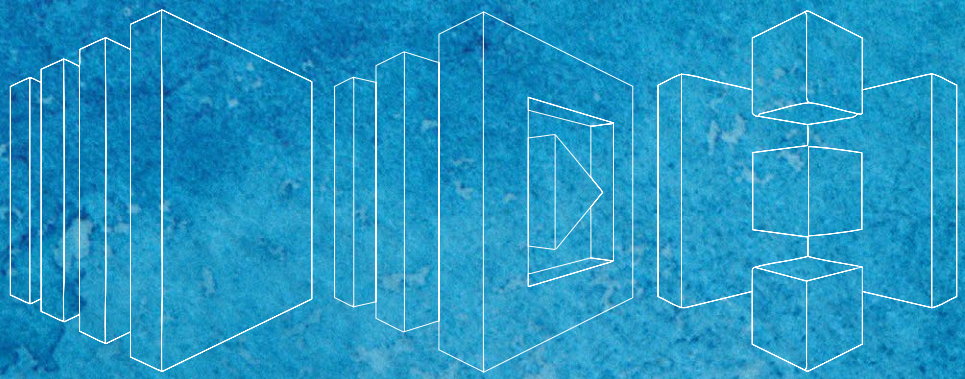
HELECLOUD

Windows EC2 Instances: **Secure Offline Active Directory Join Using Lambda & SSM**

Saturday, 3 June, 2017



HELECLOUD

Contents

About the Author

Dimitar Vasilev is a Principal Consulting with HeleCloud. He is a successful, well-connected technology and business practitioner. Throughout his more than twenty years in business Dimitar has:

- Co-founded two successful startups;
- Established the foundations of the Unified Communications community in Bulgaria as a member of the Consulting and later in Sales of the Microsoft Bulgaria team;
- Worked in challenging and demanding technology and business context in Europe and the Middle East.

HELECLOUD

AD Join Without Username and Password

Problem statement

In very broad terms, joining Windows computers to Active Directory (AD) requires a privileged domain user to use their username and password. Manually configuring instances in this way is often impractical at scale. On the AWS platform, it is possible to automate the process by using services such as AD Connector; however, such services are not available in all AWS regions, and also pose certain limitations. Scripting the domain join process is desirable but securing the credentials that will be used by the script and managing these credentials (consider password expiration, user rights assignment, etc) add additional complexity to the process, and limit its flexibility.

This whitepaper provides a secure mechanism for joining Amazon EC2 Windows instances to Active Directory domains in a scripted and fully automated fashion.

Automation and Bypass Ability

Background

As of Windows Server 2008 R2, there is a way to join Windows computers (server and client) to AD using the Offline Domain Join method (see TechNet article here).

An additional step to fully automate the process is to dynamically find the nearest Domain Controller (DC) for the client to use to join the domain. To allow for exceptions from the rule for instances that shouldn't join the AD domain, an EC2 tag can be used (we've gone for the "DoNotJoin" = "true" tag in this whitepaper); this helps to manage cases, where the instance is already an AD member (migration from on-premises or another AD environment, for example), or where one is working with standalone servers and AD membership is not required.
We will focus on the high-level approach in this article. Further automation, as well as more detailed security configuration would likely need to be considered in a well-designed production environment.

Offline Domain Join Method for EC2 Instances

Assumptions

- In this whitepaper, we assume that both the Domain Controller and the Domain Candidate are in the same AWS VPC (and thus - in the same region). In a more general setting, it is possible to configure specific IAM roles, S3 permissions and connectivity to enable domain join for Domain Controllers and Domain Candidates that are in different AWS regions/VPCs, using S3 buckets that reside in the same region, or in a different

region and AWS account;

- Both the Domain Controller and the Domain Candidate need to be supported Windows instances (2008 R2 or later);
- The Python code provided uses version 3.6 conventions. The code may need to be modified for other Python versions.

Requirements

- CloudTrail needs to be enabled and a CloudWatch rule needs to be configured to detect new instance creation/launch and invoke a Lambda function in every region where we want to add instances to the domain;
- Systems Manager (SSM) should be available in the region. SSM service availability per region can be confirmed here;
- The correct EC2 role needs to be applied to both the Domain Controller and the Domain Candidate in order to use Systems Manager (SSM). The EC2 Role needs to provide write permissions (for the Domain Controller) and read permissions (for the Domain Candidate) to the S3 bucket where the offline domain join files will be stored. In this whitepaper, we simplify the configuration and use a single EC2 role that allows for both read and write access;
- DNS needs to be configured on the instances (via DHCP options, manually, or via the EC2 userdata parameters) so that they can access AD DNS. DNS zone replication, forwarding and delegations may need to be configured in a hybrid environment.

High Level Process Overview

1. The process starts when a new EC2 instance is launched. A CloudWatch rule for EC2 instance launch will trigger a Lambda function. This function will first check whether the newly launched instance has the "DoNotJoin" tag set to "true", and if this is the case, no additional actions are performed. If the tag does not exist, the Lambda script will run a local PowerShell script on the Domain Candidate (using SSM), which will confirm domain membership for the newly-launched instance. If it is not a domain member, the local PowerShell script will look for the nearest DC to this new domain candidate.

Tip: Instead of providing the domain name as a parameter, configuring DHCP options should be considered and the script can retrieve domain name automatically. For simplicity, we provide the domain as a parameter to the Lambda function and PowerShell scripts in this whitepaper.

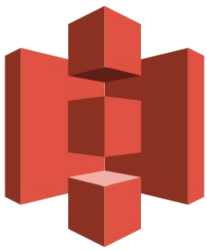
To effectively find the nearest DC, the script will expect that subnets, sites, and site links have been configured in Active Directory Sites and Services, that there is a healthy AD replication topology in place, and that in general AD is in a healthy state. The discovery process finishes by providing JSON-formatted data needed for the next steps, including the domain candidate FQDN, as well as the IP address and name of nearest domain controller.

2. The Lambda function has already received the Domain Controller FQDN in Step 1, however it still needs to find the EC2 instance ID for it in order to be able to run commands on it via SSM. The Lambda function therefore finds the instance ID for the nearest DC, and schedules the **djoin** command on it using the EC2 Systems Manager (SSM). Having executed the **djoin** command, the Lambda function uses SSM to transfer the output file to the S3 bucket, and returns the status and the file name in JSON format;

3. The Lambda function uses SSM to run a command on the domain candidate to join the domain providing the previously saved **djoin** output file (stored on S3) as a parameter. Upon success, the member candidate will reboot, and start as a new domain member.

Technical implementation

The following sections provide more information on the technical aspects of the solution.



Create an S3 bucket

You need to create an S3 bucket that will be used by Domain Controller and Domain Candidates to establish domain membership by storing offline domain join files in this bucket. For the purposes of this whitepaper, all files are stored in the bucket **djoin-bucket. DJoin** is a folder in the **djoin-bucket.** bucket.

All three PowerShell scripts (see the Appendices) are stored in the **djoin-bucket.** bucket.

Create an EC2 IAM role

It is recommended that separate roles be used for the Domain Controller and the Domain Candidate in a production environment.

Create a new IAM role for Active Directory domain controllers and candidates; let's assume the name of the role is **EC2-Role-DJoin**. Attach the standard AWS IAM policies **AmazonEC2RoleforSSM** and **AmazonS3FullAccess** to this role. Note that we need to be able to read from and write to the S3 bucket: the Domain Controller will need to write a new file to S3, and the Domain Candidate needs to be able to read this file from S3. For the purposes of this whitepaper, having such broad permissions is acceptable, however it is recommended that much more specific permissions be used in a production environment. Here, we'll use the same role for both the Domain Controller and the Domain Candidate, however it is recommended that separate roles be used for the Domain Controller and the Domain Candidate in a production environment.

HELECLOUD

Install the PowerShell Modules for AD

Sometimes the installation of PowerShell AD components may take longer, and may lead to timeouts in the Lambda function. It is recommended to preinstall PowerShell modules on all Windows instances by adding the following lines to the EC2 userdata parameter to install them at launch and optimise next steps:

For 2008R2:

```
Import-Module Servermanager
Add-WindowsFeature RSAT-AD-PowerShell
```

For 2012 and later:

```
Install-WindowsFeature -Name RSAT-AD-PowerShell
```

The Lambda code that we are offering in this whitepaper contains debugging and tracing messages to help with the troubleshooting.

Create a Role for the Lambda Function

Create a new IAM role for the Lambda function – let's call it **Lambda-Role-DJoin.** We'll use the following IAM policy for the Lambda script:

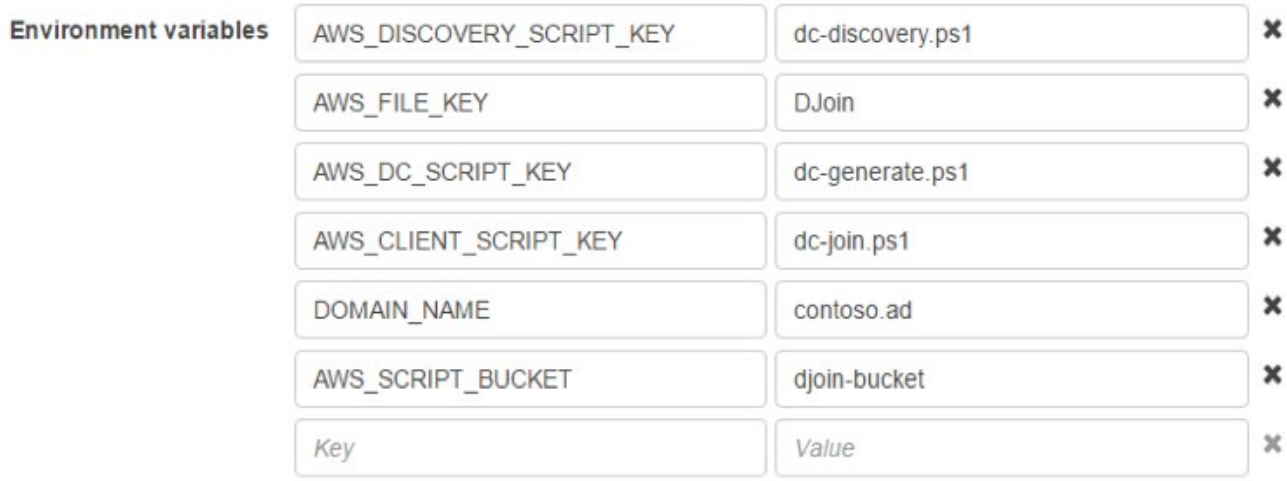
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "s3:GetObject",
                "ssm:SendCommand",
                "ssm:GetCommandInvocation",
                "ssm:DescribeInstanceInformation"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
}
```

HELECLOUD

Lambda Function Configuration and Code

Create a Lambda function using the code below by selecting the Python 3.6 runtime. In this whitepaper, we've called the Lambda function OfflineDomainJoin. Instance startup may take time, so it's recommended to configure the Lambda timeout to 3 minutes or longer.

The Lambda function requires the following parameters:



The Lambda function code can be found below.

Create a CloudWatch Rule

Create a new CloudWatch rule for **EC2 Instance State-change Notification;** the trigger should be configured to activate when the instance enters the running state. The following CloudWatch event pattern can be used:
Specify the **OfflineDomainJoin** Lambda function as the target, so that it be triggered every time a new EC2 instance is launched in the region:

```
{
  "source": [
    "aws.ec2"
  ],
  "detail-type": [
    "EC2 Instance State-change Notification"
  ],
  "detail": {
    "state": [
      "running"
    ]
  }
}
```

HELECLOUD

Ideas for Future Extensions and Features

- Use more restrictive IAM roles and policies to access specific resources only.
- Instead of waiting for SSM commands to complete, use SNS notifications to invoke Lambda code again when task has finished.
- Renaming the instance: if the instance needs to be renamed, it's better to do so before joining the domain. One of the possible solutions is to check for a particular AWS tag with the desired new name, and rename the instance after a Domain Controller has been found.
- Remove instances from AD on termination: this is possible by name (assuming that the EC2 instance has a name tag that corresponds to its host name), or by finding its private IP address in VPC, and querying DNS for the host name via reverse name resolution (PTR records), and then deleting it.

HELECLOUD

APPENDIX

The source code referenced in this whitepaper can also be found in this public BitBucket repository.

Lambda Code (Python 3.6)

```
import boto3
import time
import json
import os

# Global variables - please provide these environment variables for Lambda
script_bucket = os.getenv("AWS_SCRIPT_BUCKET")
file_store = os.getenv("AWS_FILE_KEY")
discovery_script_file = os.getenv("AWS_DISCOVERY_SCRIPT_KEY")
client_script_file = os.getenv("AWS_CLIENT_SCRIPT_KEY")
dc_script_file = os.getenv("AWS_DC_SCRIPT_KEY")
domain = os.getenv("DOMAIN_NAME")

def lambda_handler(event, context):

    print("OfflineDomainJoin starting:")
    print(event)

    instance_id = event['detail']['instance-id']
    region =  event['region']

    print("Instance ID: ", instance_id)
    print("Region: ", region)

    ec2Res = boto3.resource('ec2', region_name=region)

    instance = ec2Res.Instance(instance_id)
#    print ("Instance Tags: ", instance.tags)
    if hasattr(instance, 'tags'):
        name_tag = [ tag[ 'Value' ] for tag in instance.tags if str(tag[ 'Key' ]).lower()
== 'donotjoin' ]
        if ''.join(name_tag).lower() == 'true':
            print("Found DoNotJoin tag. Lambda will quit.")
            return

    ssmClient = boto3.client("ssm", region_name=region)
    s3 = boto3.resource("s3")

#------------------------ Discovery BEGIN -------------------------
    script = get_s3_text_content(s3, script_bucket, discovery_script_file)
    discovery_result = discovery_dc(script, ssmClient, instance_id, domain)

    if discovery_result == None:
        print("Discovery results missing - check instance, role and SSM service status.
Lambda will quit.")
        return
    SSM_output = json.loads(discovery_result['StandardOutputContent'])
    SSM_status = SSM_output['Status']

    print("Discovery result (all): ", SSM_output)
    print("Discovery result (status): ", SSM_status)

    if SSM_status.lower() == 'success' and discovery_result['StandardErrorContent'] == '':
        print("DC %s (%s)" % (SSM_output['HostName'], SSM_output['IPv4Address']))
    else:
        print("DC discovery failed (%s)" % (discovery_result['StandardErrorContent']))
        print("Lambda will quit.")
        return
#------------------------ Discovery END -------------------------

#-------- find dc instance_id by FQDN & IP and generate DJoin file -----------
    dc_instance_ids = find_dc(ec2Res, SSM_output['HostName'], SSM_output['IPv4Address'])
```

```
    print("Found %s DC(s)" % (len(dc_instance_ids)))

    script = get_s3_text_content(s3, script_bucket, dc_script_file)
    generation_result = generate_offline_file(ssmClient, SSM_output, script, dc_instance_ids,
script_bucket, file_store)

    if generation_result == None:
        print("File generation results missing - check instance, role and SSM service
status. Lambda will quit.")
        return
    SSM_output = json.loads(generation_result['StandardOutputContent'])
    SSM_status = SSM_output['Status']

    print("File generation result (all): ", SSM_output)
    print("File generation result (status): ", SSM_status)

    if SSM_status.lower() == 'success' and generation_result['StandardErrorContent'] == '':
        print("File generation was successfull: %s" % (SSM_output['Description']))
        print("Domain join bucket is (%s)" % (SSM_output['DJoinBucket']))
        print("Domain join key is (%s)" % (SSM_output['DJoinKey']))
        print("Domain join file name is (%s)" % (SSM_output['DJoinFile']))
    else:
        print("File generation failed: %s" % (generation_result['StandardErrorContent']))
        print("Lambda will quit.")
        return
#------------------------ Djoin generation END -------------------------

#--------------------- run generated DJoin file on client and restart ---------
    script = get_s3_text_content(s3, script_bucket, client_script_file)
    DJoinBucket =  SSM_output['DJoinBucket']
    DJoinKey = SSM_output['DJoinKey']
    DJoinFile = SSM_output['DJoinFile']
    join_result = apply_offline_file(ssmClient, script, instance_id, DJoinBucket, DJoinKey,
DJoinFile)

    if join_result == None:
        print("Domain join results missing - check instance, role and SSM service status.
Lambda will quit.")
        return
    SSM_output = json.loads(join_result['StandardOutputContent'])
    SSM_status = SSM_output['Status']

    print("Domain join result (all): ", SSM_output)
    print("Domain join result (status): ", SSM_status)

    if SSM_status.lower() == 'success' and join_result['StandardErrorContent'] == '':
        print("Domain join was successfull: %s" % (SSM_output['Description']))
    else:
        print("Domain join failed: %s" % (join_result['StandardErrorContent']))
        print("Lambda end.")
        return

############################ Main END ##############################

def discovery_dc(script, ssm, instance_id, domain_name):
# Script is expecting variable:
#   Name of the domain - $domain
    script.insert(1, '')
    script.insert(2, '$domain=\"' + domain_name + '\"')
    script.insert(3, '')
    print("Prepare to run discovery script: ", script)
    wait_until_instance_available(ssm, instance_id)
    try:
        ssm_response = ssm.send_command(
            InstanceIds=[
                instance_id,
            ],
            DocumentName='AWS-RunPowerShellScript',
            Comment='(DJoin Step1) DC discovery for {0} domain'.format(domain_name),
            Parameters={
                "commands": script
```

```
            }
        )
    except Exception as ex:
        print("Unable to find instance - Please check instance role\nSSM Send Command
Exception: ", ex)
        return
    command_id = ssm_response['Command']['CommandId']
    wait_for_command_to_complete(ssm, command_id, instance_id)
    result = get_command_output(ssm, command_id, instance_id)

    print("DC discovery result: ", result)
    return result


def get_s3_text_content(s3, s3Bucket, s3Key):
    doc_object = s3.Object(s3Bucket, s3Key)
    doc_string = str(doc_object.get()['Body'].read())
    if doc_string.find("\\r\\n") != -1:
        doc_string = doc_string.replace("\\r\\n", "\n")
    if doc_string.find("\\r") != -1:
        doc_string = doc_string.replace("\\r", "\n")
    if doc_string.find("\\t") != -1:
        doc_string = doc_string.replace("\\t", " ")
    if doc_string.find("\\\\") != -1:
        doc_string = doc_string.replace("\\\\", "\\")
    if doc_string[:2] == "b\'" and doc_string[-1:] == "\'":
        doc_string = doc_string[2:-1]
    doc_string = doc_string.splitlines()

    return doc_string


def wait_for_command_to_complete(ssm, command_id, instance_id):
    while True:
        print ("Waiting 5 seconds for command (%s) to complete..." % command_id)
        time.sleep(5)

        command_info = ssm.get_command_invocation(CommandId=command_id,
InstanceId=instance_id)

        if command_info['Status'] == 'Success':
            return True
        elif command_info['Status'] in ['Failed', 'Cancelled', 'TimedOut', 'Terminated']:
            return False

def get_command_output(ssm, command_id, instance_id):
    command_output = ssm.get_command_invocation(CommandId=command_id, InstanceId=instance_
id)
    return command_output

def find_dc(ec2r, dc_name, dc_ip):
# dc_name will not be used at this point
    filters = [
        {
            'Name':'private-ip-address',
            'Values':[dc_ip]
        }
    ]

    dc_instances = list(ec2r.instances.filter(Filters=filters))
    print("List DCs InstanceIDs: ", dc_instances)
#    print("First DC ID: ", dc_instances[0].id)
    return dc_instances

def generate_offline_file(ssm_client, ssm_output, script, dc_instance_ids, bucket, folder_
for_files):
    client_name = ssm_output['ClientFQDN']
    client_ip =  ssm_output['ClientIP']

    print("Generating djoin file for {0} ({1})".format(client_name, client_ip))
# Script is expecting variables:
```

```
#   Name of the new domain member - $newADClient
#   S3 location (bucket) where to put file - $S3Bucket
#   S3 key (sufolder) location where to put djoin files - $S3Key
    script.insert(1, '')
    script.insert(2, '$newADClient=\"' + client_name + '\"')
    script.insert(3, '$S3Bucket=\"' + bucket + '\"')
    script.insert(4, '$S3Key=\"' + folder_for_files + '\"')
    script.insert(5, '')
    print("Prepare to run generation script: ", script)

    for instance in dc_instance_ids:
# even it's list, dc_instance_ids contain only one item which is
# comming from dc_discovery function
        try:
            instance_id = instance.id
            ssm_response = ssm_client.send_command(
                InstanceIds=[
                    instance_id,
                ],
                DocumentName='AWS-RunPowerShellScript',
                Comment='(DJoin Step2) Generate offline join file for {0} ({1})'.
format(client_name, client_ip),
                Parameters={
                    "commands": script
                }
            )
        except Exception as ex:
            print("Unable to find instance ({0}) - Please check instance role".
format(instance_id))
            print("SSM Send Command Exception: ", ex)
            return

    print("SSM response (djoin file generation): ", ssm_response)
    command_id = ssm_response['Command']['CommandId']
    wait_for_command_to_complete(ssm_client, command_id, instance_id)
    result = get_command_output(ssm_client, command_id, instance_id)

    print("DJoin file generation result: ", result)
    return result

def wait_until_instance_available(ssm, instance_id):
    while True:
        instances_info = ssm.describe_instance_information(
        InstanceInformationFilterList=[
                {
                'key' : 'InstanceIds',
                'valueSet' : [ instance_id ]
                }
            ],
       )
#        print("Describe result: ", instances_info)
        if instances_info['InstanceInformationList'].__len__() != 0:
            print("Instance is ready")
            return
        print("Waiting 10 seconds for instance to become available...")
        time.sleep(10)

def apply_offline_file(ssm_client, script, client_instance_id, bucket, key, file):
# Script is expecting variables:
#   S3 location (bucket) where to put file - $S3Bucket
#   S3 key (subfolder) location where to put djoin files - $S3Key
#   S3 DJoin filename - $S3File
    script.insert(1, '')
    script.insert(2, '$S3Bucket=\"' + bucket + '\"')
    script.insert(3, '$S3Key=\"' + key + '\"')
    script.insert(4, '$S3File=\"' + file + '\"')
    script.insert(5, '')
    print("Prepare to run join script: ", script)
    wait_until_instance_available(ssm_client, client_instance_id)
    try:
        ssm_response = ssm_client.send_command(
```

```
            InstanceIds=[
                client_instance_id,
            ],
            DocumentName='AWS-RunPowerShellScript',
            Comment='(DJoin Step3) Domain join',
            Parameters={
                "commands": script
            }
        )
    except Exception as ex:
        print("Unable to find instance - Please check instance role\nSSM Send Command
Exception: ", ex)
        return
    command_id = ssm_response['Command']['CommandId']
    wait_for_command_to_complete(ssm_client, command_id, client_instance_id)
    result = get_command_output(ssm_client, command_id, client_instance_id)

    print("Domain join result: ", result)
    return result
```

PowerShell file - dc-discovery.ps1

This script checks the Windows Operation System version; there are slight differences between Windows Server 2008 R2, 2012 R2 and 2016. The script also finds the nearest Active Directory Domain Controller using the **Get-ADDomainController** cmdlet. The result is JSON-formatted, and can be easily accessed and managed in Lambda code. The result contain consists of the following variables:

- Status: success or failure. If any of the commands in the script fails, the result from the script is failure;
- HostName – The FQDN of the nearest Domain Controller
- IPv4Address – The IP address of the nearest Domain Controller

In case of an error, a detailed message is provided and recorded in the Lambda logs.

```
# The global $domain variable needs to be added here

function Get-OS(){
                        $sOS =Get-WmiObject -class Win32_OperatingSystem -computername
localhost
                        if ($sOS.Caption -match "(\d{4})"){ return $Matches[0] }
                        else { return 0 }
}

function CheckDomainMembership(){
                        if ((gwmi win32_computersystem).partofdomain -eq $true) { return
$true }
                        else { return $false }
}

############################## Main script start here ##############################

try{
                        if ( CheckDomainMembership ) { throw "Already Domain Member" }
                        try{
                          $DC = Get-ADDomainController -DomainName $domain -Discover
-NextClosestSite
                        } catch { # There was no AD PowerShell modules installed, so we'll
install them. We are leaving them, but you can uninstall them at the end
                          switch ( Get-OS ){
                                2008 {
                                      Import-Module Servermanager
                                      Add-WindowsFeature RSAT-AD-PowerShell | Out-Null
                                }
                                2012 { Install-WindowsFeature -Name RSAT-AD-PowerShell |
Out-Null }
                                2016 { Install-WindowsFeature -Name RSAT-AD-PowerShell |
Out-Null }
                                default { throw "Unsupported OS" }
```

```
                        }
                    }
                    $DC = Get-ADDomainController -DomainName $domain -Discover
-NextClosestSite
                    $oRet = New-Object System.Object
                    foreach( $oProp in $DC.psobject.Properties ){
                      $oRet | Add-Member -type NoteProperty -name $oProp.Name -value
$oProp.Value
                    }
                    $oRet | Add-Member -type NoteProperty -name "Status" -value
"Success"
                    $oRet | Add-Member -type NoteProperty -name "ClientFQDN" -value
([System.Net.Dns]::GetHostByName("localhost").HostName)
                    $oRet | Add-Member -type NoteProperty -name "ClientIP" -value
([System.Net.Dns]::GetHostByName($oRet.ClientFQDN).AddressList.IPAddressToString)

} catch [Exception] {
                    $oRet = New-Object System.Object
                    $oRet | Add-Member -type NoteProperty -name "Status" -value "Failed"
                    $oRet | Add-Member -type NoteProperty -name "Description" -value
$_.Exception.Message
}

Write-Host ( $oRet | ConvertTo-Json )
```

PowerShell file - dc-generate.ps1

This PowerShell script runs on the Domain Controller, discovered in the previous step. It produces a binary file with the offline domain join information and stores it in an S3 bucket for the domain candidate to use to join the domain at the next phase.

```
# The global "$newADClient", "$S3Bucket" & "$S3Key" variables needs to be added here

function CheckDomainMember(){
                    if ( (Get-ADComputer -Filter { Name -eq $newADClient }) -ne $null )
{ return $true }
                    else { return $false }
}

function InstallAWSCLI() {
    if( -not ( Test-Path "C:/Program Files/Amazon/AWSCLI/aws.exe" ) ){
                    try{
            $package_name = "AWSCLI64.msi"
            $DownloadFile = "https://s3.amazonaws.com/aws-cli/${package_name}"
            $OutFile = "${Env:TEMP}\${package_name}"
            wget $DownloadFile -OutFile $OutFile | Out-Null

            if( -not (Test-Path $OutFile)){ throw "Unable to find downloaded AWS CLI" }
        } catch [Exception] { throw "Unable to download AWS CLI: " + $_.Exception.Message }

        try{
            if( (Start-Process -FilePath "msiexec.exe" -ArgumentList "/i $OutFile /quiet"
-Wait -Passthru).ExitCode -ne 0) { throw "Error installing AWS CLI" }
        } catch [Exception] { throw "Unable to install AWS CLI: " + $_.Exception.Message }
    }
}

function InstallAWSPS() {
    if( -not ( Test-Path "C:/Program Files (x86)/AWS Tools/PowerShell/AWSPowerShell/
AWSPowerShell.psd1" ) ){
        try{
            $package_name = "AWSToolsAndSDKForNet.msi"
            $DownloadFile = "http://sdk-for-net.amazonwebservices.com/latest/${package_
name}"
            $OutFile = "${Env:TEMP}\${package_name}"
            wget $DownloadFile -OutFile $OutFile | Out-Null

            if( -not (Test-Path $OutFile)){ throw "Unable to find downloaded AWS PowerShell"
}
        } catch [Exception] { throw "Unable to download AWS PowerShell: " + $_.Exception.
```

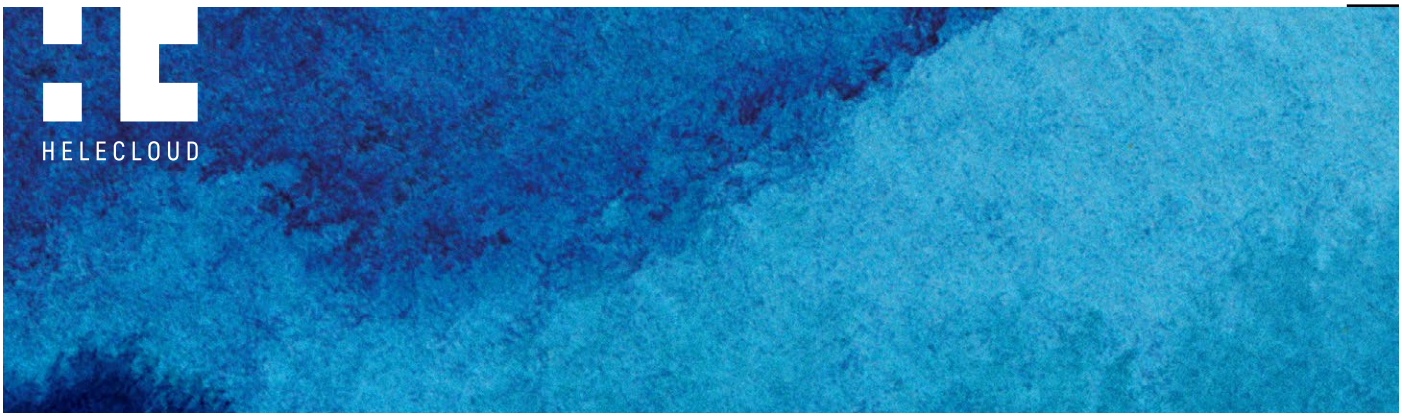
```
Message }

        try{
            if( (Start-Process -FilePath "msiexec.exe" -ArgumentList "/i $OutFile /quiet"
-Wait -Passthru).ExitCode -ne 0) { throw "Error installing AWS PowerShell" }
                            Import-Module "C:/Program Files (x86)/AWS Tools/PowerShell/
AWSPowerShell/AWSPowerShell.psd1"
        } catch [Exception] { throw "Unable to install AWS PowerShell: " + $_.Exception.
Message }
    }
}


try{
                        if( CheckDomainMember ) { throw "Computer $newADClient is already a
domain member"  }
                        $FileName = $newAdClient + "." + (Get-Date).ToString("MM-dd-yyyy-HH-
mm-ss") + ".djoin"
                        $outFile = "${Env:TEMP}\${FileName}"
                        $result = djoin /provision /printblob /domain (get-addomain).DNSRoot
/machine $newAdClient /savefile $outFile
                        if( $result[4].ToLower().IndexOf("success") -eq -1) { throw
$result[4] }

                        InstallAWSCLI
                        InstallAWSPS
                        Import-Module "C:/Program Files (x86)/AWS Tools/PowerShell/
AWSPowerShell/AWSPowerShell.psd1"
                        Write-S3Object -BucketName $S3Bucket -File $outFile -Key
"${S3Key}/${FileName}"

                        $oRet = New-Object System.Object
                        $oRet | Add-Member -type NoteProperty -name "Status" -value
"Success"
                        $oRet | Add-Member -type NoteProperty -name "Description" -value
$result[4]
                        $oRet | Add-Member -type NoteProperty -name "DJoinBucket" -value
$S3Bucket
                        $oRet | Add-Member -type NoteProperty -name "DJoinKey" -value $S3Key
                        $oRet | Add-Member -type NoteProperty -name "DJoinFile" -value
$FileName
} catch [Exception] {
#    Remove-ADComputer -Identity $newADClient -Confirm:$false -ErrorAction Ignore | Out-Null
                        $oRet = New-Object System.Object
                        $oRet | Add-Member -type NoteProperty -name "Status" -value "Failed"
                        $oRet | Add-Member -type NoteProperty -name "Description" -value
$_.Exception.Message
}

Write-Host ( $oRet | ConvertTo-Json )
```

PowerShell file - dc-join.ps1

This PowerShell script runs on the Domain Candidate. It retrieves the binary file for offline domain join from S3 (where it was stored by the **dc-generate.ps1** script) and uses the **djoin** command to convert the domain candidate to a domain member.

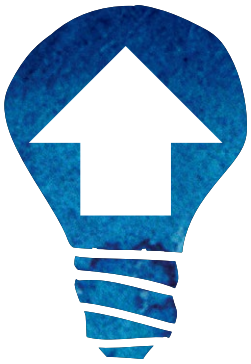
```
# The global "$S3Bucket", "$S3Key" & "$S3File" variables needs to be added here

function InstallAWSPS() {
    if( -not ( Test-Path "C:/Program Files (x86)/AWS Tools/PowerShell/AWSPowerShell/
AWSPowerShell.psd1" ) ){
        try{
            $package_name = "AWSToolsAndSDKForNet.msi"
            $DownloadFile = "http://sdk-for-net.amazonwebservices.com/latest/${package_
name}"
            $OutFile = "${Env:TEMP}\${package_name}"
            wget $DownloadFile -OutFile $OutFile | Out-Null
```

```
            if( -not (Test-Path $OutFile)){ throw “Unable to find downloaded AWS PowerShell”
}
        } catch [Exception] { throw “Unable to download AWS PowerShell: “ + $_.Exception.
Message }

        try{
            if( (Start-Process -FilePath “msiexec.exe” -ArgumentList “/i $OutFile /quiet”
-Wait -Passthru).ExitCode -ne 0) { throw “Error installing AWS PowerShell” }
                            Import-Module “C:/Program Files (x86)/AWS Tools/PowerShell/
AWSPowerShell/AWSPowerShell.psd1”
        } catch [Exception] { throw “Unable to install AWS PowerShell: “ + $_.Exception.
Message }
    }
}

try{
                        InstallAWSPS
                        Import-Module “C:/Program Files (x86)/AWS Tools/PowerShell/
AWSPowerShell/AWSPowerShell.psd1”
                        $localFile = “${Env:TEMP}\${S3File}”

                        Read-S3Object -BucketName $S3Bucket -Key “${S3Key}/${S3File}” -File
$localFile | Out-Null
                        $SystemRoot = $env:SystemRoot
                        $result = djoin /requestODJ /loadfile $localFile /windowspath
$SystemRoot /localos
                        if( $result[4].ToLower().IndexOf(“success”) -eq -1) { throw
$result[4] }

                        $oRet = New-Object System.Object
                        $oRet | Add-Member -type NoteProperty -name “Status” -value
“Success”
                        $oRet | Add-Member -type NoteProperty -name “Description” -value
$result[4]
                        Invoke-Command -ScriptBlock { shutdown /r /f /t 5 /c “Domain join” }
| Out-Null
} catch [Exception] {
                        $oRet = New-Object System.Object
                        $oRet | Add-Member -type NoteProperty -name “Status” -value “Failed”
                        $oRet | Add-Member -type NoteProperty -name “Description” -value
$_.Exception.Message
}

Write-Host ( $oRet | ConvertTo-Json )
```



TAKE THE NEXT STEP:

Let Us Take You on a Complete Journey into the Cloud Environment
We're Putting Together a Wealth of Downloads and Insights into Cloud Computing.



View our thought leadership library

We've been working across numerous organisations and industries; from start-ups to global corporates and government to help them take advantage of the digital transformation. We've been fortunate to enjoy a lot of success in that time, but it took hard work and we've made our share of lessons learned along the way.
What if you could directly benefit from those years of experience – and avoid those mistakes?
What if you could easily keep up with state-of-the-art mechanisms to maximise the potential of your Cloud solutions?
What if you could seamlessly keep up-to-date with the latest cloud developments?

Are you ready to move your business into the future? We are.

Contact our consulting team

About HeleCloud™

HeleCloud™ is an Amazon Web Services technology consultancy with offices in Maidenhead, UK, and Sofia, Bulgaria that helps enterprises of all sizes establish Cloud vision, and execute Cloud strategies through their industry leading Cloud Roadmap methodology. HeleCloud™ also provides Cloud managed services to further amplify Cloud benefits and enable enterprises to focus on their core business and customers. To learn more about how HeleCloud™ is delivering the vision of Cloud, visit http://www.helecloud.com/.
info@helecloud.com

Maidenhead, UK
1 Bell Street, Maidenhead,
Berkshire, SL6 1BU, UK,
+44 20 3286 2227

Sofia, Bulgaria
Sofia Tech Park, 111B
Tsarigradsko Shose Blvd., In-
cubator Bld., Floor 2, Sofia 1784