Deploying an Active Directory Forest

Introduction

Wow, it is amazing how time flies. Almost two years ago, I wrote a set of blogs that showed how one can use Azure Resource Manager (ARM) templates and Desired State Configuration (DSC) scripts to deploy an Active Directory Forest automatically.

Recently, I have been playing with AWS CloudFormation and I am simply in awe by its power. For those that are not familiar with AWS CloudFormation, it is a tool, similar to Azure Resource Manager, that allows you to "code" your computing infrastructure in Amazon Web Services. Long gone are the days when you would have to sit down, pressing each button and choosing each option to deploy your environment. Cloud computing provides you with a way to interface with the fabric, so that you can script the build of your environment. The benefits of this are enormous. Firstly, it allows you to standardise all your builds. Secondly, it allows you to have a live as-built document (the code is the as-built document). Thirdly, the code is re-useable. Most important of all, since the deployment is now scripted, you can automate it.

In this blog I will show you how to create an AWS CloudFormation template to deploy an AWS Elastic Compute Cloud (EC2) Windows Server instance. The template will also include steps to promote the EC2 instance to a Domain Controller in a new Active Directory Forest.

Guess what the best part is? Once the template has been created, all you will have to do is to load it into AWS CloudFormation, provide a few values and sit back and relax. AWS CloudFormation will do everything for you from there on!

Sounds interesting? Lets begin.

Creating the CloudFormation Template

A CloudFormation template starts with a definition of the parameters that will be used. The person running the template (lets refer to them as an operator) will be asked to provide the values for these parameters.

When defining a parameter, you will provide the following

- a name for the parameter
- its type
- a brief description for the parameter so that the operator knows what it will be used for
- any constraints you want to put on the parameter, for instance
    - a maximum length (for strings)
    - a list of allowed values (in this case a drop down list is presented to the operator, to choose from)

- a default value for the parameter

For our template, we will use the following parameters.

```
AWSTemplateFormatVersion: 2010-09-09
Description: CloudFormation Template to deploy an EC2 instance and then install ADDS
Parameters:
Client:
Type: String
Description: Summation of client name - to allow same pattern to be used across
clients
Default: Nivlesh Chandra
Hostname:
Type: String
Description: Hostname - maximum 15 characters
MaxLength: '15'
OS:
Type: String
Description: OS Version
Default: WindowsServer2012R2Base
AllowedValues:
- "WindowsServer2012R2Base"
- "WindowsServer2016Base"
InstanceSize:
Type: String
Description: Instance Size
Default: t2.micro
AllowedValues:
- "t2.micro"
- "t2.small"
- "t2.medium"
DomainDNSName:
Type: String
Description: Fully Qualified Domain Name
Default: mydomain.com.au
DomainNetBiosName:
Type: String
Description: NETBIOS Domain Name
Default: mydomain
```

```
  DomainMode:
    Type: String
    Description: Domain Mode
    Default: Win2012R2
  ForestMode:
    Type: String
    Description: Domain Mode
    Default: Win2012R2
  SafeModeAdministratorPassword:
    MinLength: '8'
    NoEcho: 'true'
    Type: String
    Description: SafeModeAdministrator Password - used when creating ADDS
  Environment:
    Type: String
    Description: Specific Environment - leveraged for AD group creation as well as OU
    location for server objects
    AllowedValues:
    - Dev
    - Test
  AvailabilityZone:
    Type: String
    Description: Default AZ
    AllowedValues:
    - us-east-1a
    - us-east-1b
    Default: us-east-1a
  KeyPair:
    Type: String
    Description: KeyPair Name
    Default: aws_keypair
  Owner:
    Description: Resource Owner - defaults to Niv
    Type: String
    Default: Niv
  Project:
    Type: String
    Description: Tag Value for Project
```

```
    Default: Niv
    Application:
    Type: String
    Description: Tag value for Application - leveraged for Active Directory Group
    creation.
    Default: Lab
    AllowedValues:
    - Lab
    - Learning
    - CustomerSandbox
    S3BucketName:
    Default: s3-cfn-bootstrap
    Description: S3 bucket for boot artefacts
    Type: String
```

view rawCF_Deploying ADDS_Parameters.yaml hosted with by **GitHub**

Next, we will define some mappings. Mappings allow us to define the values for variables, based on what value was provided for a parameter.

When creating EC2 instances, we need to provide a value for the Amazon Machine Image (AMI) to be used. In our case, we will use the OS version to decide which AMI to use.

To find the subnet into which the EC2 instance will be deployed in, we will use the **Environment** and **AvailabilityZone** parameters to find it.

The code below defines the mappings we will use

```
    Mappings:
    AMIMap:
    WindowsServer2012R2Base:
    "ImageId": "ami-2a9a1655"
    WindowsServer2016Base:
    "ImageId": "ami-f0df538f"
    SubnetMap:
    us-east-1a:
    Dev: "subnet-cf2ee1a8"
    Test: "ssubnet-5f25ea38"
    us-east-1b:
    Dev: "subnet-8c70b6a2"
    Test: "subnet-1875b336"
```

view rawCF_Deploying_ADDS_Mappings.yaml hosted with by **GitHub**

The next section in the CloudFormation template is Resources. This defines all resources that will be created.

If you have any experience deploying Active Directory Forests, you will know that it is extremely simple to do it using PowerShell scripts. Guess what, we will be using PowerShell

scripts as well Now, after the EC2 instance has been created, we need to provide the PowerShell scripts to it, so that it can run them. We will use AWS Simple Storage Service (S3) buckets to store our PowerShell scripts.

To ensure our PowerShell scripts are stored securely, we will allow access to it only via a certain role and policy.

The code below will create an AWS Identity and Access Management (IAM) role and policy to access the S3 Bucket where the PowerShell scripts are stored.

```
Profile:
Type: 'AWS::IAM::InstanceProfile'
Properties:
Roles:
- !Ref HostRole
Path: /
InstanceProfileName: !Join
- ''
- - 'instance-profile-'
- !Ref S3BucketName
HostRole:
Type: 'AWS::IAM::Role'
Properties:
RoleName: !Join
- ''
- - 'role-s3-read-'
- !Ref S3BucketName
Policies:
- PolicyDocument:
Version: 2012-10-17
Statement:
- Action:
- 's3:GetObject'
Resource: !Join
```

```
- ''
- - 'arn:aws:s3:::'
- !Ref S3BucketName
- '/*'
Effect: Allow
PolicyName: s3-policy-read
Path: /
AssumeRolePolicyDocument:
Statement:
- Action:
- 'sts:AssumeRole'
Principal:
Service:
- ec2.amazonaws.com
Effect: Allow
Version: 2012-10-17
```

view rawCF_Deploying_ADDS_Roles_Policies.yaml hosted with by **GitHub**

We will use cf-init to do all the heavy lifting for us, once the EC2 instance has been created. cf-init is a utility that is present by default in EC2 instances and we can ask it to perform tasks for us.

To trigger cf-init, we will use the **Userdata** feature of EC2 instance provisioning. cf-init, when started, will check the EC2 Metadata for the credentials it will use, and it will also check it for all the tasks it needs to perform.

Below is the metadata that will be used. For simplicity, I have hardcoded the URL to the files in the S3 bucket.

```
Metadata:
'AWS::CloudFormation::Authentication':
S3AccessCreds:
type: S3
buckets:
- !Ref S3BucketName
roleName: !Ref HostRole
'AWS::CloudFormation::Init':
configSets:
config:
- get-files
```

```
- configure-instance
get-files:
files:
'c:\s3-downloads\scripts\Join-Domain.ps1':
source: https://s3.amazonaws.com/s3-cfn-bootstrap/scripts/Join-Domain.ps1
authentication: S3AccessCreds
'c:\s3-downloads\scripts\Add-WindowsComponents.ps1':
source: https://s3.amazonaws.com/s3-cfn-bootstrap/scripts/Add-WindowsComponents.ps1
authentication: S3AccessCreds
'c:\s3-downloads\scripts\Configure-ADForest.ps1':
source: https://s3.amazonaws.com/s3-cfn-bootstrap/scripts/Configure-ADForest.ps1
authentication: S3AccessCreds
configure-instance:
commands:
1-set-powershell-execution-policy:
command: >-
powershell.exe -Command "Set-ExecutionPolicy UnRestricted -Force"
waitAfterCompletion: '0'
2-rename-computer:
command: !Join
- ''
- - >-
- powershell.exe -Command "Rename-Computer -Restart -NewName "
- !Ref Hostname
waitAfterCompletion: forever
3-install-windows-components:
command: >-
powershell.exe -Command "c:\s3-downloads\scripts\Add-WindowsComponents.ps1"
waitAfterCompletion: '0'
4-install-ADForest:
command: !Join
- ''
- - >-
- powershell.exe -Command "c:\s3-downloads\scripts\Configure-ADForest.ps1 -
DomainName '
- !Ref DomainDNSName
- ''' -DomainNetBiosName '''
- !Ref DomainNetBiosName
```

```
      - ''' -DomainMode '''
      - !Ref DomainMode
      - ''' -ForestMode '''
      - !Ref ForestMode
      - ''' -SafeModeAdministratorPassword '''
      - !Ref SafeModeAdministratorPassword
      - '''"'
    waitAfterCompletion: forever
```

view rawCF_Deploying_ADDS_Metadata.yaml hosted with by **GitHub**

As you can see, I have first defined the role that cf-init will use to access the S3 bucket. Next, the following tasks will be carried out, in the order defined in the configuration set

- get-files
  - it will download the files from S3 and place them in the local directory c:\s3-downloads\scripts.
- configure-instance (the commands in this section are run in alphabetical order, that is why I have prefixed them with a number, to ensure it follows the order I want)
  - It will change the execution policy for PowerShell to unrestricted (please note that this is just for demonstration purposes and the execution policy should not be made this relaxed).
  - next, the name of the server will be changed to what was provided in the Parameters section
  - the following Windows Components will be installed (as defined in the Add-WindowsComponents.ps1 script file)
    - RSAT-AD-PowerShell
    - AD-Domain-Services
    - DNS
    - GPMC
  - the Active Directory Forest will be created, using the Configure-ADForest.ps1 script and the values provided in the Parameters section

In the last part of the CloudFormation template, we will provide the UserData information that will trigger cfn-init to run and do all the configuration. We will also tag the the EC2 instance, based on values from the Parameters section.

For simplicity, I have hardcoded the security group that will be attached to the EC2 instance (this is defined as **GroupSet** under **NetworkInterfaces**). You can easily create an additional parameter for this, if you want.

Finally, our template will output the instance's hostname, environment it has been created in and its privateip. This provides an easy way to identify the EC2 instance once it has been created.

Below is the last part of the template

```
Properties:
DisableApiTermination: 'false'
AvailabilityZone: !Sub "${AvailabilityZone}"
InstanceInitiatedShutdownBehavior: stop
IamInstanceProfile: !Ref Profile
ImageId: !FindInMap [ AMIMap, !Ref OS, ImageId ]
InstanceType: !Sub "${InstanceSize}"
KeyName: !Sub "${KeyPair}"
UserData: !Base64
'Fn::Join':
- ''
- - "<powershell>\n"
- "cfn-init.exe "
- " --stack "
- "Ref": "AWS::StackId"
- " --resource Instance"
- " --region "
- "Ref": "AWS::Region"
- " --configsets config"
- " -v \n"
- "cfn-signal.exe "
- " ---exit-code 0"
- " --region "
- "Ref": "AWS::Region"
- " --resource Instance"
- " --stack "
- "Ref": "AWS::StackName"
- "\n"
- "</powershell>\n"
Tags:
- Key: Environment
Value: !Sub "${Environment}"
- Key: Application
Value: !Sub "${Application}"
- Key: Name
Value: !Sub "${Hostname}"
- Key: Owner
```

```
      Value: !Sub "${Owner}"
      - Key: OS
      Value: !Sub "${OS}"
      - Key: Project
      Value: !Sub "${Project}"
      NetworkInterfaces:
      - DeleteOnTermination: 'true'
      Description: Primary network interface
      DeviceIndex: 0
      SubnetId: !FindInMap [SubnetMap, !Ref AvailabilityZone, !Ref Environment]
      GroupSet:
      - sg-b15634f9
      Outputs:
      InstanceId:
      Description: 'InstanceId'
      Value: !Ref Instance
      Export:
      Name: !Sub '${Hostname}-${Environment}-InstanceId'
      InstancePrivateIP:
      Description: 'InstancePrivateIP'
      Value: !GetAtt Instance.PrivateIp
      Export:
      Name: !Sub '${Hostname}-${Environment}-InstancePrivateIP'
```

view rawCF_Deploying_ADDS_Properties_Tags_Outputs.yaml hosted with by **GitHub**

Now all you have to do is login to AWS CloudFormation, load the template we have created, provide the parameter values and sit back and relax.

AWS CloudFormation will take it from here and do everything for you

The complete CloudFormation template is available at –

https://gist.github.com/sunilake/32a285d8fb620957a94f56db1687039b.js

https://gist.github.com/nivleshc/e1527eeb6d6ac7f556ded9f0d261e295.js

https://gist.github.com/sunilake/29d7baaf7afd032bea6f4a92d1c94bc5.js