



Masterclass

AWS CloudFormation

Masterclass

1 A technical deep dive that goes beyond the basics

2 Intended to educate you on how to get the best from AWS services

3 Show you how things work and how to get things done

AWS CloudFormation



An easy way to create & manage a collection of AWS resources
Allows orderly and predictable provisioning and updating of resources
Allows you to version control your AWS infrastructure
Deploy and update stacks using console, command line or API
You only pay for the resources you create

Transparent and Open

Don't reinvent the wheel

Declarative & Flexible



AWS CloudFormation

No Extra Charge

Customized via Parameters

Integration Ready

You can use the same template to create identical copies of the same stack of resources. You also do not need to write these templates from scratch because there are available sample templates that you can use or customize.

There are also execution time Parameters that you can pass to your templates to be used during stack creation, you can pass in things like the RDS database size, database or webserver port numbers, etc. this is good when creating different environments with different types of instances, settings, thresholds, etc.

You can also use the programmatic access to CF via the CLI, SDKs or APIs. CF also publishes progress reports via SNS, this allows you to see what is going on and track progress using SNS.

CloudFormation - Components & Technology

Template



JSON formatted file

Parameter definition

Resource creation

Configuration actions

CloudFormation



Framework

Stack creation

Stack updates

Error detection and rollback

Stack



Configured AWS services

Comprehensive service support

Service event aware

Customisable

Agenda



Creating Templates
Using a Template to Create and Manage a Stack
Working with the CloudFormation API
Working with AWS Resources
Bootstrapping Applications and Handling Updates

CREATING TEMPLATES

Manage Relationships

Reusable

Familiar JSON Format

Provide Feedback

CLOUDFORMATION TEMPLATES

Automate Generation

Avoid Collisions

Look Up Resources

Write & Go

Templates are JSON/YAML text files that allow you to manage the relationships between the different resources within your stack. You can also automate the generation of the templates using any tool that can output JSON text files.

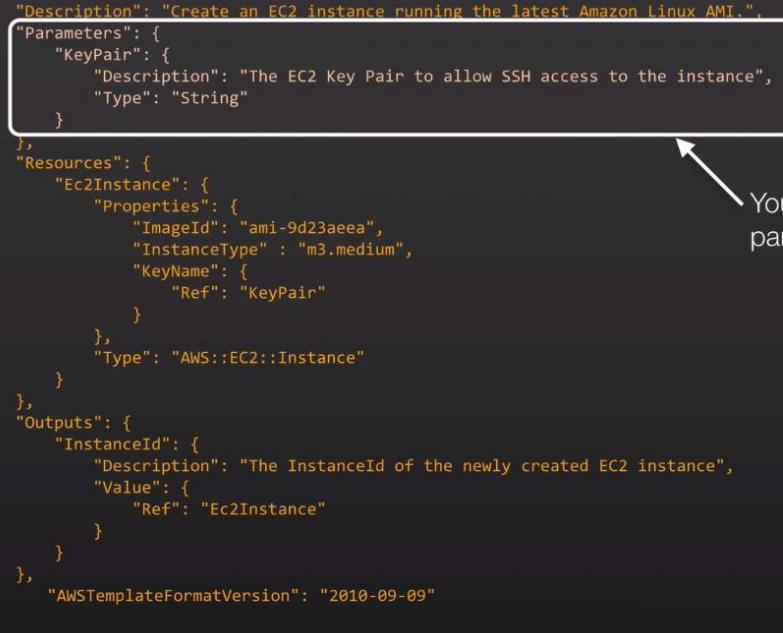
High Level Template Structure

```
{  
    "Description" : "A text description for the template usage",  
    "Parameters": {  
        // A set of inputs used to customize the template per deployment  
    },  
    "Resources" : {  
        // The set of AWS resources and relationships between them  
    },  
    "Outputs" : {  
        // A set of values to be made visible to the stack creator  
    },  
    "AWSTemplateFormatVersion" : "2010-09-09"  
}
```

There are 4 main parts of a CF template shown above

A Simple Template that creates an EC2 Instance

```
{  
    "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",  
    "Parameters": {  
        "KeyPair": {  
            "Description": "The EC2 Key Pair to allow SSH access to the instance",  
            "Type": "String"  
        }  
    },  
    "Resources": {  
        "Ec2Instance": {  
            "Properties": {  
                "ImageId": "ami-9d23aeea",  
                "InstanceType" : "m3.medium",  
                "KeyName": {  
                    "Ref": "KeyPair"  
                }  
            },  
            "Type": "AWS::EC2::Instance"  
        }  
    },  
    "Outputs": {  
        "InstanceId": {  
            "Description": "The InstanceId of the newly created EC2 instance",  
            "Value": {  
                "Ref": "Ec2Instance"  
            }  
        }  
    },  
    "AWSTemplateFormatVersion": "2010-09-09"  
}
```



You will be asked to enter values for these parameters when you create your stack

A Simple Template that creates an EC2 Instance

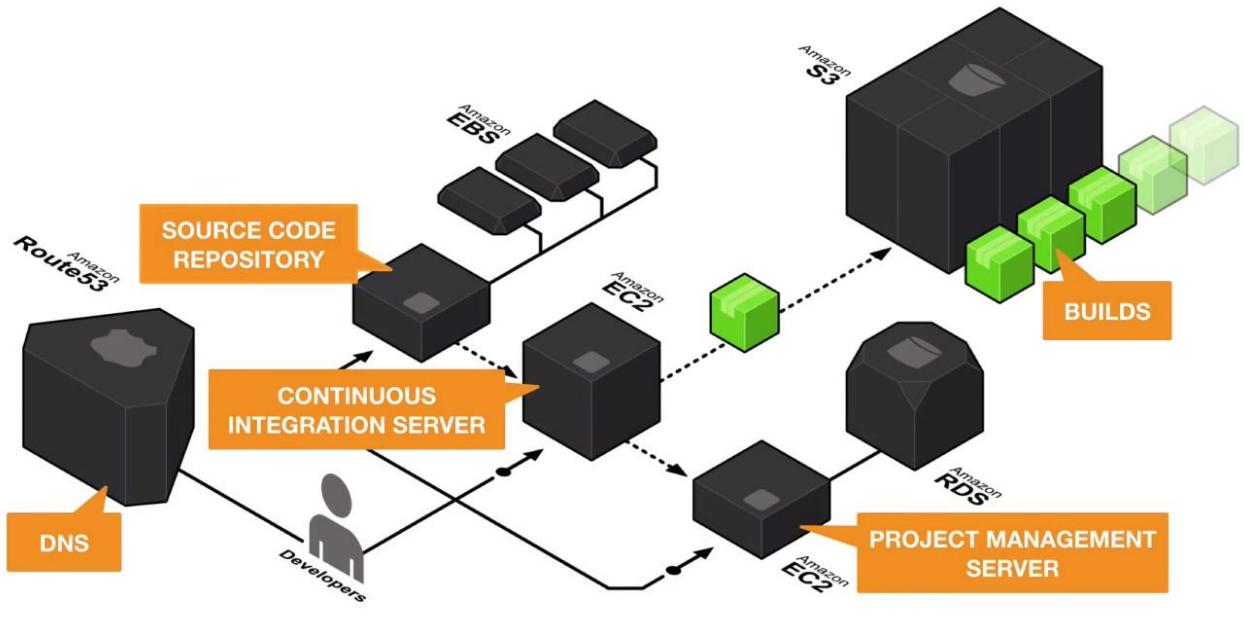
```
{  
    "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",  
    "Parameters": {  
        "KeyPair": {  
            "Description": "The EC2 Key Pair to allow SSH access to the instance",  
            "Type": "String"  
        }  
    },  
    "Resources": {  
        "Ec2Instance": {  
            "Properties": {  
                "ImageId": "ami-9d23aeea",  
                "InstanceType" : "m3.medium",  
                "KeyName": {  
                    "Ref": "KeyPair"  
                }  
            },  
            "Type": "AWS::EC2::Instance"  
        }  
    },  
    "Outputs": {  
        "InstanceId": {  
            "Description": "The InstanceId of the newly created EC2 instance",  
            "Value": {  
                "Ref": "Ec2Instance"  
            }  
        }  
    },  
    "AWSTemplateFormatVersion": "2010-09-09"  
}
```

Includes statically defined properties (ImageID & Instance Type) plus a reference to the KeyPair parameter. ImageID is the AMI specific to the region that you will launch this stack in, in this case the eu-west-1 region

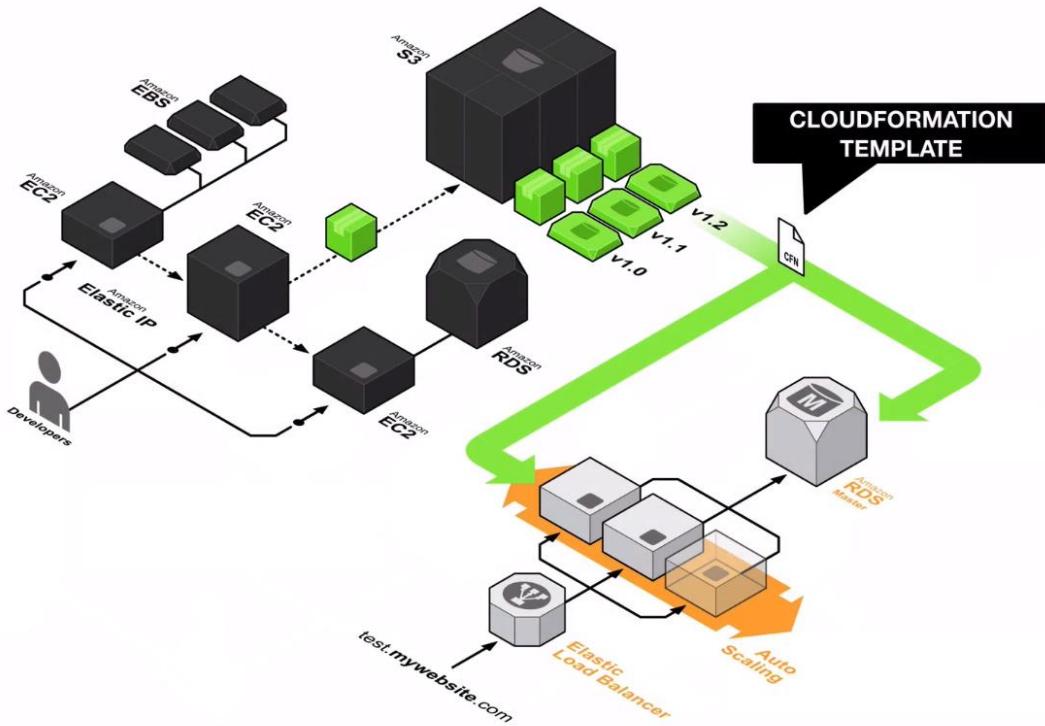
A Simple Template that creates an EC2 Instance

```
{  
    "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",  
    "Parameters": {  
        "KeyPair": {  
            "Description": "The EC2 Key Pair to allow SSH access to the instance",  
            "Type": "String"  
        }  
    },  
    "Resources": {  
        "Ec2Instance": {  
            "Properties": {  
                "ImageId": "ami-9d23aeea",  
                "InstanceType" : "m3.medium",  
                "KeyName": {  
                    "Ref": "KeyPair"  
                }  
            },  
            "Type": "AWS::EC2::Instance"  
        }  
    },  
    "Outputs": {  
        "InstanceId": {  
            "Description": "The InstanceId of the newly created EC2 instance",  
            "Value": {  
                "Ref": "Ec2Instance"  
            }  
        }  
    },  
    "AWSTemplateFormatVersion": "2010-09-09"  
}
```

These outputs will be returned once the template has completed execution



It is quite common to integrate template creation with the CI/CD pipeline as shown above,



CF templates are often created as part of the CI/CD process and checked in and maintained along with the codebase itself. This allows us to create a fresh stack of resources during application bootstrapping using CF templates.

github.com/cloudtools/troposphere

... but remember that a CloudFormation template is just JSON, so any tool that can generate output in JSON can be used

The python tool called troposphere is a python library tool that enables you to create CF resources easy, it then generates the template for your based on the python code.

CREATING & MANAGING STACKS

Using a template to create and manage a stack

```

1  {
2      "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",
3      "Parameters": {
4          "KeyPair": {
5              "Description": "The EC2 Key Pair to allow SSH access to the instance",
6              "Type": "String"
7          }
8      },
9      "Resources": {
10         "Ec2Instance": {
11             "Properties": {
12                 "ImageId": "ami-dd925boa",
13                 "InstanceType" : "m3.medium",
14                 "KeyName": {
15                     "Ref": "KeyPair"
16                 }
17             },
18             "Type": "AWS::EC2::Instance"
19         }
20     },
21     "Outputs": {
22         "InstanceId": {
23             "Description": "The InstanceId of the newly created EC2 instance",
24             "Value": {
25                 "Ref": "Ec2Instance"
26             }
27         }
28     }
29 }
30
31 "AWSTemplateFormatVersion": "2010-09-09"

```

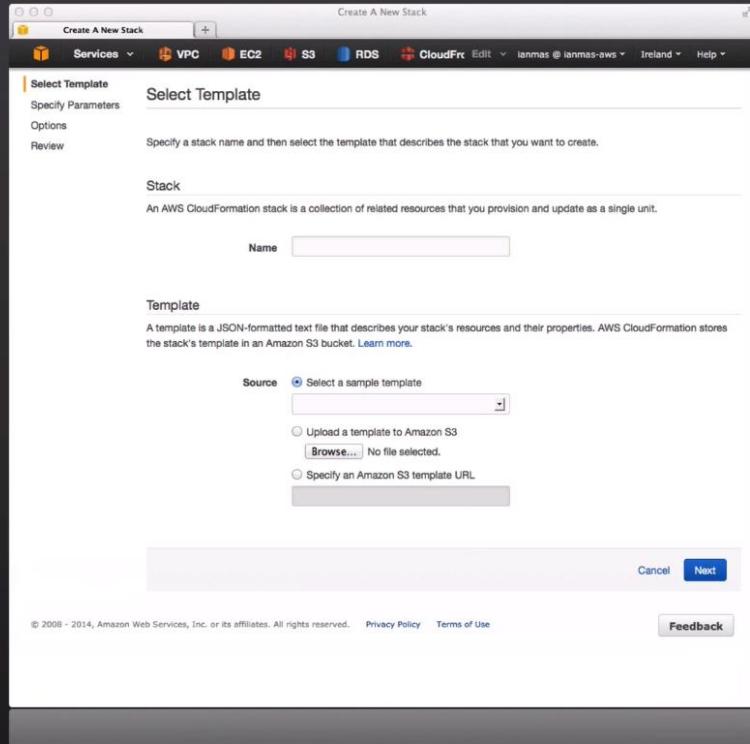
Using a template to create and manage a stack

The screenshot shows the AWS Management Console homepage. The CloudFormation icon is highlighted with a red box. The page lists various AWS services under categories like Compute & Networking, Database, Analytics, App Services, and Deployment & Management.

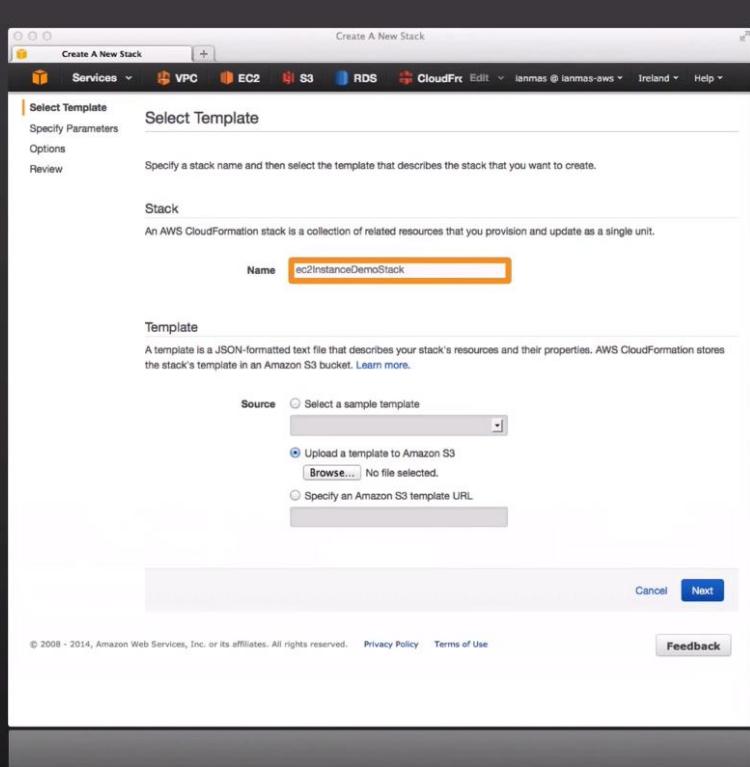
Using a template to create and manage a stack

The screenshot shows the CloudFormation Management Console. It features two main sections: 'Create a Stack' and 'Create a Template from your Existing Resources'. The 'Create a Stack' section contains a brief description of CloudFormation and a 'Create New Stack' button, which is highlighted with a red box. The 'Create a Template from your Existing Resources' section contains instructions for using CloudFormer and a 'Launch CloudFormer' button, which is also highlighted with a red box.

Using a template to create and manage a stack

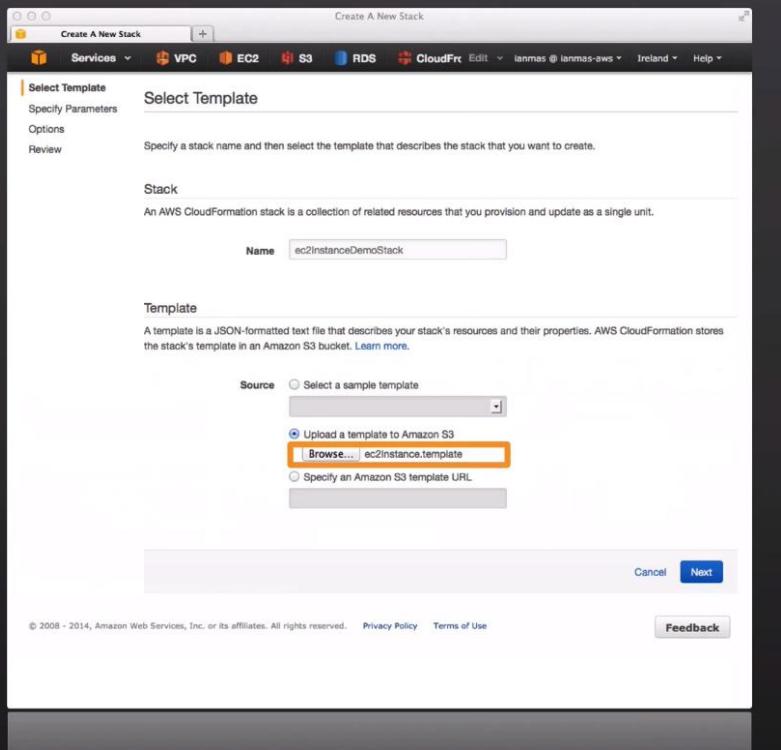


Using a template to create and manage a stack



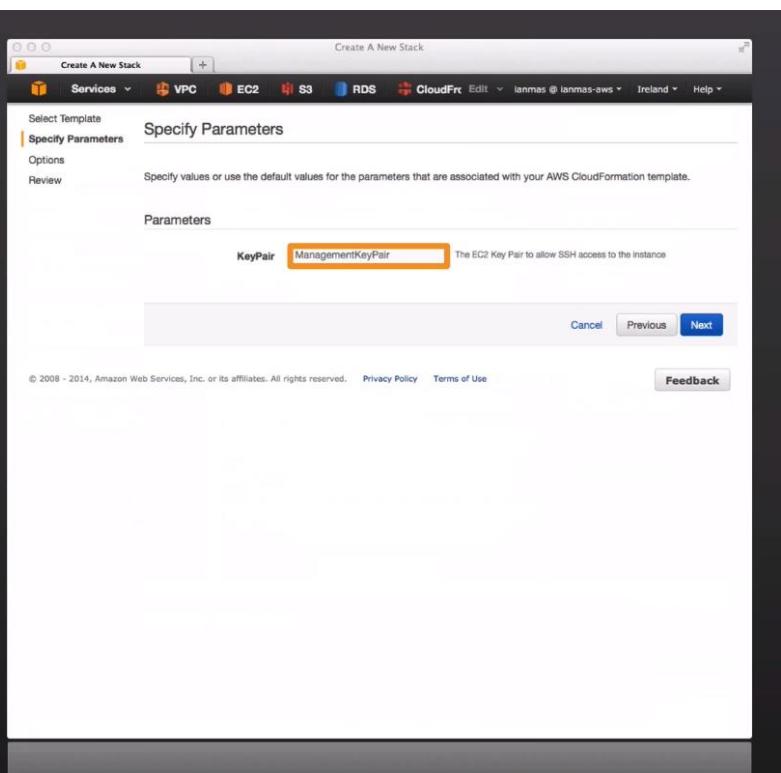
We give the name of the stack we want to create

Using a template to create and manage a stack



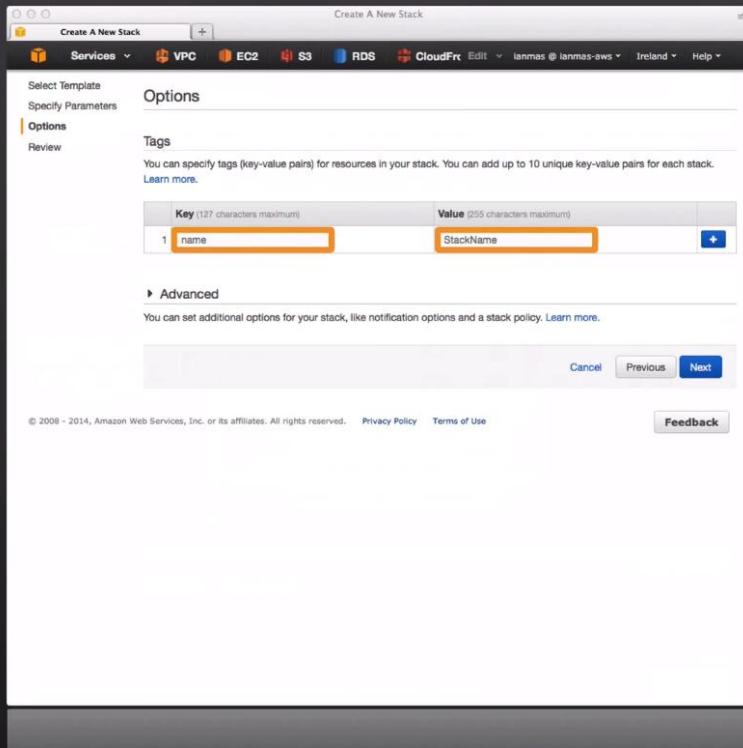
We will be uploading the created template to S3. A best practice will be to use IAM to restrict access to the S3 bucket being used to hold the CF templates.

Using a template to create and manage a stack



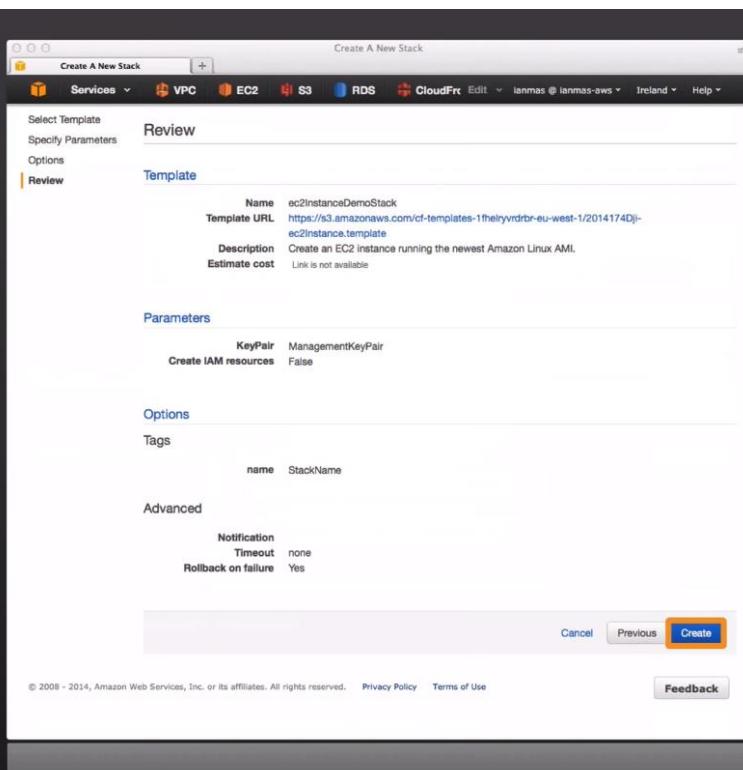
You then will be asked to specify the parameters needed for the CF template

Using a template to create and manage a stack



You then will be asked to name your stack for later use

Using a template to create and manage a stack



Review and click Create button if okay

Using a template to create and manage a stack

CloudFormation Management Console

Showing 1 stack

Stack Name	Created Time	Status	Description
ec2InstanceDemoStack	2014-06-23 12:59:47 UTC+0100	CREATE_COMPLETE	Create an EC2 Instance running the newest Amazon

Events

Date	Status	Type	Logical ID	Status Reason
2014-06-23 12:59:47 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2InstanceDemoStack	User Initiated

Feedback

Feedback

Using a template to create and manage a stack

CloudFormation Management Console

Showing 1 stack

Stack Name	Created Time	Status	Description
ec2InstanceDemoStack	2014-06-23 12:59:47 UTC+0100	CREATE_COMPLETE	Create an EC2 Instance running the newest Amazon

Events

Date	Status	Type	Logical ID	Status Reason
13:00:43 UTC+0100	CREATE_COMPLETE	AWS::CloudFormation::Stack	ec2InstanceDemoStack	
13:00:42 UTC+0100	CREATE_COMPLETE	AWS::EC2::Instance	Ec2Instance	
12:59:54 UTC+0100	CREATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance	Resource creation initiated
12:59:52 UTC+0100	CREATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance	
12:59:47 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2InstanceDemoStack	User Initiated

Feedback

Feedback

You can now begin to see the event stream for the stack creation process

Using a template to create and manage a stack

Status	Type	Logical ID	Status Reason
CREATE_COMPLETE	AWS::CloudFormation::Stack	ec2InstanceDemoStack	
CREATE_COMPLETE	AWS::EC2::Instance	Ec2Instance	
CREATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance	Resource creation initiated
CREATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance	User Initiated
CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2InstanceDemoStack	

Using a template to create and manage a stack

Key	Value	Description
Instanceld	i-d2a60792	The InstanceId of the newly created EC2 instance

You can now see the outputs that we want from the new stack

Using a template to create and manage a stack

```
ec2Instance.template

1 {
2     "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",
3     "Parameters": {
4         "KeyPair": {
5             "Description": "The EC2 Key Pair to allow SSH access to the instance",
6             "Type": "String"
7         }
8     },
9     "Resources": {
10        "Ec2Instance": {
11            "Properties": {
12                "ImageId": "ami-dd925baa",
13                "InstanceType": "m3.medium",
14                "KeyName": {
15                    "Ref": "KeyPair"
16                },
17                "SecurityGroups": "ssh-tools"
18            },
19            "Type": "AWS::EC2::Instance"
20        }
21    },
22    "Outputs": {
23        "InstanceId": {
24            "Description": "The InstanceId of the newly created EC2 instance",
25            "Value": {
26                "Ref": "Ec2Instance"
27            }
28        }
29    },
30    "AWSTemplateFormatVersion": "2010-09-09"
31 }
32 }

33 }
34 ``VM216ubfgeL0w0fA6L2OU``: ``5816-00-00``
35 }
36 }
```

Next, let us see what we need to do when we update our stack definition as above where we have edited the Resources section of the stack template to add a security group access. We have deliberately put in a syntax error in the change above

Using a template to create and manage a stack

CloudFormation Management Console

Actions

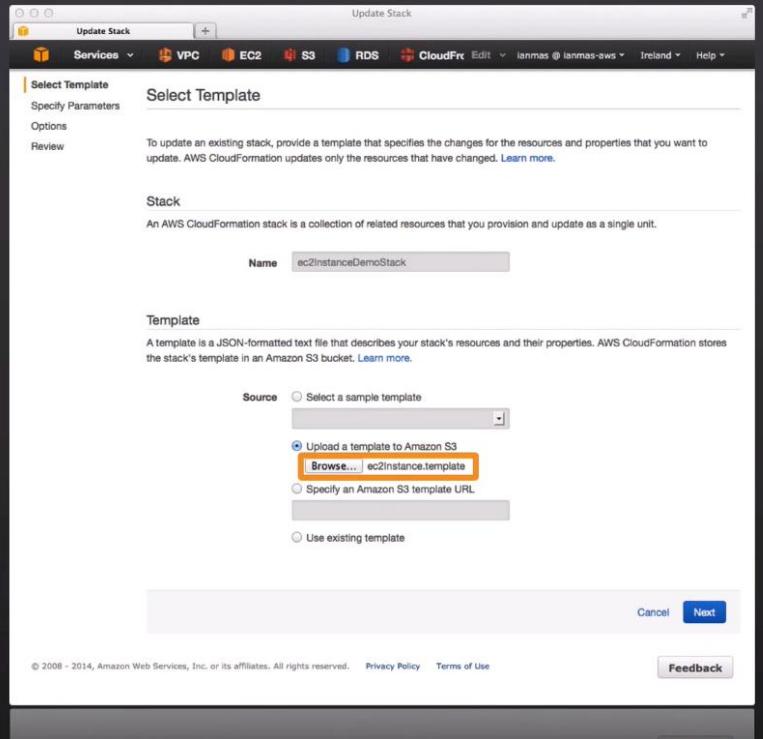
Name:	Created Time	Status	Description
ec2instancecolumstack	2014-06-23 12:58:47 UTC+0100	CREATE_COMPLETE	Create an EC2 instance running the newest Amazon

Outputs

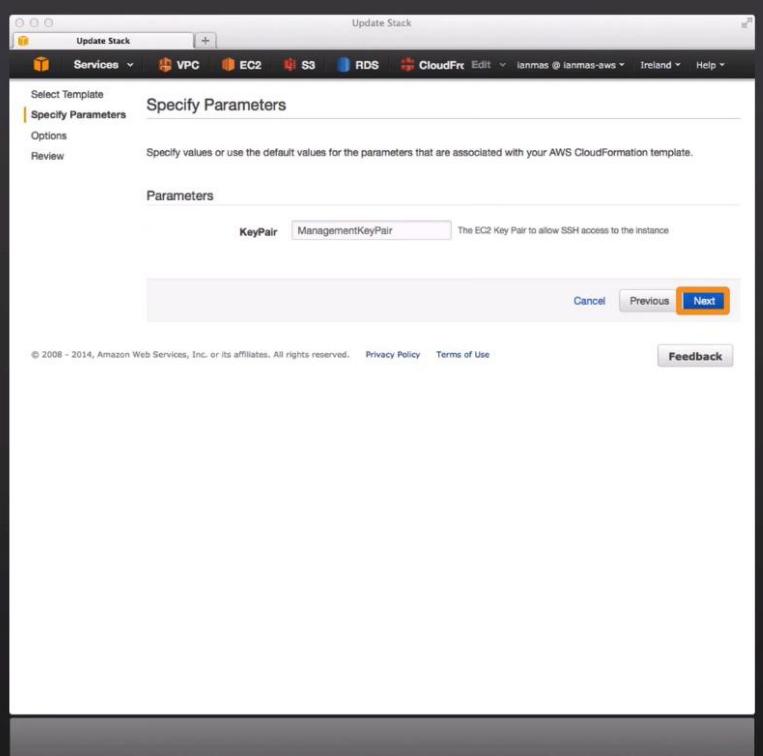
Key	Value	Description
InstanceId	i-d2a60792	The InstanceId of the newly created EC2 instance

Feedback

Using a template to create and manage a stack



Using a template to create and manage a stack



Using a template to create and manage a stack

The screenshot shows the CloudFormation Management Console. A blue diagonal banner in the top right corner says "Provides Feedback". In the center, a table lists a single stack named "ec2InstanceDemoStack" with a status of "UPDATE_ROLLBACK_COMPLETE". Below the table, a section titled "Events" shows four log entries. The fourth entry, at 13:37:26 UTC+0100, has a red error message: "UPDATE_FAILED" for resource "Ec2Instance". A callout box highlights this message with the text: "Value of property SecurityGroups must be of type List of String". At the bottom right of the screen, there is a "Feedback" button.

Note that we see details of what happened since our update as failed.

The screenshot shows a code editor with the file "ec2Instance.template" open. A blue diagonal banner in the top right corner says "Correct Syntax". The code editor displays the following AWS CloudFormation template:

```
{  
  "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",  
  "Parameters": {  
    "KeyPair": {  
      "Description": "The EC2 Key Pair to allow SSH access to the instance",  
      "Type": "String"  
    }  
  },  
  "Resources": {  
    "Ec2Instance": {  
      "Properties": {  
        "ImageId": "ami-dd925baa",  
        "InstanceType": "m3.medium",  
        "KeyName": {  
          "Ref": "KeyPair"  
        },  
        "SecurityGroups": [ "ssh-tools" ]  
      },  
      "Type": "AWS::EC2::Instance"  
    }  
  },  
  "Outputs": {  
    "InstanceId": {  
      "Description": "The InstanceId of the newly created EC2 instance",  
      "Value": {  
        "Ref": "Ec2Instance"  
      }  
    }  
  },  
  "AWSTemplateFormatVersion": "2010-09-09"  
}  
Line: 19:41 | Plain Text | Tab Size: 4 |   
File: 10:47 | 100% | 100% | 100% | 100% |
```

We can then fix the syntax problem

Using a template to create and manage a stack

The screenshot shows the CloudFormation Management Console interface. At the top, a table lists a single stack named 'ec2InstanceDemoStack' with a status of 'UPDATE_COMPLETE'. Below this, a detailed log of events is displayed, showing the sequence of operations from creation to update completion. The log includes entries for AWS::CloudFormation::Stack and AWS::EC2::Instance resources, with status codes like 'UPDATE_COMPLETE', 'DELETE_COMPLETE', and 'CREATE_IN_PROGRESS'. A large blue watermark reading 'Replaces Resources' is overlaid across the bottom right of the console window.

We can now successfully complete the update and recreate the stack, we now have a new EC2 instance with the appropriate security group

The screenshot shows the EC2 Management Console interface. On the left, a sidebar menu includes 'Instances', which is currently selected. The main pane displays a list of instances, with one instance named 'i-86872bc5' highlighted. Below this, a detailed view of the selected instance is shown, including its Public DNS ('ec2-54-76-124-9.eu-west-1.compute.amazonaws.com'), Instance ID ('i-86872bc5'), and various configuration details like Instance Type ('m3.medium'), State ('running'), and Security Group ('ssh-tools'). A large blue watermark reading 'Replaces Resources' is overlaid across the top right of the console window.

We can see that the old EC2 instance is being terminated and we have a new EC2 instance running from the updateStack operation

Using a template to create and manage a stack

The screenshot shows the CloudFormation Management Console interface. At the top, there's a navigation bar with tabs for Services, VPC, EC2, S3, RDS, CloudFront, and Help. The user is signed in as 'ianmas @ ianmas-aws'. Below the navigation bar, there's a search bar with 'Name:' and a dropdown for 'Actions' with options: Create Stack, Update Stack, and Delete Stack. The 'Delete Stack' option is highlighted with a red box. A table below shows one stack entry:

Name	Created Time	Status	Description
ec2instanceDemoStack	2014-06-23 12:59:48 UTC+0100	UPDATE_COMPLETE	Create an EC2 instance running the latest Amazon Linux AMI.

Below the stack table is an 'Events' section with a table showing the history of events for the stack:

Date	Status	Type	Logical ID	Status Reason
2014-06-23	13:45:53 UTC+0100	UPDATE_COMPLETE	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:45:53 UTC+0100	DELETE_COMPLETE	AWS::EC2::Instance	Ec2Instance
	13:45:27 UTC+0100	DELETE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:45:25 UTC+0100	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:45:23 UTC+0100	UPDATE_COMPLETE	AWS::EC2::Instance	Ec2Instance
	13:44:35 UTC+0100	UPDATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:44:34 UTC+0100	UPDATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:44:27 UTC+0100	UPDATE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:55 UTC+0100	UPDATE_ROLLBACK_COMPLETE	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:54 UTC+0100	UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:52 UTC+0100	UPDATE_COMPLETE	AWS::EC2::Instance	Ec2Instance

At the bottom of the page, there are links for Feedback, Help, and a search bar.

Using a template to create and manage a stack

The screenshot shows the CloudFormation Management Console interface. At the top, there's a navigation bar with tabs for Services, VPC, EC2, S3, RDS, CloudFront, and Help. The user is signed in as 'ianmas @ ianmas-aws'. Below the navigation bar, there's a search bar with 'Filter: Active' and 'By Name:' and a dropdown for 'Actions' with options: Create Stack, Update Stack, and Delete Stack. The 'Delete Stack' option is highlighted with a red box. A table below shows one stack entry:

Stack Name	Created Time	Status	Description
ec2instanceDemoStack		ACTIVE	

Below the stack table is an 'Events' section with a table showing the history of events for the stack:

Date	Status	Type	Logical ID	Status Reason
2014-06-23	13:53:48 UTC+0100	DELETE_COMPLETE	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:53:45 UTC+0100	DELETE_COMPLETE	AWS::EC2::Instance	Ec2Instance
	13:53:17 UTC+0100	DELETE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:52:55 UTC+0100	DELETE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:45:53 UTC+0100	UPDATE_COMPLETE	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:45:53 UTC+0100	DELETE_COMPLETE	AWS::EC2::Instance	Ec2Instance
	13:45:27 UTC+0100	DELETE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:45:25 UTC+0100	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:45:23 UTC+0100	UPDATE_COMPLETE	AWS::EC2::Instance	Ec2Instance
	13:44:35 UTC+0100	UPDATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:44:34 UTC+0100	UPDATE_IN_PROGRESS	AWS::EC2::Instance	Ec2Instance
	13:44:27 UTC+0100	UPDATE_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:55 UTC+0100	UPDATE_ROLLBACK_COMPLETE	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:54 UTC+0100	UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack	ec2instanceDemoStack
	13:37:52 UTC+0100	UPDATE_COMPLETE	AWS::EC2::Instance	Ec2Instance

At the bottom of the page, there are links for Feedback, Help, and a search bar.

Using a template
to create and
manage a stack

A screenshot of the EC2 Management Console. The left sidebar shows various services like VPC, EC2, S3, RDS, CloudFront, and CloudWatch Metrics. Under the EC2 section, 'Instances' is selected. The main pane shows a table of instances. One instance, 'i-86872bc5', is highlighted with a blue border and has its details expanded. The status is listed as 'terminated'. A large blue diagonal banner across the top right of the screenshot reads 'Cleans Up Resources'.

Name	Instance ID	Instance Type	Availability Zone	Instance State
i-86872bc5	i-86872bc5	m3.medium	eu-west-1a	terminated
i-d2af0792	i-d2af0792	m3.medium	eu-west-1c	terminated

Instance: i-86872bc5 Public DNS: -

Description	Status Checks	Monitoring	Tags
Instance ID: i-86872bc5	Instance state: terminated	Public DNS: -	Public IP: -
Instance type: m3.medium	Private DNS: -	Elastic IP: -	Availability zone: eu-west-1a
Secondary private IPs: -	Private IPs: -	Security groups: -	Scheduled events: -
VPC ID: -	AMI ID: amzn-ami-hvm-2014.03.2.x86_64-gp2 (ami-d9f92b5aa)		

Using a template to create and manage a stack via the AWS CLI

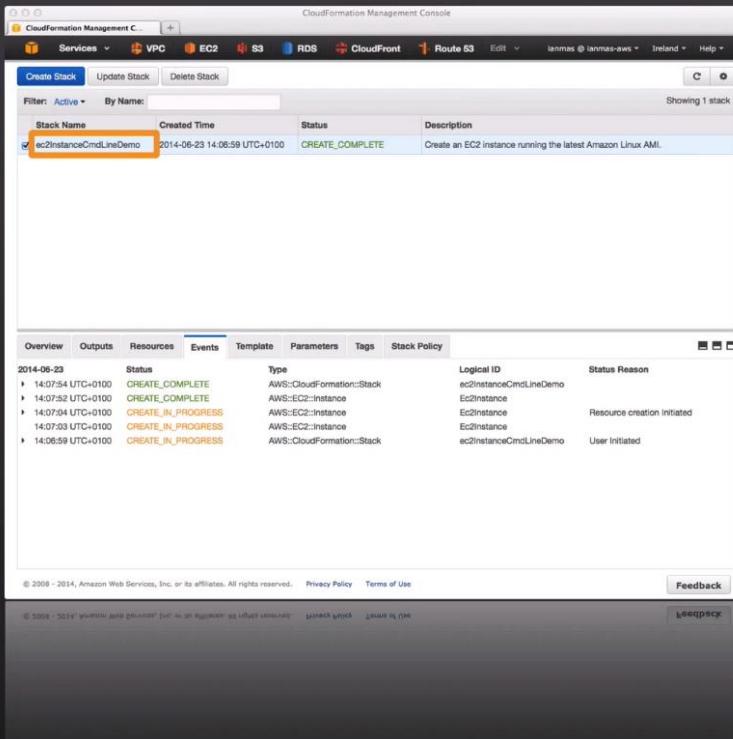
```
aws cloudformation create-stack
  --stack-name ec2InstanceCmdLineDemo
  --template-url https://s3-eu-west-1.amazonaws.com/cf-
templates-1fhelyvrvdrbr-eu-west-1/2014174d0r-ec2Instance.template
  --parameters ParameterKey=KeyPair,ParameterValue=ManagementKeyPair
```

Returns the details of the created stack, in the output format of your choice

```
arn:aws:cloudformation:eu-west-1:554625704737:stack/ec2InstanceCmdLineDemo/
42cc6150-fad7-11e3-8f4d-5017e1aef4e7
```

You can also execute the same stack create operation using the AWS CLI commands above

Using a template to create and manage a stack via the AWS CLI



Other AWS CLI actions for CloudFormation

cancel-update-stack	get-stack-policy
create-stack	get-template
delete-stack	list-stack-resources
describe-stack-events	list-stacks
describe-stack-resource	set-stack-policy
describe-stack-resources	update-stack
describe-stacks	validate-template

As usual, you can get more details via the AWS CLI

```
$ aws cloudformation update-stack help
```

Help via the AWS CLI

```
$ aws cloudformation update-stack help
```

SYNOPSIS

```
    update-stack
    --stack-name <value>
    [--template-body <value>]
    [--template-url <value>]
    [--use-previous-template | --no-use-previous-template]
    [--stack-policy-during-update-body <value>]
    [--stack-policy-during-update-url <value>]
    [--parameters <value>]
    [--capabilities <value>]
    [--stack-policy-body <value>]
    [--stack-policy-url <value>]
    [--notification-arns <value>]
```

```
$ aws cloudformation update-stack help
```

SYNOPSIS

```
    update-stack
    --stack-name <value>
    [--template-body <value>]
    [--template-url <value>]
    [--use-previous-template | --no-use-previous-template]
    [--stack-policy-during-update-body <value>]
    [--stack-policy-during-update-url <value>]
    [--parameters <value>]
    [--capabilities <value>]
    [--stack-policy-body <value>]
    [--stack-policy-url <value>]
    [--notification-arns <value>]
```

Built using the
CloudFormation API

CloudFormation API Reference : [That concludes the introduction for creating, updating, and deleting a CF stack from the console and the CLI.](https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference>Welcome.html</p></div><div data-bbox=)

WORKING WITH AWS RESOURCES

Let us see how to define the resources in CF that we want to use in our stack

Designed to use your existing experience with AWS

Each resource has a set of parameters with names that are identical to the names used to create the resources through their native API

Template reference: docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-reference.html

```
"myVolume" : {  
    "Type" : "AWS::EC2::Volume",  
    "Properties" : {  
        "Size" : "10",  
        "SnapshotId" : "snap-7b8fd361",  
        "AvailabilityZone" : "eu-west-1a"  
    }  
}
```

This example defines an Amazon EBS Volume with a logical name ‘myVolume’. Its type is “AWS::EC2::Volume”

If you’ve used EBS previously, the properties should look very familiar

```
"InstanceSecurityGroup" : {  
    "Type" : "AWS::EC2::SecurityGroup",  
    "Properties" : {  
        "GroupDescription" : "Enable SSH access via port 22",  
        "SecurityGroupIngress" : [ {  
            "IpProtocol" : "tcp",  
            "FromPort" : "22",  
            "ToPort" : "22",  
            "CidrIp" : "0.0.0.0/0"  
        } ]  
    }  
}
```

Creating a Security Group resource

Supported AWS Services:

- Auto Scaling
- Amazon CloudFront
- AWS CloudWatch
- Amazon DynamoDB
- Amazon EC2
- Amazon ElastiCache
- AWS Elastic Beanstalk
- AWS Elastic Load Balancing
- AWS Identity and Access Management
- Amazon RDS
- Amazon Redshift
- Amazon Route 53
- Amazon S3
- Amazon SimpleDB
- Amazon SNS
- Amazon SQS
- Amazon VPC

REFERENCING THE PROPERTIES OF ANOTHER RESOURCE

This is useful for avoiding namespace collisions or avoiding collisions when using the same templates.

```
{
  "Resources" : {
    "Ec2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
        "KeyName" : "mykey",
        "ImageId" : "ami-7a11e213"
      }
    },
    "InstanceSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable SSH access via port 22",
        "SecurityGroupIngress" : [ {
          "IpProtocol" : "tcp",
          "FromPort" : "22",
          "ToPort" : "22",
          "CidrIp" : "0.0.0.0/0" } ]
      }
    }
  }
}
```

One of the ways to avoid this is by using Reference functions **Ref** within your CF templates, this will allow you to **back reference** to resources that are being created within the same CF template. Above we can see an example where we are creating a reference Ref called **InstanceSecurityGroup**, which is later used in creating the security group itself. These referenced resources will be given execution time names by CF itself so that you don't have to be explicit about the names of the resources before creating them.

```
{
  "Resources" : {
    "Ec2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" }, ,
"MyExistingSG" ], "KeyName" : "mykey",
        "ImageId" : "ami-7a11e213" }
    },
    "InstanceSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable SSH access via port 22",
        "SecurityGroupIngress" : [ {
          "IpProtocol" : "tcp",
          "FromPort" : "22",
          "ToPort" : "22",
          "CidrIp" : "0.0.0.0/0" } ]
      }
    }
  }
}
```

There are also situations where you may need to refer to resources that have been created outside of CF, like if you have pre-existing VPC with security groups in it that you want to make use of or you want to statically find a keypair that you want to use for SSH access. You can simply use literal references for accessing pre-existing resources within our account as above.

REFERENCING INPUT PARAMETERS

```
{  
  "Parameters" : {  
    "KeyPair" : {  
      "Description" : "The EC2 Key Pair to allow SSH access to the instance",  
      "Type" : "String"  
    },  
    "Resources" : {  
      "Ec2Instance" : {  
        "Type" : "AWS::EC2::Instance",  
        "Properties" : {  
          "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],  
          "KeyName" : { "Ref" : "KeyPair" },  
          "ImageId" : ""  
        },  
        ...  
      } }  
}
```

This is how we can use Ref to reference a parameter within a resource to be used at execution time.

The screenshot shows the AWS CloudFormation console interface. A blue diagonal banner in the top right corner reads "Input Parameters". The main window is titled "Specify Parameters" and displays the following information:

- Select Template:** Options, Review
- Specify Parameters:** Options, Review
- Specify Values or Default Values:** Specify values or use the default values for the parameters that are associated with your AWS CloudFormation template.
- Parameters:** A table with one row:

KeyPair	ManagementKeyPair	The EC2 Key Pair to allow SSH access to the instance
----------------	--------------------------	--
- Buttons:** Cancel, Previous, Next
- Footer:** © 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Feedback

```

"WordPressUser": {
    "Default": "admin",
    "Description" : "The WordPress database admin account username",
    "Type": "String",
    "MinLength": "1",
    "MaxLength": "16",
    "AllowedPattern" : "[a-zA-Z][a-zA-Z0-9]*"
},

```

Validate your input parameters with :

Maxlength, MinLength, MaxValue, MinValue, AllowedPattern, AllowedValues

CONDITIONAL VALUES

```

{"Mappings" : {
    "RegionMap" : {
        "us-east-1" : { "AMI" : "ami-76f0061f" },
        "us-west-1" : { "AMI" : "ami-655a0a20" },
        "eu-west-1" : { "AMI" : "ami-7fd4e10b" },
        "ap-southeast-1" : { "AMI" : "ami-72621c20" },
        "ap-northeast-1" : { "AMI" : "ami-8e08a38f" } } },
    "Resources" : {
        "Ec2Instance" : {
            "Type" : "AWS::EC2::Instance",
            "Properties" : {
                "KeyName" : { "Ref" : "KeyName" },
                "ImageId" : {
                    "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]
                }
            }
        }
    }
}

```

This is important because there might be region-specific parameters that you might require within a template. An example is when we use a CF template to create an EC2 instance in a particular region, then we want to make this template also portable across different regions. We may need different AMI IDs in those different regions, we can use **mappings** to handle how to get these AMI IDs in the different regions by creating region map **RegionMap** as above. We have defined a 2-level map with the lower level as tuples containing the AMI IDs that we will be using. **For the ap-northeast-1 region, the above will automatically set the ImageId to the AMI image Id to ami-8e08a38f.**

Other intrinsic functions and pseudo parameters

Intrinsic functions	Pseudo parameters
Fn::Base64	AWS::NotificationARNs
Fn::FindInMap	AWS::Region
Fn::GetAtt	AWS::StackId
Fn::GetAZs	AWS::StackName
Fn::Join	
Fn::Select	
Ref	

Intrinsic Functions: The **Fn::Base64** function is typically used to pass in coded data to EC2 instances through the use of data property. The **Fn::FindInMap** function is used for finding values corresponding to specific keys in a 2-level map declared in the mapping section of a CF template. The **Fn::GetAtt** function returns the value of an attribute from a resource in the CF template. The **Fn::GetAZs** function returns an array listing all the availability zones for the specified region, **Fn::Join** allows you to combine several values into a single value separated by a specified delimiter, **Fn::Select** enables you to select and return a single object from a list of objects by index, **Ref** enables you to return the value of a specified parameter or resource.

Pseudo Parameters: The **AWS::NotificationsARNs** pseudo-parameter returns a list of notification ARNs amazon resource names for the current stack. **AWS::Region** returns a string representing the region in which the resource is being created. **AWS::StackId** and **AWS::StackName** are for returning the stack Id and the stack name.

Working with non-AWS Resources

Defining custom resources allows you to include non-AWS resources in a CloudFormation stack

More on Custom Resources in 'AWS CloudFormation under the Hood' from re:Invent 2013: <http://youtu.be/ZhGMaw67Yu0>
AWS CloudFormation Custom Resource Walkthrough documentation:

docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/crpg-walkthrough.html

BOOTSTRAPPING APPLICATIONS AND HANDLING UPDATES

Option 1: Continue to use EC2 UserData, which is available as a property of AWS::EC2::Instance resources

```
"Resources" : {
    "Ec2Instance" : {
        "Type" : "AWS::EC2::Instance",
        "Properties" : {
            "KeyName" : { "Ref" : "KeyName" },
            "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
            "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ] },
            "UserData" : { "Fn::Base64" : { "Fn::Join" : [ "", [
                "#!/bin/bash -ex",
                "yum -y install gcc-c++ make",
                "yum -y install mysql-devel sqlite-devel",
                "yum -y install ruby-rdoc rubygems ruby-mysql ruby-devel",
                "gem install --no-ri --no-rdoc rails",
                "gem install --no-ri --no-rdoc mysql",
                "gem install --no-ri --no-rdoc sqlite3",
                "rails new myapp",
                "cd myapp",
                "rails server -d",
                "curl -X PUT -H 'Content-Type:' --data-binary '{\"Status\": \"SUCCESS\", \"Reason\": \"The application myapp is ready\", \"UniqueId\": \"myapp\", \"Data\": \"Done\"}' \"\"", { "Ref" : "WaitForInstanceWaitHandle"}, "\"\n" ]]} }
        }
    }
}
```

Option 2: AWS CloudFormation provides helper scripts for deployment within your EC2 instances



Metadata Key — `AWS::CloudFormation::Init`

Cfn-init reads this metadata key and installs the packages listed in this key (e.g., httpd, mysql, and php). Cfn-init also retrieves and expands files listed as sources.

Installing Packages & Expanding Files

```
"Resources" : {
    "WebServer": {
        "Type": "AWS::EC2::Instance",
        "Metadata" : {
            "Comment1" : "Configure the bootstrap helpers to install the Apache Web Server and PHP",
            "Comment2" : "The website content is downloaded from the CloudFormationPHPSample.zip file",
        }
        "AWS::CloudFormation::Init" : {
            "config" : {
                "packages" : {
                    "yum" : {
                        "mysql" : [],
                        "mysql-server" : [],
                        "mysql-libs" : [],
                        "httpd" : [],
                        "php" : [],
                        "php-mysql" : []
                    }
                },
                "sources" : {
                    "/var/www/html" : "https://s3.amazonaws.com/cloudformation-examples/CloudFormationPHPSample.zip"
                }
            }
        }
    }
}
```

Installing & executing CloudFormation helper

The `UserData` key allows you to execute shell commands.

This template issues two shell commands: the first command installs the AWS CloudFormation helper scripts; the second executes the `cfn-init` script.

```
"Properties": {
    "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },
        { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] },
    "InstanceType" : { "Ref" : "InstanceType" },
    "SecurityGroups" : [ {"Ref" : "WebServerSecurityGroup"} ],
    "KeyName" : { "Ref" : "KeyName" },
    "UserData" : { "Fn::Base64" : { "Fn::Join" : [ "", [
        "#!/bin/bash -v\n",
        "yum update -y aws-cfn-bootstrap\n",

        "# Install packages\n",
        "/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" }, " -r WebServer ",
        " --region ", { "Ref" : "AWS::Region" }, " || error_exit 'Failed to run cfn-init'\n"
    ]]} }
},
```

The files key allows you to write files to the instance filesystem

```
"files" : {  
    "/tmp/setup.mysql" : {  
        "content" : { "Fn::Join" : [ "", [  
            "CREATE DATABASE ", { "Ref" : "DBName" }, ";\\n",  
            "GRANT ALL ON ", { "Ref" : "DBName" }, ".* TO '", { "Ref" : "DBUsername" }, "'@localhost IDENTIFIED BY  
'", { "Ref" : "DBPassword" },"';\\n"  
        ]]},  
        "mode" : "000644",  
        "owner" : "root",  
        "group" : "root"  
    }  
}
```

The services key allows you ensures that the services are not only running when cfn-init finishes (ensureRunning is set to true); but that they are also restarted upon reboot (enabled is set to true).

```
"services" : {  
    "sysvinit" : {  
        "mysqld" : {  
            "enabled" : "true",  
            "ensureRunning" : "true"  
        },  
        "httpd" : {  
            "enabled" : "true",  
            "ensureRunning" : "true"  
        }  
    }  
}
```

More on Deploying Applications with AWS CloudFormation:

docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/deploying.applications.html

Yes!

All this functionality is available for Windows instances too!

Bootstrapping AWS CloudFormation Windows Stacks:

docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-windows-stacks-bootstrapping.html

What about Chef?

and/or

What about Puppet?

AWS CloudFormation Articles and Tutorials

Create Free Account

AWS CloudFormation gives developers and systems administrators an easy way to create a collection of related AWS resources and provision them in an orderly and predictable fashion. The following articles and documents provide guidance on building templates and using the various AWS CloudFormation features to provision your AWS resources.

Bootstrapping Applications via AWS CloudFormation

AWS CloudFormation gives you an easy way to create the set of resources such as Amazon EC2 instance, Amazon RDS database instances and Elastic Load Balancers needed to run your application. The template describes what resources you need and AWS CloudFormation takes care of how: provisioning the resources in an orderly and predictable fashion, handling and recovering from any failures or issues. While AWS CloudFormation takes care of provisioning all the resources, it raises the obvious question of how your application software is deployed, configured and executed on the Amazon EC2 instances. There are many options, each of which has implications on how quickly your application is ready and how flexible you need to be in terms of deploying new versions of the software.

[Read on... ↗](#)

Integrating AWS CloudFormation with Opscode Chef

AWS CloudFormation can help you to configure and/or install your application as well as how to bootstrap deployment and management tools that you may already use in your environment. Chef is an open source infrastructure automation solution from Opscode, written in Ruby, that allows you to automate the configuration of your systems and the applications that sit on top of it. AWS CloudFormation and Chef can be used together to automate your entire deployment and management processes, from your AWS resources through to your application artifacts.

[Read on... ↗](#)

Integrating AWS CloudFormation with Puppet

AWS CloudFormation can help you to configure and/or install your application as well as how to bootstrap deployment and management tools that you may already use in your environment. Puppet is an open source platform for provisioning, configuring and patching applications and operating system components. AWS CloudFormation and Puppet can be used together to automate your entire deployment and management processes, from your AWS resources through to your application artifacts.

[Read on... ↗](#)

Find out more here: aws.amazon.com/cloudformation/aws-cloudformation-articles-and-tutorials/

SUMMARY

- ① An easy way to create & manage a collection of AWS resources
- ② Allows orderly and predictable provisioning and updating of resources
- ③ Allows you to version control your AWS infrastructure
- ④ Deploy and update stacks using console, command line or API

RESOURCES YOU CAN USE TO LEARN MORE

aws.amazon.com/cloudformation/

Getting Started with AWS CloudFormation:

aws.amazon.com/cloudformation/getting-started/

AWS CloudFormation Templates & Samples:

aws.amazon.com/cloudformation/aws-cloudformation-templates/

AWS cfncluster HPC deployment framework:

github.com/awslabs/cfncluster/

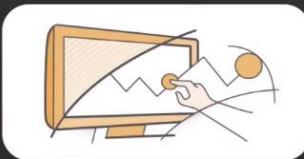
CloudFormation Template to Deploy Wordpress



https://s3-us-west-1.amazonaws.com/cloudformation-templates-us-west-1/WordPress_Multi_AZ.template

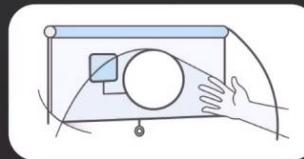
AWS Training & Certification

Self-Paced Labs



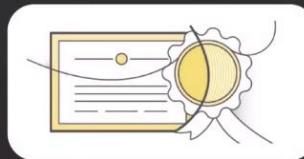
Try products, gain new skills, and get hands-on practice working with AWS technologies

Training



Build technical expertise to design and operate scalable, efficient applications on AWS

Certification



Validate your proven skills and expertise with the AWS platform

[aws.amazon.com/training/
self-paced-labs](https://aws.amazon.com/training/self-paced-labs)

aws.amazon.com/training

aws.amazon.com/certification