

CloudFormation: Updating our Stack with ChangeSets

Let's learn how to safely update our stack using CloudFormation Change Sets

Updating a stack can be a risky action, especially in production-like environments

It's important to understand how changes will affect running resources

For example, Let's look at the simple action of renaming an SQS Queue

If our YAML template were to look like the following:

Resources:

MyQueue:

Type: "AWS::SQS::Queue"

Properties:

QueueName: "TestQueue"

So we have one resource which is logically named MyQueue and the queue is named "TestQueue"

And we update our queue name to "ProductionQueue" like so:

Resources:

MyQueue:

Type: "AWS::SQS::Queue"

Properties:

QueueName: "ProductionQueue"

What would happen?

Renaming the SQS Queue will create a new queue and drop the old one in one update process. So it will recreate the resource.

For example if we had TestQueue, CloudFormation will request a new queue to be created named ProductionQueue. Once that is complete, CloudFormation will clean up by removing the (old) TestQueue.

This could be dangerous for many reasons, such as the impact to other systems which interact with this queue.

Do message publishers know which queue to use and when to switch?

Do message consumers know which queue to use and when to switch?

Can messages be lost? All these questions are worth asking before making this one line change.

This is where ChangeSets can help. A ChangeSet allows us to preview how the changes will impact our running resources.

A ChangeSet would inform us before the update that the queue will be replaced with a new queue.

Now let's take a look at what we can do with the ChangeSet.

There are four operations for a ChangeSet:

Create

Create a ChangeSet for an existing stack by submitting a modified template, and / or parameter values. This does not modify the existing stack (aka the existing running resources), and you can create as many as you would like.

View

After creating, you can view proposed changes. So it enables you to view the proposed changes.

Execute

Executing a Change Set updates the changes on the existing stack. So it performs the changes on the existing resources, as described in the ChangeSet.

Delete

You can delete a ChangeSet without performing the changes. So it simply deletes the ChangeSet without performing the updates.

What does a ChangeSet look like?

ChangeSet details are provided to you in a nice table summary in the CloudFormation Management Console. But let's take a look at the detailed data that's available.

The ChangeSet details is an array of resource change data types.
Let's have a look at what makes a resource change data type.

Resource Change

Action

Action taken on the resource (Add | Modify | Remove)

The action that will be taken on the resource can be add, modify, or remove

LogicalResourceId & PhysicalResourceId

Resource's Logical Id (EC2Instance) and Physical Id (i-0a29e20fd8a1b308b)

These are the identifiers for the resource. The logical ID is the name given inside the template.

ResourceType

ResourceType is the type of CloudFormation resource (AWS::EC2::Instance)

Replacement

For Modify action, will CloudFormation delete the old resource and create a new one.

Replacement indicates if a resource will be replaced or simply modified in place.

If true, CloudFormation will delete the old resource and create a new one.

Scope

Which resource attribute is triggering the update

Scope is the attribute of the resource which is triggering the updates

Details

Description of the change to the resource

The Details section provides a description of the change to the resource
such as the property being updated and the causing entity

Examples

Example: Adding a tag to an existing EC2 Instance
(See the JSON below)

```

"resourceChange":{
  "resourceType": "AWS::EC2::Instance",    ←the resource type is an ec2 instance
  "logicalResourceId": "EC2Instance",      ←the logicalResourceId is EC2Instance
  "physicalResourceId": "i-0a29e20fd8a1b308b", ←here is the physical resource Id
  "action": "Modify", ←the action is to modify the EC2 instance
  "replacement": "False", ← it says here that it will NOT have to replace the resource
  "details": [
    {
      "target": {
        "name": null,
        "requiresRecreation": "Never", ← there will be no interruption to the service
        "attribute": "Tags" ←within the details it says the Tags attribute will be modified
      },
      the new tag can thus be added to the resource in place
      "causingEntity": null,
      "evaluation": "Static",
      "changeSource": "DirectModification"
    }
  ],
  "scope": ["Tags"]
}

```

Example: Increase Instance Type of EC2 Instance

We will increase the instance type from T2 micro to a T2 small

(See the ChangeSet below)

```

"resourceChange": {
  "resourceType": "AWS::EC2::Instance",
  "logicalResourceId": "EC2Instance",
  "physicalResourceId": "i-0a29e20fd8a1b308b",
  "action": "Modify",    ← As you can see here the action is still a modify
  "replacement": "True", ← But in this case we are going to need to replace the resource
  "details": [
    {
      "target": {
        "name": "InstanceType",    ← it outlines the instance type as the property which causes update
        "requiresRecreation": "Conditionally",
        "attribute": "Properties"
      },
      "causingEntity": null,
      "evaluation": "Static",
      "changeSource": "DirectModification"
    }
  ],
  "scope": ["Properties"]
}

```

Example: Changing a Security Group on EC2 Instance
 We will assign a new Security Group to an EC2 Instance
 (See the ChangeSet below)

```

"resourceChange": {
  "resourceType": "AWS::EC2::Instance",
  "logicalResourceId": "EC2Instance",
  "physicalResourceId": "i-0a29e20fd8a1b308b",
  "action": "Modify", ← In this example, the action is to Modify
  "replacement": "Conditional", ← However, the replacement is conditional which
  "details": [          means that the resource might be replaced and cannot
  {                    be determined by the template alone
    "target": {
      "name": "SecurityGroupIds",          ←The SecurityGroupIds have been modified
      "requiresRecreation": "Conditionally",      which is a conditional recreation
      "attribute": "Properties"
    },
    "causingEntity": "MySecurityGroup",
    "evaluation": "Static",
    "changeSource": "ResourceReference"
  }
],
"scope": ["Properties"]
}

```

CloudFormation cannot determine with certainty the action that is going to be taken. It is best to refer to the resource types documentation.

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html> ← Here is the link to the AWS resource and property types reference. This indicates the update behaviors of each of the resource type and properties.

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-update-behaviors.html> ← Here is the link to the Update behaviors of stack resources documentation. This is a list of update behaviors that resources can have.

From the documentation for the EC2 Instance type and the security group ID's property in CloudFormation per the screenshot below. We can see that the update requires NO interruption for instances that are in a VPC.

However, if the instance is not in a VPC then replacement is required. So the action depends on the resource and cannot be determined by CloudFormation alone.

Update requires:

- *Update requires:* **No interruption** for instances that are in a VPC.
- *Update requires:* **Replacement** for instances that are not in a VPC.

This scenario demonstrates, CloudFormation does its best to predict the actions that will be taken. But occasionally it will not be able to determine whether there will be a replacement or the change will be in place. In such cases, it is best to check the official documentation for the properties being modified in order to figure it out for yourself.

Let's do a lab/tutorial to see how ChangeSets can help us.

LAB: Updating our Stack with ChangeSets

We will introduce a change of modifying the security group to the previous template that we saved and utilized called CFNInit.yaml

Below is the YAML code for that template:

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}'

Now we will introduce a change of modifying the security group. So our new template will simply have a modification to the security group. Copy and paste the YAML code below into a code editor and save the file as ChangeSet-SGIngressRule.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 ${AWS::Region}
  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Open Ports 22 and 80
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '80'
          ToPort: '80'
          CidrIp: 0.0.0.0/0
  Outputs:
    Website:
      Description: The Public DNS for the EC2 Instance
      Value: !Sub 'http://${EC2Instance.PublicDnsName}'
```

Here is the difference between the old template and the new template

Old template

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

New template

SecurityGroupIngress:

- IpProtocol: tcp
FromPort: '80'
ToPort: '80'
CidrIp: 0.0.0.0/0

The difference is that we have removed a security group ingress rule of opening up port 22. So the only port that should be open is port 80. That is the **ONLY** change that we are making in this template.

So if we make this change. Do we know what is going to happen? Is it going to replace a security group? Is it going to replace the EC2 instance?

Let's switch over to the AWS management console to find out and create a ChangeSet and see what it thinks is going to happen.

★★★ First launch the template that you created in the last lab/tutorial called CFNInit.yaml (Unless you still have it launched and never deleted it after the last lab/tutorial ended. If you already have it launched then skip ahead)

Go 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 update our stack with the file called **ChangeSet-SGIngressRule.yaml**

But first we will create a ChangeSet for our current stack called CFNInitStack

Click **Services** and click **CloudFormation**

✓ Select the CFNInitStack

Then click the **Stack actions** button

Then click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file

Click the **Choose File** button

★★★ Choose the **ChangeSet-SGIngressRule.yaml** file because we are creating a ChangeSet that is going to compare our current template to the template that we plan to update to which is **ChangeSet-SGIngressRule.yaml**

Click **Next**

KeyName: Test Key Pair ← leave as such

NameOfService: Testing CFNInit ← leave as such

Click **Next**

This will direct you to the Configure stack options page

Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.

Just scroll down and click **Create change set**

Change set name: UpdatingIngressRule ← give the change set a name like so

Then click **Create change set**

Wait about a minute and the Status should say CREATE_COMPLETE

So basically this is doing a dry run to figure out what actions are going to be performed if the template were to be updated the way we specified. So none of the changes are actually being performed it is just a dry run.

Once the ChangeSet has been successfully created

You will see in the changes section there is an action to modify the MySecurityGroup. And the Replacement is False which means it is simply going to update it in place. So it will NOT create a new resource as a replacement.

(Below is a screenshot for clarification)

The screenshot displays the AWS CloudFormation console for a change set named 'UpdatingIngressRule'. The interface includes tabs for 'Changes', 'Input', 'Template', and 'JSON changes'. The 'Overview' section provides details about the change set, including its ID, description, and status. The 'Changes (1)' section lists the specific actions to be performed, showing a 'Modify' action on the 'MySecurityGroup' resource.

Action	Logical ID	Physical ID	Resource type	Replacement
Modify	MySecurityGroup	CFNInit-Stack-MySecurityGroup-1DM52W8EGC4BA	AWS::EC2::SecurityGroup	False

Click the JSON changes tab to see the changes in JSON which are as follows:

```
[
  {
    "resourceChange": {
      "logicalResourceId": "MySecurityGroup",
      "action": "Modify",
      "physicalResourceId": "CFNInitStack-MySecurityGroup-1DM52W8EGC4BA",
      "resourceType": "AWS::EC2::SecurityGroup",
      "replacement": "False",
      "details": [
        {
          "target": {
            "name": "SecurityGroupIngress",
            "requiresRecreation": "Never",
            "attribute": "Properties"
          },
          "causingEntity": null,
          "evaluation": "Static",
          "changeSource": "DirectModification"
        }
      ],
      "scope": [
        "Properties"
      ]
    },
    "type": "Resource"
  }
]
```

Here we can see the logical resource ID that we are changing which is MySecurityGroup

We are performing a modify action.

The replacement is false, so it is not going to replace the existing security group.

And the details are the target property that is being changed is the SecurityGroupIngress

So this change should be fairly straightforward.

We can execute the changes (which means the stack will be updated) or we can delete the ChangeSet.

Let click the **Execute** button

Wait a minute and then click the refresh icon in the Events section and it should say UPDATE_COMPLETE

Let's open the EC2 Console to confirm the changes were made to the security group as we desired.

Click **Services** and then click **EC2**

Click **Running Instances**

✓ Select Testing CFNInit

And click the description tab and within the description where it says Security Groups click **View inbound rules**

There you will see that only port 80 is open. So we closed off port 22 per the change that we made to our template. We had no interruption to any of our services.

Now in the next change we want to modify the security group description. Copy and paste the below YAML code into a blank code editor file and save the file as ChangeSet-SGDescription.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 ${AWS::Region}
MySecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Open Port 80 for website
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: 0.0.0.0/0
Outputs:
  Website:
    Description: The Public DNS for the EC2 Instance
    Value: !Sub 'http://${EC2Instance.PublicDnsName}'
```

So here is the difference between the old template and the new template. All we have done is modified the GroupDescription from Open Ports 22 and 80 (which was incorrect since we closed port 22) to Open Port 80 for website. So that is the only change that we are going to make with this next ChangeSet

OLD TEMPLATE

```
MySecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Open Ports 22 and 80
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
```


CidrIp: 0.0.0.0/0

NEW TEMPLATE

MySecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: Open Port 80 for website

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: '80'

ToPort: '80'

CidrIp: 0.0.0.0/0

Let's see what the ChangeSet thinks are the actions that are going to take place.
In the AWS console in the Sydney region click **Services** and click **CloudFormation**

✓ Select our stack which is still called CFNInitStack

Then click the **Stack Actions** buttons

And click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file

Click the **Choose File** button

★★★ Choose the **ChangeSet-SGDescription.yaml** file because we are creating a ChangeSet that is going to compare our current template to the template that we plan to update to which is **ChangeSet-SGDescription.yaml**

Click **Next**

KeyName: Test Key Pair ← leave as such

NameOfService: Testing CFNInit ← leave as such

Click **Next**

This will direct you to the Configure stack options page

Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.

Just scroll down and click **Create change set**

Change set name: UpdatingSGDescription ← give the change set a name like so
Then click **Create change set**

Give it a minute for the ChangeSet to be created. Hit the refresh icon located in the console if necessary.

Once The Status says CREATE_COMPLETE

Check out the Changes section.

MySecurityGroup is being modified and it says **Replacement True**, which means it is going to replace the security group. That's interesting. It means a new security group will need to be created.

The EC2Instance is also going to be modified and it says **Replacement Conditional**, which means CloudFormation cannot figure out if it is going to be replaced with a new instance or not. It depends on the resource itself and its configuration.

(Here is a Screenshot for clarification)

The screenshot shows the AWS CloudFormation console's 'Changes' section for a change set. The top navigation bar includes 'Changes', 'Input', 'Template', and 'JSON changes'. The 'Overview' section displays the change set ID, status (CREATE_COMPLETE), description, and creation time. Below this, the 'Changes (2)' section shows a table of resources to be modified.

Action	Logical ID	Physical ID	Resource type	Replacement
Modify	EC2Instance	i-0ec43d0899676fe0b	AWS::EC2::Instance	Conditional
Modify	MySecurityGroup	CFNInit-Stack-MySecurityGroup-1DM52W8EGC4BA	AWS::EC2::SecurityGroup	True

Click the **Execute** button

Once the update is complete

Check out the Events section and you will see that it created a new security group. It also created a new instance.

Let's check on that EC2 instance in the EC2 console. Click **Services**, then click **EC2**, then click **Running instances**

✓ Select the new instance that has been created, which is still called TestingCFNInit

Click on the **Description** tab and within the Description, click on the Security Group, and in the security group description it will say Open Port 80 for website. So the new GroupDescription was propagated per the change we made to the template.

The only issue we have here is that we literally have only updated the description of the Security Group in yet our web server went down and then was replaced. So there was downtime. How could we avoid such a scenario?

Let's check the **AWS Resource Types Reference documentation for CloudFormation** to find the answer. Here again is the link:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

Within the documentation, let's find the resource type that we were working with which is **AWS::EC2::SecurityGroup** and click the **Properties** section and find the GroupDescription

Properties

GroupDescription

A description for the security group. This is informational only.

Constraints: Up to 255 characters in length

Constraints for EC2-Classic: ASCII characters

Constraints for EC2-VPC: a-z, A-Z, 0-9, spaces, and
._-:/()#,@[]+=&;{}!\$*

Required: Yes

Type: String

Update requires: Replacement

Per the screenshot above which is directly from the AWS documentation, we can see that updating the GroupDescription property requires the replacement of the resource (which is the security group). So changing the GroupDescription will mean that the security group will be replaced. So changing the GroupDescription can be a drastic action. But why does the EC2 instance get replaced as well? In the documentation, click on **AWS::EC2::Instance** and then go to the properties section. And let's see what the behavior is for SecurityGroupIds

SecurityGroupIds

A list that contains the security group IDs for VPC security groups to assign to the Amazon EC2 instance. If you specified the `NetworkInterfaces` property, do not specify this property.

Required: Conditional. Required for VPC security groups.

Type: List of String values

Update requires:

- *Update requires:* **No interruption** for instances that are in a VPC.
- *Update requires:* **Replacement** for instances that are not in a VPC.

Per the screenshot of the AWS Documentation it says that an update requires replacement for instances that are not in a VPC.

In our template where we defined our EC2 instance, we did not define a VPC ID or a subnet ID. So let's add a VPC ID to our security group and a subnet to our EC2 instance to see if we can flip the behavior from replacing the instances to no interruption whatsoever.

So let's update our template. Copy and paste the following YAML code into a code editor file and save it as `ChangeSet-AddVPC.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

VpcId:

Description: Enter the VpcId

Type: AWS::EC2::VPC::Id

SubnetId:

Description: Enter the SubnetId

Type: AWS::EC2::Subnet::Id

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

SubnetId: !Ref SubnetId

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

```
VpcId: !Ref VpcId
SecurityGroupIngress:
- 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}'
```

So in the above YAML we have added the following parameters:

```
VpcId:
  Description: Enter the VpcId
  Type: AWS::EC2::VPC::Id
SubnetId:
  Description: Enter the SubnetId
  Type: AWS::EC2::Subnet::Id
```

The two additional parameters that we have added are the VPC ID with an AWS data type of `AWS::EC2::VPC::Id` as well as the SubnetID which is also an AWS data type

And on Line 54 of our YAML code we have also added the SubnetId like so:

```
SubnetId: !Ref SubnetId
```

This means that our EC2 instance will be provisioned within a SubnetID of a VPC

Additionally for our security group we specified a Vpc ID as per below:

```
MySecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Open Ports 22 and 80
    VpcId: !Ref VpcId    ←we added a VpcId like so
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: '80'
      ToPort: '80'
      CidrIp: 0.0.0.0/0
```

Now referring to the documentation (see screenshot below), if we change the SubnetId for an EC2 instance, then it will be replaced. So we expect this EC2 instance to be completely replaced with a new one.

SubnetId

If you're using Amazon VPC, this property specifies the ID of the subnet that you want to launch the instance into. If you specified the `NetworkInterfaces` property, do not specify this property.

Required: No

Type: String

Update requires: [Replacement](#)

Now let's check the documentation to see what happens when you add a VPC ID.

VpcId

The physical ID of the VPC. You can obtain the physical ID by using a reference to an `AWS::EC2::VPC`, such as: `{ "Ref" : "myVPC" }`.

For more information about using the Ref function, see [Ref](#).

Required: Yes, for VPC security groups without a default VPC

Type: String

Update requires: [Replacement](#)

Note

For more information about VPC security groups, see [Security Groups](#) in the *Amazon VPC User Guide*.

A VpcId will also replace a security group. So when we initially add the security group and the EC2 instance to a VPC we expect them to be replaced. But once we change the security group's GroupDescription for the second time, we would expect it to be updated in place with NO interruptions.

So let's see what happens. We will create a ChangeSet for adding the resources into a VPC.
Go to the AWS Console in the Sydney region

✓ Select our stack which is still called CFNInitStack

Then click the **Stack Actions** buttons

And click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file

Click the **Choose File** button

★★★ Choose the **ChangeSet-AddVPC.yaml** file because we are creating a ChangeSet that is going to compare our current template to the template that we plan to update to which is **ChangeSet-AddVPC.yaml**

Click **Next**

KeyName: Test Key Pair ← leave as such

NameOfService: Testing CFNInit ← leave as such

SubnetId: subnet-b9d44ae1 (172.31.16.0/20) ← select this subnet. Yours will be slightly different.

VpcId: vpc-5eebef39 (172.31.0.0/16) ← select this VPC. Yours will be slightly different.

Click **Next**

This will direct you to the Configure stack options page

Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.

Just scroll down and click **Create change set**

Change set name: AddToVPC ← give it a name like so

Then click the **Create change set** button

Once the Status says CREATE_COMPLETE,

In the Changes section (see screenshot below), you will see that MySecurityGroup was replaced and the EC2Instance was replaced as well since they need to be created in the VPC and Subnet (which was explicitly stated in the documentation)

Changes (2)				
<input type="text" value="Search changes"/>				
< 1 > ⚙				
Action	Logical ID	Physical ID	Resource type	Replacement
Modify	EC2Instance	i-03d359e4b5ed4408e ↗	AWS::EC2::Instance	True
Modify	MySecurityGroup	CFNInit-Stack-MySecurityGroup-1QWU0CS30CFS1 ↗	AWS::EC2::SecurityGroup	True

Now go ahead and click the **Execute** button

Wait a minute or two for the updates to complete and hit the refresh icon in the console if necessary. In the Events section we see that the security group was replaced and the EC2 instance was replaced. Let's confirm that in the EC2 management console.

Click Services, then click EC2, and then click **Running instances**

✓ Select the new instance that has been created, which is still called TestingCFNInit

Click on the **Description** tab and within the Description click on the Security Group **CFNInitStack-MySecurityGroup** and the description for the security group has rolled back to say Open Ports 22 and 80. However, this was done on purpose. When we change our security group description next time within a VPC Id, we should see different behavior. Once these resources are in the VPC and subnet we can then see if we can simply update the security group description, which will replace the security group, but will NOT replace the EC2 instance. Because in the past it disrupted our web server when we tried to change the security group description.

Copy and paste the below YAML code into a text editor and save the file as **ChangeSet-SGDescriptionPart2**

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

VpcId:

Description: Enter the VpcId

Type: AWS::EC2::VPC::Id

SubnetId:

Description: Enter the SubnetId

Type: AWS::EC2::Subnet::Id

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

SubnetId: !Ref SubnetId

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 Port 80 for website

VpcId: !Ref VpcId

SecurityGroupIngress:

- 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}'

Go to the AWS Management Console in the Sydney region.

Click **Services** and click **CloudFormation**

✓ Select our stack which is still called CFNInitStack

Then click the **Stack Actions** buttons

And click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file

Click the **Choose File** button

★★★ Choose the **ChangeSet-SGDescriptionPart2**

file because we are creating a ChangeSet that is going to compare our current template to the template that we plan to update to which is **ChangeSet-SGDescriptionPart2**

Click **Next**

KeyName: Test Key Pair ← leave as such

NameOfService: Testing CFNInit ← leave as such

SubnetId: subnet-b9d44ae1 (172.31.16.0/20) ← leave as such

VpcId: vpc-5eebef39 (172.31.0.0/16) ← leave as such

(So our parameters are all already in place)

Click **Next**

This will direct you to the Configure stack options page

Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.
Just scroll down and click **Create change set**

Change set name: ChangeSGDescInVPC ← give it a name like so
Then click the **Create change set** button

Once the ChangeSet is complete you will see in the Changes section that
MySecurityGroup says Replacement True
and EC2Instance says Replacement Conditional
(Here is a screenshot for clarification)

Changes (2)				
<input type="text" value="Search changes"/>				
< 1 > ⚙				
Action	Logical ID	Physical ID	Resource type	Replacement
Modify	EC2Instance	i-077de18f91eacd483	AWS::EC2::Instance	Conditional
Modify	MySecurityGroup	sg-0b572c4704a0a6bed	AWS::EC2::SecurityGroup	True

So the security group will be modified and replaced but now the EC2 instance is conditional.

Click the JSON changes tab for further clarification

```
[
  {
    "resourceChange": {
      "logicalResourceId": "EC2Instance",
      "action": "Modify",
      "physicalResourceId": "i-077de18f91eacd483",
      "resourceType": "AWS::EC2::Instance",
      "replacement": "Conditional",
      "details": [
        {
          "target": {
            "name": "SecurityGroupIds",
            "requiresRecreation": "Conditionally",
            "attribute": "Properties"
          },
          "causingEntity": "MySecurityGroup",
          "evaluation": "Static",
          "changeSource": "ResourceReference"
        }
      ],
      "scope": [
        "Properties"
      ]
    },
    "type": "Resource"
  }
]
```

```

    },
    {
      "resourceChange": {
        "logicalResourceId": "MySecurityGroup",
        "action": "Modify",
        "physicalResourceId": "sg-0b572c4704a0a6bed",
        "resourceType": "AWS::EC2::SecurityGroup",
        "replacement": "True",
        "details": [
          {
            "target": {
              "name": "GroupDescription",
              "requiresRecreation": "Always",
              "attribute": "Properties"
            },
            "causingEntity": null,
            "evaluation": "Static",
            "changeSource": "DirectModification"
          }
        ],
        "scope": [
          "Properties"
        ]
      },
      "type": "Resource"
    }
  ]

```

In the above JSON it says

```

"logicalResourceId": "EC2Instance",
  "action": "Modify",          ←so the EC2 instance is to be modified

```

```

"target": {
  "name": "SecurityGroupIds",    ← the SecurityGroupIds is causing the trigger
  "requiresRecreation": "Conditionally",

```

So based on the documentation that we have read we know that if the EC2 instance is in a VPC this instance should NOT be replaced and will instead simply be modified in place. Let's validate this by clicking the **Execute** button and seeing for ourselves.

Again to summarize what went on, when we initially updated the security group description we did not specify the VPC ID, which meant that the EC2 instance was replaced. However, in this case we have specified the VPC ID and we'd expect the security group to be replaced but NOT the EC2 instance. Which means that our web server should see no interruptions.

Take a look at the events

The security group was replaced and the EC2 instance was NOT replaced.
The EC2 instance was simply updated.

So go to the EC2 console to confirm that another EC2 instance was not created and that it is the same EC2 instance running as before. Click **Services** and click **EC2** and click **Running Instances**.

✓ Select our running instance and click the Description tab.

Click on the security group within the description and you will see that the security group has a new description which is Open Port 80 for website

So we have replaced our security group but this time our web server saw no interruptions. And that is how we can more confidently predict the behavior of our stack changes by using documentation and change sets.

CloudFormation: Updating our Stack with ChangeSets Part 2

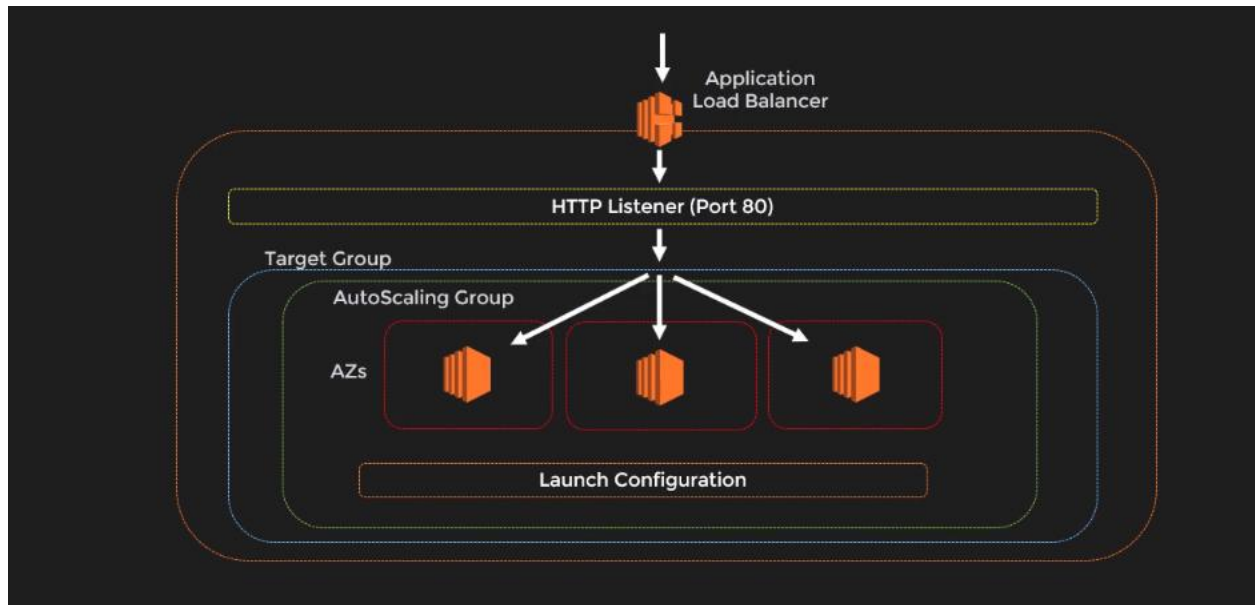
Per the previous lab/tutorial, Our current stack has a single EC2 instance with a security group.

Let's make this a lot more scalable. So our desired end state is as follows:

We will introduce an application load balancer. We will create an HTTP Listener on Port 80 with the load balancer. We will then create a Target Group and an AutoScaling Group. We will use three availability zones. We will use a launch configuration to create and start up instances in each of the Availability Zones.

So when a request comes in for our web server, it is going to hit our load balancer, which is going to be passed on to the HTTP listener and then passed on to the target group. The target group will then be able to send the request on to any of the EC2 instances in the Availability Zone. This is a much more scalable, cloud-native approach to creating infrastructure. Let's see what our ChangeSet might look like if we try and migrate towards this sort of template.

The diagram is what our desired end state looks like:



We will drastically change our current architecture which only has one ec2 instance and a security group and we are going to introduce all of the resources that are in the diagram above.

Below is the new template that we will use to accomplish our desired end state. Copy and paste the YAML code into a new code editor file and save the file as FullStack-ChangeSet.yaml

Parameters:

myKeyPair:

Description: Amazon EC2 Key Pair

Type: AWS::EC2::KeyPair::KeyName

VpcId:

Description: Enter the VpcId

Type: AWS::EC2::VPC::Id

SubnetIds:

Description: Enter the Subnets

Type: List<AWS::EC2::Subnet::Id>

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:

LoadBalancer: # Application Load Balancer
Type: AWS::ElasticLoadBalancingV2::LoadBalancer
Properties:
 SecurityGroups:
 - !Ref ALBSecurityGroup
 Subnets: !Ref SubnetIds
LoadBalancerListener: # Port 80 Listener for ALB
Type: AWS::ElasticLoadBalancingV2::Listener
Properties:
 LoadBalancerArn: !Ref LoadBalancer
 Port: 80
 Protocol: HTTP
 DefaultActions:
 - Type: forward
 TargetGroupArn:
 Ref: TargetGroup
TargetGroup:
Type: AWS::ElasticLoadBalancingV2::TargetGroup
Properties:
 Port: 80
 Protocol: HTTP
 VpcId: !Ref VpcId
AutoScalingGroup:
Type: AWS::AutoScaling::AutoScalingGroup
Properties:
 AvailabilityZones: !GetAZs
 LaunchConfigurationName: !Ref LaunchConfiguration
 MinSize: 1
 MaxSize: 3
 TargetGroupARNs:
 - !Ref TargetGroup
LaunchConfiguration:
Type: AWS::AutoScaling::LaunchConfiguration
Metadata:
 Comment: Install php and httpd
 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:

KeyName: !Ref myKeyPair

InstanceType: t2.micro

SecurityGroups:

- !Ref EC2SecurityGroup

ImageId:

Fn::FindInMap:

- RegionMap
- !Ref AWS::Region
- AMI

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 LaunchConfiguration --region \${AWS::Region}

ALBSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: ALB Security Group

VpcId: !Ref VpcId

SecurityGroupIngress:

- IpProtocol: tcp
- FromPort: 80
- ToPort: 80
- CidrIp: 0.0.0.0/0

EC2SecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: EC2 Instance

EC2InboundRule: # EC2 can only accept traffic from ALB

Type: AWS::EC2::SecurityGroupIngress

Properties:

IpProtocol: tcp

FromPort: 80

ToPort: 80

SourceSecurityGroupId:

!GetAtt

- ALBSecurityGroup
- GroupId

GroupId:

!GetAtt

- EC2SecurityGroup
- GroupId

Outputs:

PublicDns:

Description: The Public DNS

Value: !Sub 'http://\${LoadBalancer.DNSName}'

Let's analyze each item in the above YAML code to fully understand it.

Starting with the parameters section...we have three parameters in total. Our three parameters include a key pair to log in to the EC2 instance, a VPC id, as well as a Subnet id.

Parameters:

myKeyPair:

Description: Amazon EC2 Key Pair

Type: AWS::EC2::KeyPair::KeyName

VpcId:

Description: Enter the VpcId

Type: AWS::EC2::VPC::Id

SubnetIds:

Description: Enter the Subnets

Type: List<AWS::EC2::Subnet::Id>

Then we have our mappings which define our AMI IDs across regions

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

Now let's move on to our resources. First we've got our Application Load Balancer.

This Application Load Balancer references an ALB security group.

LoadBalancer: # Application Load Balancer

Type: AWS::ElasticLoadBalancingV2::LoadBalancer

Properties:

SecurityGroups:

- !Ref ALBSecurityGroup ← here our alb references an alb security group

Subnets: !Ref SubnetIds

Let's check out the ALBSecurity group which is on line 92 of the YAML code. For the ALB security group, which is within a Vpc ID, we are creating a single ingress rule which is going to allow anyone to come in via port 80, which is what we want for our load balancer to do.

ALBSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: ALB Security Group

VpcId: !Ref VpcId ← as you can see the ALBSecurityGroup is within a VpcId

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 80

ToPort: 80

CidrIp: 0.0.0.0/0

On line 30 of the YAML code we have a LoadBalancerListener which is essentially going to say listening on port 80. We want to forward all the traffic to a TargetGroup. The LoadBalancerListener is referencing the LoadBalancer.

LoadBalancerListener: # Port 80 Listener for ALB

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

LoadBalancerArn: !Ref LoadBalancer ← our listener is referencing this loadbalancer

Port: 80 ← our LoadBalancerListener is essentially going to say it's listening on port 80

Protocol: HTTP

DefaultActions:

- Type: forward ← we want to forward all the traffic to a TargetGroup

TargetGroupArn:

Ref: TargetGroup

Now let's have a look at the TargetGroup which is on line 40 of the YAML code. The TargetGroup on port 80 for HTTP is within this VpcID. And this target group can be associated to an AutoScalingGroup.

TargetGroup:

Type: AWS::ElasticLoadBalancingV2::TargetGroup

Properties:

Port: 80

Protocol: HTTP

VpcId: !Ref VpcId

Now let's have a look at the AutoScalingGroup which is on line 46 of the YAML code. The AutoScalingGroup is basically saying that within the availability zones and within the launch configuration, these are the number of instances that we want to have. A minimum of 1 and a maximum of 3. The AutoScalingGroup is associated with the TargetGroup. So the AutoScalingGroup will control the number of healthy EC2 instances within the TargetGroup.

AutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

AvailabilityZones: !GetAZs ← within these availability zones

LaunchConfigurationName: !Ref LaunchConfiguration ←and within this launch configuration
MinSize: 1 ←these are the number of instances that we want to have. A min of one
MaxSize: 3 ←and a maximum of 3
TargetGroupARNs:
- !Ref TargetGroup ←The AutoScalingGroup is associated with this TargetGroup

Our LaunchConfiguration (starting on line 55) is a configuration which defines all the parameters to start up an EC2 instance. And so this is basically setting up each of our EC2 instances. It's going to install our PHP and web server through the metadata. We defined our key pair on line 75, and defined our instance type on line 76, and then defined the security group for our ec2 instance, and then defined the AMI, and then defined the user data. Normally you would see all of these properties defined on an EC2 instance, but in this case we are going to use a launch configuration. This allows the AutoScalingGroup to scale up and scale down as needed.

LaunchConfiguration:
Type: AWS::AutoScaling::LaunchConfiguration

Metadata: ←It's going to install our PHP and web server through the metadata

Comment: Install php and httpd
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:

KeyName: !Ref myKeyPair
InstanceType: t2.micro
SecurityGroups:
- !Ref EC2SecurityGroup
ImageId:
 Fn::FindInMap:
 - RegionMap
 - !Ref AWS::Region
 - AMI
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 LaunchConfiguration --region ${AWS::Region}
```

Now looking at our EC2SecurityGroup starting on line 102 of the YAML code...we have defined an EC2SecurityGroup and one of the ingress rules that we've defined for it is basically to only allow traffic from port 80 from the ALBSecurityGroup. This means that nobody can hit this EC2 directly. Only traffic which is sent from the ALB can hit this EC2SecurityGroup. So it is a lot more locked down, unlike before which was an EC2 instance which was open to the world.

```
EC2SecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: EC2 Instance
EC2InboundRule: # EC2 can only accept traffic from ALB
  Type: AWS::EC2::SecurityGroupIngress
  Properties:
    IpProtocol: tcp
    FromPort: 80
    ToPort: 80
    SourceSecurityGroupId:
      !GetAtt
      - ALBSecurityGroup
      - GroupId
    GroupId:
      !GetAtt
      - EC2SecurityGroup
      - GroupId
```

And so that was all the resource that we are going to define in this new stack. And the Output will be the public DNS name of the LoadBalancer

```
Outputs:
  PublicDns:
    Description: The Public DNS
    Value: !Sub 'http://${LoadBalancer.DNSName}'
```

So as you can see, we no longer have an EC2 instance. We've got an AutoScalingGroup and a LaunchConfiguration. We've got security groups for the EC2 instance and also for the ALB and the load balancers. So our stack has changed drastically since the last lab/tutorial. It is now a lot more secure and a lot more scalable. We will thus have a lot of changes to resources. Let's see what the ChangeSet will look like when we upload this new template.

Go to the AWS Management Console in the Sydney region.
Click **Services** and click **CloudFormation**

✓ Select our stack which is still called CFNInitStack

Then click the **Stack Actions** buttons
And click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file
Click the **Choose File** button

★★★ Choose the **FullStack-ChangeSet.yaml** file because we are creating a ChangeSet that is going to compare our current template to the template that we plan to update to which is **FullStack-ChangeSet.yaml**

Click **Next**

KeyName: Test Key Pair ← choose the key pair that we created in a previous lab/tutorial

SubnetIds: ← ★ select all three subnet ids (172.31.0.0/20 , 172.31.16.0/20 , 172.31.32.0/20)

This is for the AutoScalingGroup so it knows which subnets it can allocate resources to

VpcId: vpc-5eebef39 (172.31.0.0/16) ← leave as such

Click **Next**

This will direct you to the Configure stack options page

Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.

Just scroll down and click **Create change set**

Change set name: FullStack ← give it a name like so

Then click the **Create change set** button

Per the Changes section (see screenshot below) we can see that we are adding an ALBSecurityGroup, We are adding an AutoScalingGroup, we are adding the EC2InboundRule, we are adding the EC2SecurityGroup, we are adding the LaunchConfiguration, we are adding the LoadBalancer, we are adding the LoadBalancerListener, and we are adding the TargetGroup.
We are removing the original EC2Instance and removing the original security group.

Changes (10)				
<input type="text" value="Search changes"/>				
<div> < 1 > ⚙ </div>				
Action	Logical ID	Physical ID	Resource type	Replacement
Add	ALBSecurityGroup	-	AWS::EC2::SecurityGroup	-
Add	AutoScalingGroup	-	AWS::AutoScaling::AutoScalingGroup	-
Add	EC2InboundRule	-	AWS::EC2::SecurityGroupIngress	-
Remove	EC2Instance	i-098bbfddb6633ebd6 🔗	AWS::EC2::Instance	-
Add	EC2SecurityGroup	-	AWS::EC2::SecurityGroup	-
Add	LaunchConfiguration	-	AWS::AutoScaling::LaunchConfiguration	-
Add	LoadBalancerListener	-	AWS::ElasticLoadBalancingV2::Listener	-
Add	LoadBalancer	-	AWS::ElasticLoadBalancingV2::LoadBalancer	-
Remove	MySecurityGroup	sg-0c5bba8abe8b8b69c 🔗	AWS::EC2::SecurityGroup	-
Add	TargetGroup	-	AWS::ElasticLoadBalancingV2::TargetGroup	-

It is a complete full stack creation and a removal of all the resources that existed. Essentially, every resource is being cleaned out and created in a new set. Nothing is being replaced it is simply completely new resources and new logical IDs. This is what we'd expect because this is a drastic change. We don't have any EC2 instances anymore. They are being automatically created by the AutoScalingGroup and the LaunchConfiguration. All of our traffic should go via the LoadBalancer and our security groups should lock down our EC2 instances a lot more strictly. So only allowing traffic from the ALB aka application load balancer.

Now click the **Execute** button.

The Status will say UPDATE_IN_PROGRESS

The security group is being created for the ALB and the EC2.

Our TargetGroup is being created.

Then the AutoScalingGroup will then be created and the LaunchConfiguration will also be created which will follow on with an EC2 instance which does not need to be managed by CloudFormation as it is NOT an explicit resource in the template.

Click the refresh icon in Events section and it should say UPDATE_COMPLETE

So now it finished creating the ALBSecurityGroup and our LaunchConfiguration. It created the AutoScalingGroup which will kick off the creation of our EC2 instances. It also created the LoadBalancer and set up the listener as well.

Now click the **Outputs** tab

And you will see the PublicDns Url link which will look something like the following:

<http://CFNIn-LoadB-1WSZFZAJMS783-936066101.ap-southeast-2.elb.amazonaws.com>

Click on the link and it will say May the Force be with you!

So it hit our load balancer which is then hitting our PHP script on the instance.

Referring back to our template called **FullStack-ChangeSet.yaml**

Let's change the MinSize to 3 (instead of 1) and then save the file!!! No need to save it as a different name if you do not want to. Just save it as **FullStack-ChangeSet.yaml**

AutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

AvailabilityZones: !GetAZs

LaunchConfigurationName: !Ref LaunchConfiguration

MinSize: 3

MaxSize: 3

TargetGroupARNs:

- !Ref TargetGroup

Go to the AWS Management Console in the Sydney region.

Click **Services** and click **CloudFormation**

✓ Select our stack which is still called CFNInitStack

Then click the **Stack Actions** buttons

And click **Create change set for current stack**

Prepare Template: ✓ Replace current template ← select replace current template

Template Source: ✓ Upload a template file ← select Upload a template file

Click the **Choose File** button

★★★ Choose the **FullStack-ChangeSet.yaml** file (which we just made a change to and saved. So the MinSize of our AutoScalingGroup is now 3)

Click **Next**

KeyName: Test Key Pair ← leave as is

SubnetIds: (172.31.0.0/20 , 172.31.16.0/20 , 172.31.32.0/20) ← make sure all three are selected

VpcId: vpc-5eebef39 (172.31.0.0/16) ← leave as is

Click **Next**

This will direct you to the Configure stack options page

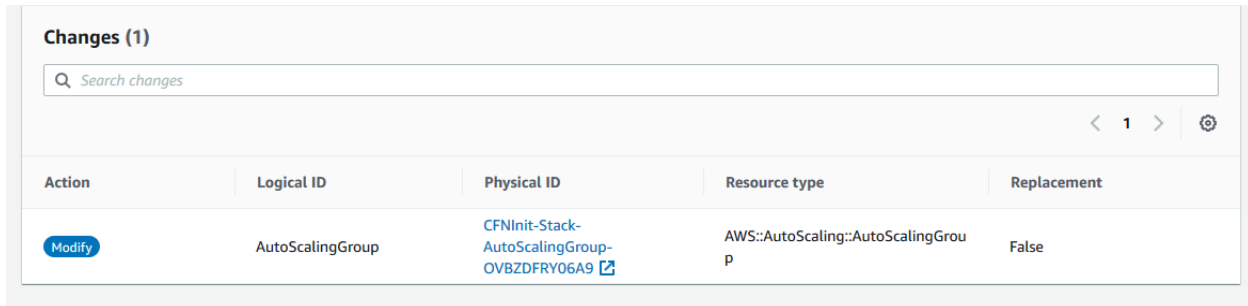
Leave everything as is.

Just scroll down and click **Next**

This will direct you to the Review page. Leave everything as is.
Just scroll down and click **Create change set**

Change set name: ASGof3 ← give it a name like so
Then click the **Create change set** button

All we are doing is asking the AutoScalingGroup to increase the number of minimum instances. So it is going to modify the AutoScalingGroup and not replace it. (See the screenshot for clarification)



Changes (1)				
Q Search changes				
Action	Logical ID	Physical ID	Resource type	Replacement
Modify	AutoScalingGroup	CFNInit-Stack-AutoScalingGroup-OVBZDFRY06A9	AWS::AutoScaling::AutoScalingGroup	False

Click the **Execute** button

Now let's observe what happens in the EC2 console. So click **Services** and then click **EC2** and then click **Running Instances**

Here we can see that two more instances are starting up. In total we will then have three instances per what we requested in the template. CloudFormation knows nothing of the EC2 instances it only knows of the AutoScalingGroup.

Click **Auto Scaling Groups** located in the left margin. There we can see that the minimum has been updated to 3 per the change we made to our template.

CloudFormation Summary

We learned about Infrastructure as Code and introduced CloudFormation as the tool of choice.

Terminology used in CloudFormation:

- 1) Templates – blueprints for your infrastructure
- 2) Stacks – an instance of your template
- 3) Change Sets – enable you to see what will happen when you update your stack

Anatomy of a Template

Features of CloudFormation:

- Intrinsic Functions – which are built-in functions that make it easier for you to manage your stack
- Multiple Resources – CloudFormation helps you manage multiple resources

-Pseudo Parameters – are variables available for you in templates

-Mappings – are lookups that you can build upon

-Input Parameters – allow you to make templates reusable

-Outputs – allow you to make templates reusable

How to set up and EC2 Instance:

Use UserData, CloudFormation Helper Scripts, and CloudFormation Init section (part of the metadata in the template

ChangeSets:

Assist you in managing critical production environments

There are more advanced features to learn, such as:

Nested Stacks

Stack Sets

Creation, Update, & Deletion Policies

And there are more...
