194 $MofFiles = ServerBase -ConfigurationData $ConfigurationData
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 9: The Structure of the Master Configuration Script

Lines 1 through 13 – The param block includes a number of parameters that are used to define the settings in our environment. All of these parameters map to the parameters of the AWS CloudFormation template. When you launch the stack, you’re given the opportunity to customize the environment at launch time. You can change subnet CIDR ranges, IP addresses, the DNS name of the Active Directory Domain, and more. These parameter values are passed in from AWS CloudFormation to the pull server that builds the configurations using the settings you’ve provided.

Line 16 – A single call to Get-ELBLoadBalancer to get the DNS name of the Elastic Load Balancer. This will be the endpoint that client nodes use to pull configurations from the pull server.

Lines 19 and 20 – Imports helper functions used to retrieve node GUIDs and aid in formatting IP information used by the xNetworking DSC resource.

Line 23 – The Configuration Data for the environment, covered in more detail in the following sections of this guide.

Lines 62 and 63 – The credentials used by member servers to join the Active Directory Domain.

Line 66 – The DSC configuration called “ServerBase.” Within the configuration, we import the xNetworking, xActiveDirectory, and xComputerManagement DSC resources since they are not native to Windows Server 2012 R2.

Line 294 and beyond – The code used to create the MOF files, rename them, and then move them to the appropriate directory on the pull server.

Now let’s take a closer look at each aspect of the Configuration script.

DSC Resources

The only DSC resource used in our configuration script that is native to Windows Management Framework 4.0 is the [WindowsFeature](#) resource. The other resources, which include resources from the xNetworking, xActiveDirectory, and xComputerManagement resource modules, were made available by Microsoft in an out-of-band release. These resource modules have been stored in Amazon S3 and are downloaded by the pull servers in .zip format. The pull servers are then configured to host the zipped resource modules so they can be downloaded from DSC client nodes.

Configuration Data

Configuration Data provides a way to define additional environmental settings for the nodes in a configuration. The configuration data is a hash table of settings that can be passed into a configuration when generating MOF documents. Figure 10 shows an example of the configuration data used for the master DSC configuration for our environment.

```
$ConfigurationData = @{
  AllNodes = @(
    @{
      NodeName = '*'
      CertificateFile = 'C:\inetpub\wwwroot\dsc.cer'
      Thumbprint = (Get-ChildItem Cert:\LocalMachine\My | Where-Object { $_.Subject -eq "CN=$PullServer" })[0].Thumbprint
    }
    @{
      NodeName = 'RDGW1'
      Guid = (Get-EC2InstanceGuid -Region $Region -InstanceName RDGW1)
      AvailabilityZone = 'AZ1'
    }
    @{
      NodeName = 'RDGW2'
      Guid = (Get-EC2InstanceGuid -Region $Region -InstanceName RDGW2)
      AvailabilityZone = 'AZ2'
    }
    @{...}
    @{...}
    @{...}
    @{...}
  )
}
```

Figure 10: DSC Configuration Data

In the example in Figure 10 (which is condensed for the sake of brevity) the configuration data contains a hash table for each node in our deployment. Each property is described below:

- **NodeName** – The hostname of the instance, which corresponds to the node name in the configuration script.
- **Guid** – The ConfigurationID (in the form of a GUID) that the pull server and DSC client both use to determine which configuration should be pulled from the server.
- **AvailabilityZone** – This is a custom property to indicate which AWS Availability Zone the instance is located in. As we'll see, this custom property is used within the configuration script to apply settings specific to the location of the instance.
- **CertificateFile** – The physical path on the “build server” (in this case, the pull server) of the certificate that will be used to encrypt embedded credentials in the resulting MOF file. Remember, client nodes must have the private key to decrypt the credentials.
- **Thumbprint** – The certificate thumbprint on the client node to indicate which installed certificate should be used to decrypt data.

DNS Client Configuration

Since our architecture will be distributed across two Availability Zones, it makes sense for domain-joined servers to use the Active Directory DNS server in the local Availability Zone. We account for this in our configuration using some additional logic and the `xDnsServerAddress` resource, which is made available as part of the `xNetworking` resource module.

```
Node $AllNodes.where{$_ .AvailabilityZone -eq 'AZ1'}.NodeName {
  xDnsServerAddress DnsServerAddress {
    Address          = $ADServer1PrivateIp, $ADServer2PrivateIp
    InterfaceAlias   = 'Ethernet'
    AddressFamily    = 'IPv4'
  }
}

Node $AllNodes.where{$_ .AvailabilityZone -eq 'AZ2'}.NodeName {
  xDnsServerAddress DnsServerAddress {
    Address          = $ADServer2PrivateIp, $ADServer1PrivateIp
    InterfaceAlias   = 'Ethernet'
    AddressFamily    = 'IPv4'
  }
}
```

Figure 11: DNS Settings

As you can see in Figure 11, we filter `$AllNodes` (which will be defined by our configuration data) so that instances in AZ1 will point to the Domain Controller in the first Availability Zone for its primary DNS server, and vice versa.

Domain Controller Configuration

The Domain Controller configuration is mostly completed by resources from the `xActiveDirectory` resource module. In order to fully implement a distributed Active Directory topology, we add a number of additional resources to the module.

```
Node DC1 {
  CIPAddress DCIPAddress {
    InterfaceAlias = 'Ethernet'
    IPAddress      = $ADServer1PrivateIp
    DefaultGateway = (Get-AWSDefaultGateway -IPAddress $ADServer1PrivateIp)
    SubnetMask     = (Get-AWSSubnetMask -SubnetCIDR $PrivateSubnet1CIDR)
  }

  xDnsServerAddress DnsServerAddress {
    Address          = $ADServer1PrivateIp
    InterfaceAlias   = 'Ethernet'
    AddressFamily    = 'IPv4'
    DependsOn       = '[CIPAddress]DCIPAddress'
  }

  windowsFeature ADDSInstall {
    Ensure = 'Present'
    Name    = 'AD-Domain-Services'
    DependsOn = '[CIPAddress]DCIPAddress'
  }

  windowsFeature ADDSToolsInstall {
    Ensure = 'Present'
    Name    = 'RSAT-ADDS-Tools'
  }
}
```

```

xADDomain ActiveDirectory {
  DomainName = $DomainDNSName
  DomainAdministratorCredential = $Credential
  SafemodeAdministratorPassword = $Credential
  DependsOn = '[WindowsFeature]ADDSInstall'
}

CADSubnet AZ1Subnet1 {
  Name = $PrivateSubnet1CIDR
  Site = 'Default-First-Site-Name'
  Credential = $Credential
  DependsOn = '[xADDomain]ActiveDirectory'
}

CADSubnet AZ1Subnet2 {
  Name = $DMZ1CIDR
  Site = 'Default-First-Site-Name'
  Credential = $Credential
  DependsOn = '[xADDomain]ActiveDirectory'
}

CADSite AZ2Site {
  Name = 'AZ2'
  DependsOn = '[WindowsFeature]ADDSInstall'
  Credential = $Credential
}

CADSubnet AZ2Subnet1 {
  Name = $PrivateSubnet2CIDR
  Site = 'AZ2'
  Credential = $Credential
  DependsOn = '[CADSite]AZ2Site'
}

CADSubnet AZ2Subnet2 {
  Name = $DMZ2CIDR
  Site = 'AZ2'
  Credential = $Credential
  DependsOn = '[CADSite]AZ2Site'
}

CADSiteLinkUpdate SiteLinkUpdate {
  Name = 'DEFAULTIPSITELINK'
  SitesIncluded = 'AZ2'
  Credential = $Credential
  DependsOn = '[CADSubnet]AZ2Subnet1'
}
}

```

Figure 12: DC1 Configuration

A few points to note about the configuration for DC1 as shown in Figure 12:

- The cIPAddress resource sets the IP Address for the DC through the AWS CloudFormation template parameter. The default gateway and subnet mask are set based off the IP Address and subnet CIDR using a couple of helper functions.
- The cADSubnet resource is used to create subnet definitions for each subnet in the Amazon VPC. The cADSubnet is a custom resource that was not originally included in the xActiveDirectory resource module. We added it to aid in configuring the Active Directory (AD) Site Topology for this Quick Start Reference Deployment.

- The cADSite resource is used to create AD Site objects for each Availability Zone that will host a Domain Controller. The cADSite is a custom resource that was not originally included in the xActiveDirectory resource module. We added it to aid in configuring the Active Directory Site Topology for this Quick Start Reference Deployment.
- The cADSiteLinkUpdate resource is used to link the two AD sites so the Domain Controllers will replicate data to each other. The cADSiteLinkUpdate is a custom resource that was not originally included in the xActiveDirectory resource module. We added it to aid in configuring the Active Directory Site Topology for this Quick Start Reference Deployment.

After the first Domain Controller is built, DC2 is installed in the second Availability Zone using the following node configuration:

```
Node DC2 {
  cIPAddress DC2IPAddress {
    InterfaceAlias = 'Ethernet'
    IPAddress = $ADServer2PrivateIp
    DefaultGateway = (Get-AWSDefaultGateway -IPAddress $ADServer2PrivateIp)
    SubnetMask = (Get-AWSSubnetMask -SubnetCIDR $PrivateSubnet2CIDR)
  }

  xDnsServerAddress DnsServerAddress {
    Address = $ADServer1PrivateIp
    InterfaceAlias = 'Ethernet'
    AddressFamily = 'IPv4'
    DependsOn = '[cIPAddress]DC2IPAddress'
  }

  xComputer JoinDomain {
    Name = 'DC2'
    DomainName = $DomainDNSName
    Credential = $Credential
    DependsOn = '[cIPAddress]DC2IPAddress'
  }

  windowsFeature ADDSInstall {
    Ensure = 'Present'
    Name = 'AD-Domain-Services'
    DependsOn = '[xComputer]JoinDomain'
  }

  windowsFeature ADDSToolsInstall {
    Ensure = 'Present'
    Name = 'RSAT-ADDS-Tools'
  }

  xADDomainController ActiveDirectory {
    DomainName = $DomainDNSName
    DomainAdministratorCredential = $Credential
    SafemodeAdministratorPassword = $Credential
    DependsOn = '[windowsFeature]ADDSInstall'
  }
}
```

Figure 13: DC2 Configuration

Since the forest and domain were created by DC1, you can see in Figure 13 that DC2 simply needs to be added to the domain and promoted to a Domain Controller.

Remote Desktop Gateway Configuration

The node configuration for the Remote Desktop Gateway servers is fairly straightforward. The RDGateway service and associated Remote Server Administration Tools (RSAT) are installed on the server, and the server is then joined to the Active Directory domain.

```
Node RDGW1 {
  windowsFeature RDGateway {
    Name = 'RDS-Gateway'
    Ensure = 'Present'
  }

  windowsFeature RDGatewayTools {
    Name = 'RSAT-RDS-Gateway'
    Ensure = 'Present'
  }

  xComputer JoinDomain {
    Name = 'RDGW1'
    DomainName = $DomainDNSName
    Credential = $Credential
    DependsOn = "[xDnsServerAddress]DnsServerAddress"
  }
}
```

Figure 14: RDGW1 Configuration

The configuration for RDGW2 is identical to the one shown in Figure 14 for RDGW1, with the exception of the node name. After the environment has been deployed, you can initiate a Remote Desktop Connection to either gateway using a standard TCP Port 3389 connection. To fully configure the RDGateway role with certificates (so that connections can be made securely over HTTPS) you should follow the additional steps in our [Quick Start Reference Deployment for the Remote Desktop Gateway](#).

Web Server Configuration

The web server configuration is the same for both the WEB1 and WEB2 servers. After each server is joined to the domain, then IIS, ASP.NET, and sample “hello world” web pages are installed on the systems.

```
Node WEB1 {
  xComputer JoinDomain {
    Name = 'WEB1'
    DomainName = $DomainDNSName
    Credential = $Credential
    DependsOn = "[xDnsServerAddress]DnsServerAddress"
  }

  windowsFeature IIS {
    Ensure = 'Present'
    Name = 'Web-Server'
  }

  windowsFeature AspNet45 {
    Ensure = 'Present'
    Name = 'Web-Asp-Net45'
  }

  windowsFeature IISConsole {
    Ensure = 'Present'
    Name = 'Web-Mgmt-Console'
  }
}
```

```

}
File default {
  DestinationPath = "c:\inetpub\wwwroot\index.html"
  Contents = "<h1>Hello world</h1>"
  DependsOn = "[WindowsFeature]IIS"
}
}

```

Figure 15: WEB1 Configuration

The configuration in Figure 15 creates a single IIS website listening on TCP port 80 hosting a single web page. You can navigate to either web server after the deployment to confirm that IIS is working properly.

Generating the MOF Documents

As we've discussed, the pull server runs the configuration script which produces a MOF file for each server in the environment. Each MOF file initially has a basename matching the hostname of the associated server. For example, the first Domain Controllers file will be named DC1.MOF.

In order for client nodes to pull these configurations, the files must be renamed using the nodes ConfigurationID, which is the GUID stored in the CloudFormation template, and tagged on the Amazon EC2 instance.

```

foreach($mofFile in $mofFiles) {
  $guid = ($ConfigurationData.AllNodes | Where-Object {$_.NodeName -eq $mofFile.BaseName}).Guid
  $dest = "$env:ProgramFiles\windowsPowerShell\DscService\Configuration\$($guid).mof"
  Move-Item -Path $mofFile.FullName -Destination $dest
  New-DSCChecksum $dest
}

```

Figure 16: Code Snippet

The snippet shown in Figure 16 is the code at the end of the configuration script that runs on the pull server and generates the MOFs, renames the files, and moves them to the appropriate folder. The configuration name is ServerBase, and we store the result of running the configuration in the \$mofFiles variable, which will store the collection of file system objects representing each MOF file. We then simply loop through the list of files, matching the GUID (ConfigurationID) with each node name, and renaming the file. Finally, we move the files to the Configuration folder where they are checksummed and ready for download.

Deploy the Pull Mode Stack

To launch the PowerShell DSC Pull Server infrastructure AWS CloudFormation template into the US West (Oregon) region, [launch the Quick Start](#).

Note

You are responsible for the cost of the AWS Services used while running this Quick Start Reference Deployment. The Cost for creating and running the template with default settings is approximately \$3.50 an hour. See the pricing pages of the AWS services you will be using for full details.

The DSC Pull Server infrastructure template allows for rich customization of 15 defined parameters at template launch. The template parameters include the following default values:

Parameter	Default	Description
KeyPairName	<User Provided>	Public/private key pairs allow you to connect securely to your instance after it launches.
AdminPassword	<User Provided>	Password for the administrator user account. This must be a complex password that's at least 8 characters long.
ADServer1PrivateIP	10.0.0.10	Fixed private IP for the first Active Directory server located in AZ1
ADServer2PrivateIP	10.0.64.10	Fixed private IP for the second Active Directory server located in AZ2
DMZ1CIDR	10.0.32.0/20	CIDR block for the Public subnet located in AZ1
DMZ2CIDR	10.0.96.0/20	CIDR block for the Public subnet located in AZ2
PrivateSubnet1CIDR	10.0.0.0/19	CIDR block for the Private Subnet 1 located in AZ1
PrivateSubnet2CIDR	10.0.64.0/19	CIDR block for the Private Subnet 2 located in AZ2
DomainDNSName	Example.com	DNS Domain for the AD Domain (example.com)
DomainNetBiosName	EXAMPLE	Netbios name for the domain (EXAMPLE)
NATInstanceType	t2.small	Amazon EC2 Instance type for the NAT Instances
PullServerInstanceType	t2.medium	Amazon EC2 Instance type for the Pull Server Instances
Pullserver1PrivateIP	10.0.0.15	Fixed private IP for the first DSC Pull server located in AZ1
Pullserver2PrivateIP	10.0.64.15	Fixed private IP for the second DSC Pull server located in AZ2
VPCCIDR	10.0.0.0/16	CIDR block for the VPC.
WindowsInstanceType	t2.xlarge	Amazon EC2 Instance type for the Windows Instances

You can modify these parameters, change the default values, or, if you choose to edit the code of the template itself, create an entirely new set of parameters based on your specific deployment scenario.

Testing Configuration Drift Resistance

After the deployment is complete, you can confirm that the state of each system will retain its desired configuration. One way to test this is to perform the following steps:

1. Open a web browser and navigate to either <http://web1> or <http://web2>. You'll see a placeholder web page that was installed by the DSC configuration for the web servers.
2. Delete the index.html file from c:\inetpub\wwwroot on the web server.
3. Refresh the browser to confirm that you can no longer view the page.
4. After 15 minutes, refresh the browser again to confirm that the state of the system has been re-applied and the system is resistant to configuration drift issues.

If you do not want to wait, you can force DSC to connect to the pull server and apply the current configuration using the Update-DscConfiguration cmdlet.

Deployment with a Push Server Infrastructure

In addition to deploying a pull server infrastructure, we've also developed an AWS CloudFormation template that deploys the reference architecture in "push" mode. This means that we do not deploy pull servers or an Elastic Load Balancer, and that each instance will utilize an individual configuration script which is specific to that server.

As each instance is bootstrapped, a configuration script is downloaded, run (producing a MOF file), and pushed locally to configure the server. The node configurations for each instance are almost identical to those we've looked at in the pull model, with the exception that each node configuration now resides in an individual configuration script.

Figure 17 shows the architecture diagram for the push mode infrastructure.

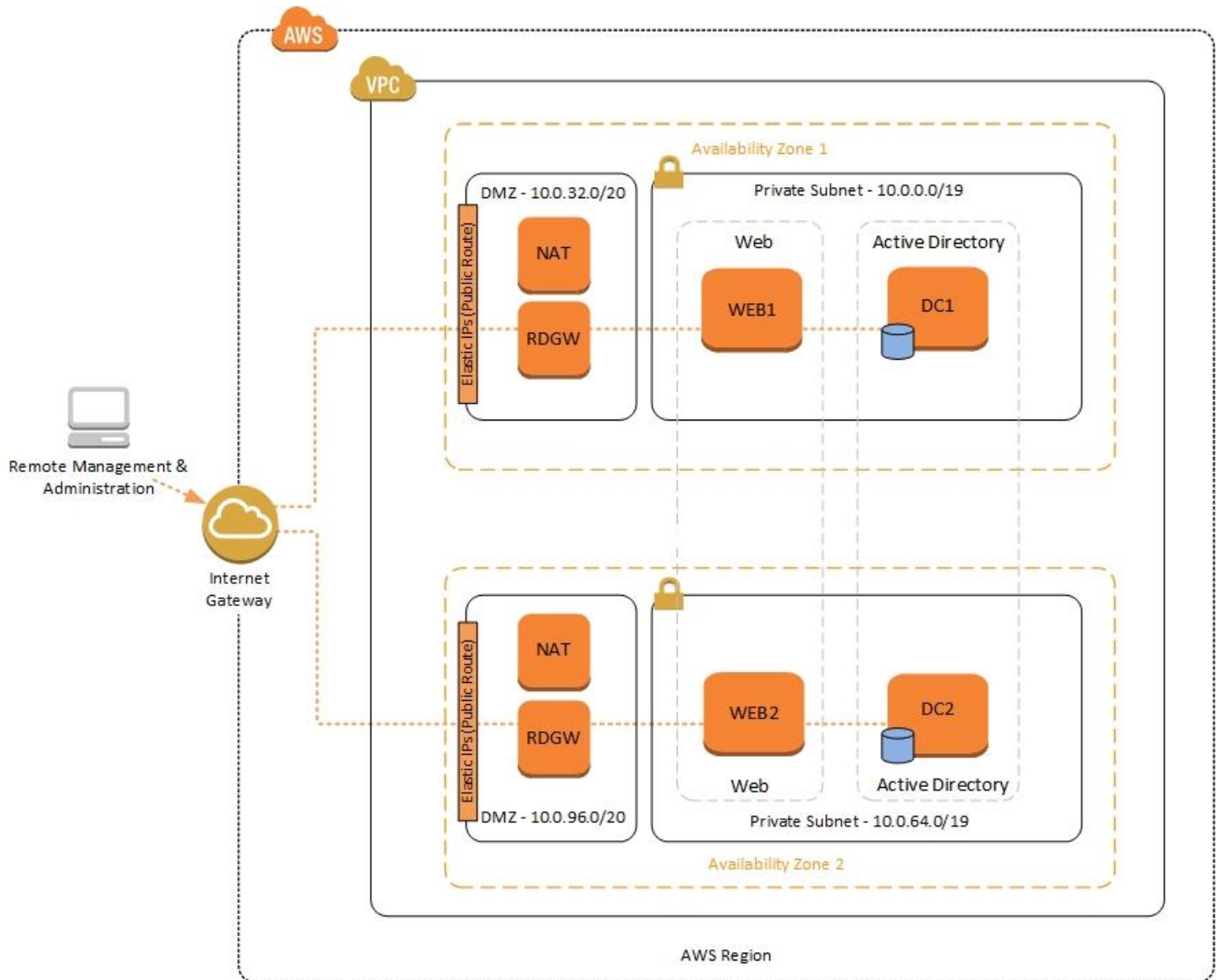


Figure 17: Reference Architecture Deployed by the DSC “Push” Mode Stack

Deploy the Push Mode Stack

To launch the PowerShell DSC Push mode infrastructure AWS CloudFormation template into the US West (Oregon) region, [launch the Quick Start](#).

Note

You are responsible for the cost of the AWS Services used while running this Quick Start Reference Deployment. The cost for creating and running the template with default settings is approximately \$3.50 an hour. See the pricing pages of the AWS services you will be using for full details.

The template parameters for the push mode template are similar to those in the pull mode template.

Additional Resources

- Microsoft on AWS:
 - <http://aws.amazon.com/microsoft/>
- Bootstrapping Applications via AWS CloudFormation:
 - <https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf>
- Bootstrapping AWS CloudFormation Windows Stacks:
 - <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-windows-stacks-bootstrapping.html>
- Amazon EC2 Windows Guide:
 - <http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/>
- AWS Windows and .NET Developer Center:
 - <http://aws.amazon.com/net>
- Microsoft License Mobility:
 - <http://aws.amazon.com/windows/mslicenseability>
- DSC topics on the PowerShell Team Blog:
 - <http://blogs.msdn.com/b/powershell/archive/tags/dsc>
- Whitepapers, Books, and Videos:
 - Active Directory Reference Architecture
<http://aws.amazon.com/microsoft/whitepapers/ad-reference-architecture/>
 - Remote Desktop Gateway Reference Architecture
<http://aws.amazon.com/microsoft/whitepapers/rdgateway-reference-architecture/>
 - Securing the Microsoft Platform on AWS
http://media.amazonwebservices.com/AWS_Microsoft_Platform_Security.pdf
 - The DSC Book
<http://powershell.org/wp/ebooks>
 - A Practical Overview of Desired State Configuration
<http://channel9.msdn.com/events/teched/northamerica/2014/DCIM-B417>

Send Us Your Feedback

Please post your feedback or questions on the [AWS Quick Start Discussion Forum](#).

Document Revisions

Date	Change	In section
March 2015	Optimized the underlying Amazon VPC design to support expansion and to reduce complexity.	Architecture diagram and template updates
November 2014	In the sample template, changed the default type for NATInstanceType to t2.small to support the EU (Frankfurt) region.	Deploy the Pull Mode Stack (template parameters table)

© 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.