CloudFormation: Setting up an EC2 Instance

We are going to learn how to set up our EC2 instance with an application. There are many activities we would want to perform on an EC2 instance such as:

-Starting up system services

-Configuring the instance

-Create Users/Groups

-Download and install our application

-Download and install dependencies / packages

- etc.

We will expand on launching a simple EC2 instance. We will do the following.

-Download & install a web server

-Download and install PHP

-Host a “May the Force be with you” PHP script (instead of a boring Hello World script)

-Start the web server, so that we can serve and host the PHP script that we created

---

Setting up an EC2 Instance: UserData

We will explore the UserData property for EC2 instances

We need to hook into the instance, which enables us to perform actions on system startup.

So we need to bootstrap an EC2 Instance

A resource of type AWS::EC2::Instance has a property called UserData

It appears within the template under the properties section of the instance resource like so:

```
EC2Instance:
  Type: AWS::EC2::Instance
  Properties:
    UserData:
```

The UserData property needs to be Base64 encoded.

It works for both Linux and Windows Operating Systems. However, each works slightly differently so it has some variances between them.

The UserData property is designed to run only on the first boot cycle.

It is good to be aware of the time it takes to execute. A lot of work will impact the startup time for your instance.

Let's look at the specifics for the UserData property in Windows

UserData for Windows

The property is executed under local administrative privileges. (Run as local administrator)

You can execute Batch Commands or Powershell scripts

The property is executed by the EC2Config or EC2Launch services, depending on the operating system you are using (So Output logs vary) It is worth noting that the log locations vary between the two services.

Here is an example of how the UserData property looks like in Windows when using Batch Commands

```
<script>
echo Current date and time >> %SystemRoot%\Temp\test.log
echo %DATE% %TIME% >> %SystemRoot%\Temp\test.log
</script>
```

Surrounded by script tags, this script is simply outputting the current date and time into a test log file. The other alternative is Powershell (see below)

```
<powershell>
$file = $env:SystemRoot + "\Temp\" + (Get-Date).ToString("MM-dd-yy-hh-mm")
New-Item $file –ItemType file
</powershell>
```

Encapsulated by Powershell tags, this script is simply creating a new file, which is date time stamped.

The UserData property in Linux has some slight differences.

UserData for Linux

-The script is run under root privileges (so there is no sudo command required)
-It is not run interactively (no user feedback) so you will need to take that into account
-Logs are output to the /var/log/cloud-init-output.log aka the var log directory
-The script needs to start with #! (a hash bang) and the interpreter you would like to use.

For example:
#!/bin/bash
yum update –y ←this will update the packages from the yum repository.
yum install –y httpd ← install the web server
service httpd start ← kicks off the web server

the dash y assumes that yes will be answered if it is prompted.

Below is how it all comes together in YAML format:

```
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      UserData:
        !Base64 |
          #!/bin/bash –xe
          yum update –y
          yum install httpd –y
          service httpd start
```

In the UserData property, we are utilizing the Base64 intrinsic function and also the | (aka the literal block) which allows us to input multiple lines of strings.
In this case, we've done a hash bang of bin and bash as the interpreter
We are then calling on the yum repository to update all packages with a dash y
Then we are installing the web server
And then we are starting the web server

```
#!/bin/bash –xe
yum update –y
yum install httpd –y
service httpd start
```

The above multi-line literal block is Base64 encoded and passed into the UserData property for the EC2Instance

Setting up an EC2 Instance: Helper Scripts

-What we looked at so far is a great starting point. We know from our code as infrastructure journey that Procedural scripting is not ideal as it can get messy very quickly.

-To solve this problem a little more elegantly CloudFormation provides Python based helper scripts to optimize this. These helper scripts make CloudFormation a lot more powerful and enable us to fine tune templates for our use case.

-And this comes pre-installed with Amazon Linux AMIs.

-These scripts are not directly executed automatically so you will need to call upon them in your template when you want to use them. We will see an example of this later in the lesson.

-They are updated periodically, and can be updated using the yum repository:
yum install –y aws-cfn-bootstrap

There are four CloudFormation helper scripts which are provided by AWS
The first one is:

cfn-init
    Reads and interprets Metadata to execute AWS::CloudFormation::Init

This helper script reads and interprets the metadata to declaratively bootstrap a resource.

The second one is:

cfn-signal
    Used to signal when a resource or application is ready

This script can be used to signal CloudFormation when a resource is ready.

The third one is:

cfn-get-metadata
    Used to retrieve metadata based on a specific key

This script is used to retrieve metadata based on a specific key

The fourth one is:

cfn-hup
Used to check for updates to metadata and execute custom hooks

This helper script is used to check metadata for updates and triggers custom hooks when this occurs.

Setting up an EC2 Instance: CloudFormation Init

CloudFormation Init is the helper script and the metadata section for EC2 instances within a CloudFormation template

The cfn-init helper script enables us to use:
AWS::CloudFormation::Init

So it enables us to use the CloudFormation Init section in the metadata for the EC2 instance resource

AWS CloudFormation Init section is a declarative approach to setting up our EC2 instances. It is separated into config keys each which contains a set of configuration sections.

```
Resources:
  MyInstance:
    Type: "AWS::EC2::Instance"
    Metadata:
      AWS::CloudFormation::Init:
        config:        ← config key and below are all the config sections
          packages:
          groups:
          users:
          sources:
          files
          commands:
          services:
```

You can have more than one config key if you would like.
Here is an example of a single config key:

AWS::CloudFormation::Init:
config:
packages:
groups:
users:
sources:
files
commands:
services:

When you have a single config key is should simply be called “config”
This enables the helper script to locate it

If you need multiple keys you need to use Configsets

Configsets contain a list of config keys, in a desired execution order

For example let’s say we have a config which installs a web server for us (see below)

installweb:
packages:
yum:
httpd: [] ←in this case we install the httpd package from the yum repository
services:
sysvinit:
httpd:
enabled: true ← and then install and request httpd to be enabled and running on startup
ensureRunning: true

(then we have another config section which installs php and also leverages the yum repository to install it)

installphp:
packages:
yum: ← leverages the yum repository to install it
php: [] ←installs php

CloudFormation now needs to know which order to execute these config keys in. This is what config sets are for. (see below)

```
AWS::CloudFormation::Init:
  configSets:
    webphp:
      -“installphp”
      -“installweb”
```

In this case, we have a config set called webphp which will first call the “installphp” config key and then will call “installweb”

To execute this config set, we pass a webphp to the cfn-init helper script

Config sets allow us to organize our configuration and selectively execute the setup.

Next, let’s take a look at how these sections within the config keys are defined.

packages
Enable us to download and install pre-packaged applications and components

groups
Enable us to create Linux/UNIX groups and to assign group IDs

users
Enable us to create Linux/UNIX users on the EC2 instance

sources
Allow us to download an archive file and unpack it in a target directory on the EC2

files
Allow us to create files on the EC2 instance in a predetermined location

commands
Allow us to execute commands on an EC2 instance

services
Define which services should be enabled or disabled when the instance is launched

Now that we have taken a look at the helper scripts and the CloudFormation Init section, let’s setup our EC2 instance in a hands-on lab.

To summarize how we are going to set up an EC2 instance:

1. UserData
   Is executed on first state (this property is executed whenever an EC2 instance first starts up)
2. CloudFormation Helper Scripts
   In the UserData section, we will call the cfn-init helper script (which will enable us to hook into the AWS CloudFormation Init metadata)
3. AWS::CloudFormation::Init
   This allows us to set up our EC2 instance in a very declarative approach which is a lot more native to CloudFormation and more powerful.

Lab: Setting up an EC2 Instance

We are going to have a closer look at setting up an EC2 instance using the user data, CloudFormation helper scripts, and the CloudFormation Init metadata section.

Copy and Paste the YAML code below into a blank code editor file and save it as CFNInit.yaml

```
Parameters:
  NameOfService:
    Description: "The name of the service this stack is to be used for."
    Type: String
  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access into the server
    Type: AWS::EC2::KeyPair::KeyName
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-1853ac65
    us-west-1:
      AMI: ami-bf5540df
    eu-west-1:
      AMI: ami-3bfab942
    ap-southeast-1:
      AMI: ami-e2adf99e
    ap-southeast-2:
      AMI: ami-43874721
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Metadata:
      AWS::CloudFormation::Init:
        config:
          packages:
            yum:
```

```
          httpd: []
          php: []
        files:
          /var/www/html/index.php:
            content: !Sub |
              <?php print "May the Force be with you!"; ?>
        services:
          sysvinit:
            httpd:
              enabled: true
              ensureRunning: true
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      SecurityGroupIds:
        - !Ref MySecurityGroup
      Tags:
        - Key: Name
          Value: !Ref NameOfService
      KeyName: !Ref KeyName
      UserData:
        'Fn::Base64':
          !Sub |
            #!/bin/bash -xe
            # Ensure AWS CFN Bootstrap is the latest
            yum install -y aws-cfn-bootstrap
            # Install the files and packages from the metadata
            /opt/aws/bin/cfn-init -v --stack ${AWS::StackName} --resource EC2Instance  --region ${AWS::Region}
  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Open Ports 22 and 80
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: 0.0.0.0/0
Outputs:
  Website:
    Description: The Public DNS for the EC2 Instance
    Value: !Sub 'http://${EC2Instance.PublicDnsName}'
```

Let's first explore the User data section which is on line 52 of the above code. Here it is:

```
UserData:
    'Fn::Base64':    ← we need to Base64 encode this string
     !Sub |
      #!/bin/bash -xe
      # Ensure AWS CFN Bootstrap is the latest
      yum install -y aws-cfn-bootstrap
      # Install the files and packages from the metadata
      /opt/aws/bin/cfn-init -v --stack ${AWS::StackName} --resource EC2Instance  --region
```

Here you can see the property which outlines the startup script for our EC2 instance
So we need to Base64 encode the string
We are then utilizing the !Sub intrinsic function which allows us to substitute pseudo parameters into the literal block that we have defined. The next part is a pipe → | which tells CloudFormation that the next section is a literal block, a multi-line block. So everything within the indentation should be passed to the user data property. The following is everything within the indentation so to speak:

```
55          #!/bin/bash -xe
56          # Ensure AWS CFN Bootstrap is the latest
57          yum install -y aws-cfn-bootstrap
58          # Install the files and packages from the metadata
 59         /opt/aws/bin/cfn-init -v --stack ${AWS::StackName} --resource EC2Instance  --region
```

So starting on line 55 have got our literal block. The first line is a hash bang bin bash, the interpreter.

Line 57 is the first line that we are actually executing which is calling on the yum repository to update our AWS CFN Bootstrap to ensure that we have got the latest CloudFormation helper scripts.

Once this is done, on line 59 we step straight into the CloudFormation Init helper script and we execute it. The first parameter we are passing it is dash "-v" which essentially makes the output more verbose.

The second parameter is passing the stack name, so that looks like… --stack ${AWS::StackName}

Then we are passing the resource logical ID to the helper script which enables it to look for the metadata section, so that looks like… --resource EC2Instance

The last parameter we are passing is the region, so that looks like… --region ${AWS::Region}

All of these are using pseudo parameters

So on line 54 where it says !Sub |
The first part is calling an intrinsic function of !Sub which essentially allows us to evaluate the pseudo parameters of name (StackName) and region on line 59. If we didn't use the sub intrinsic function, those pseudo parameters wouldn't be evaluated in CloudFormation. Hence it would just pass the strings that we see on the screen. So we expect this section to update our CloudFormation helper scripts and then execute CFNInit helper script

What the script will do is look for the metadata section which is on line 23 of the YAML code. Per what we learned in the last lesson, the CloudFormation Init section has a config which is the default config set when you only have one config section.

We define the packages that we need to install via the yum repository which is httpd and php

The next section (where it says files:) we are asking CloudFormation to create a file in the specified location of /var/www/html/index.php
The contents of this file will simply be a print "May the Force be with you!" So just a single line of text. On line 34 we've got services, where we are asking to ensure that httpd, which is our webserver, is enabled and it is running on startup. And that is what is going to happen as soon as the instance starts up.

```
Metadata:
    AWS::CloudFormation::Init:
     config:
       packages:
         yum:
           httpd: []
           php: []
       files:
         /var/www/html/index.php:
           content: !Sub |
             <?php print "May the Force be with you!"; ?>
       services:
         sysvinit:
           httpd:
             enabled: true
             ensureRunning: true
```

Now one last section that we have modified since our OutputParameters.yaml example in a past lab/tutorial is the fact that we are opening up the security group. We are going to be starting a

webserver that is going to need an open Port 80 for us to use it. So we have added an ingress rule to the security group to open up port 80 for all IPs.

```
69   - IpProtocol: tcp
70     FromPort: '80'
71     ToPort: '80'
72     CidrIp: 0.0.0.0/0
```

And then for the outputs we are essentially just trying to output a formatted URL which allows us to hit the instance. And we would expect the output of the PHP script, which as we saw earlier if we hit port 80 we should see "May the Force be with you!";

```
73   Outputs:
74     Website:
75           Description: The Public DNS for the EC2 Instance
76           Value: !Sub 'http://${EC2Instance.PublicDnsName}'
```

Let's see the template in action. Switch over to the AWS management console. Select the Sydney Region. Click **Services** and then click **CloudFormation** Then click Create Stack.

Prepare template: ✓Template is ready ← select Template is ready
Specify template: ✓Upload a template file ← select Upload a template file

Click the **Choose file** button. Choose the file that you saved as CFNInit.yaml

Then click **Next**

Stack Name: CFNInitStack ← give it a name like so

KeyName: Test Key Pair ← select the key pair we created in an earlier lab/tutorial
Name of Service: Testing CFNInit ← give it a name of service like so

Click **Next**
This will direct you to the Configure stack options page. Just leave everything as is.
Click **Next**
This will direct you to the Review page. Just leave everything as is.
Scroll down and click the **Create stack** button
The Status will say CREATE_IN_PROGRESS

Wait a minute or two and then click the **refresh** icon in the Events section

The Status will say CREATE_COMPLETE

Now let's check out the EC2 instance that was created by going to the EC2 console.
Click **Services** and click **EC2** and click **Running instances**

✓Select your instance that you named **Testing CFNInit** which is the instance you just provisioned

Click the **Description** tab and scroll down to check the security groups section. Click view inbound rules and you will see that port 80 and port 22 are open to all IPs.

Now let's return to the CloudFormation console.
So click Services and click CloudFormation
Then click on your stack called **CFNInitStack**
Then click the **Outputs** tab
There we should see a URL for our server. Go ahead and click on the URL link which will look something like: http://ec2-13-211-140-19.ap-southeast-2.compute.amazonaws.com ← yours will be different though

Once you click on it, it will say May the force be with you!
So our PHP script has executed, which means our user data step executed a CloudFormation helper script. This looked for the metadata section, installed our web server and PHP, and also put our PHP script that we are seeing now in the right location.

So our EC2 instance has essentially deployed our small PHP script onto our machine on startup.
And that is how we can start setting up our EC2 instances through CloudFormation on provision.

So the difference between our file from our earlier lab called OutputParameters.yaml and our file called CFNInit.yaml is that we added a user data section which executed a CFNInit helper script with the proper parameters. In the metadata section we installed httpd aka the webserver, as well as php. We created a php script that the web server found in the predetermined location /var/www/html/index.php
And lastly, we opened up port 80 on the security group so that we could hit the web server and see our php executed.

Next we will look at chain sets so that we can detect what is going to happen to our stack before we update it.