amazon web services

Masterclass

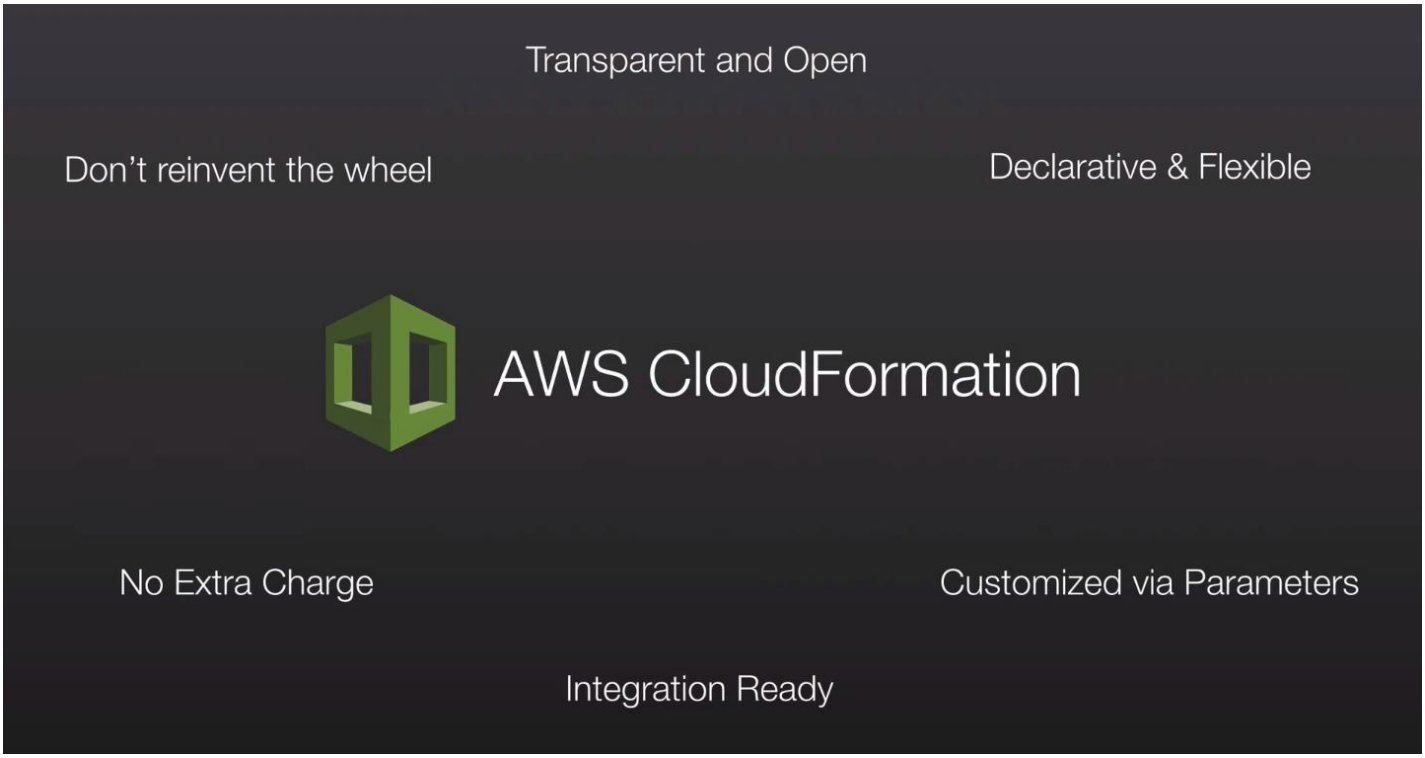
AWS CloudFormation

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

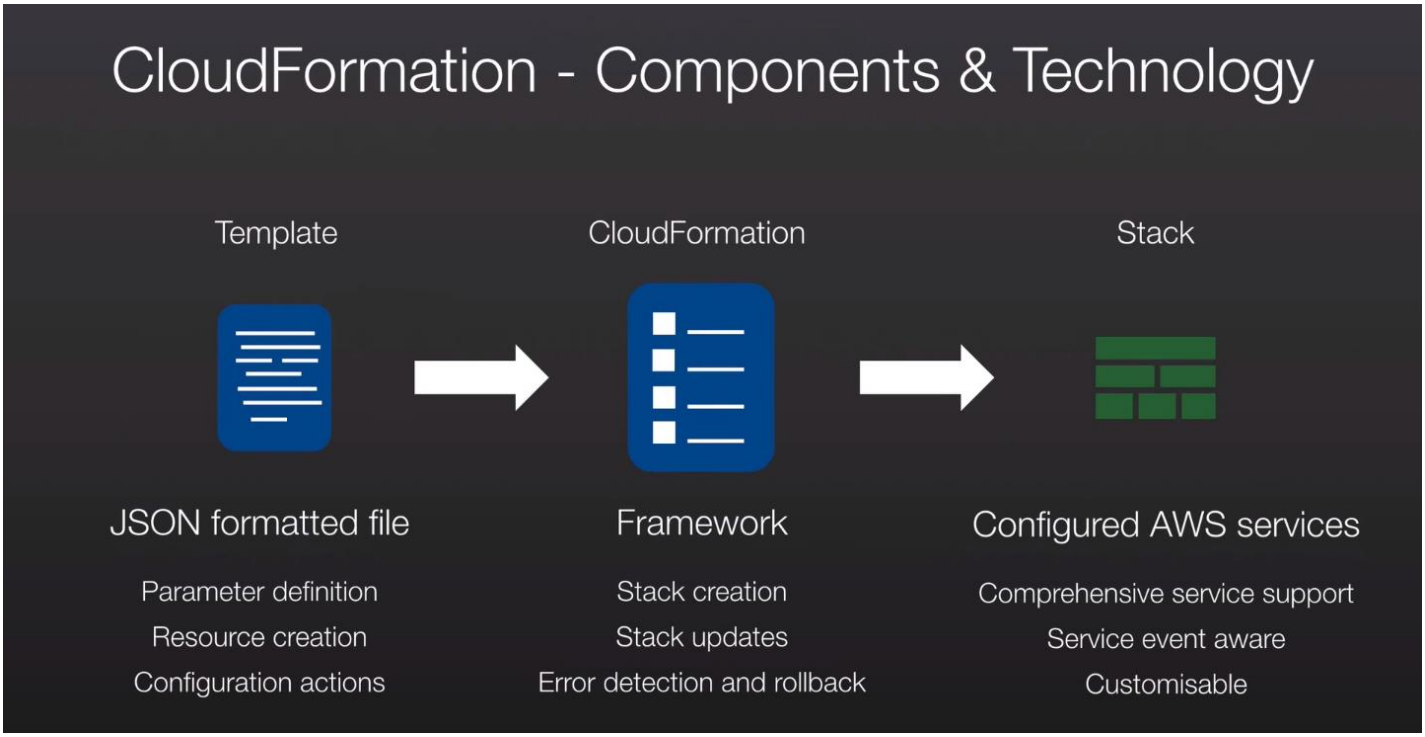
An easy way to create & manage a collection of AWS resources
Allows orderly and predictable provisioning and updating of resources
Allows you to version control your AWS infrastructure
Deploy and update stacks using console, command line or API
You only pay for the resources you create

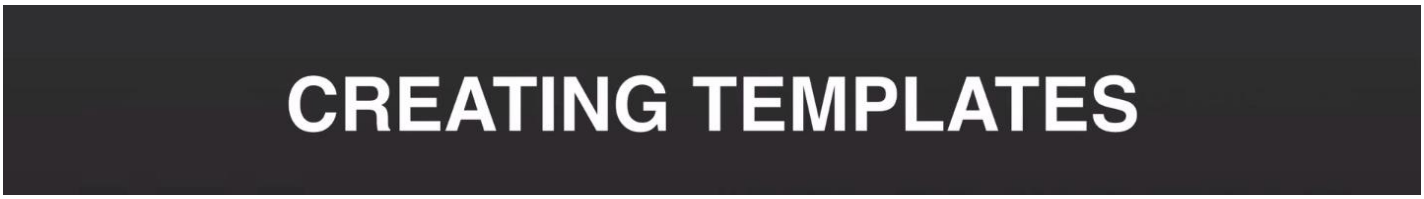


You can use the same template to create identical copies of the same stack of resources. You also do not need to write these templates from scratch because there are available sample templates that you can use or customize.

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.





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

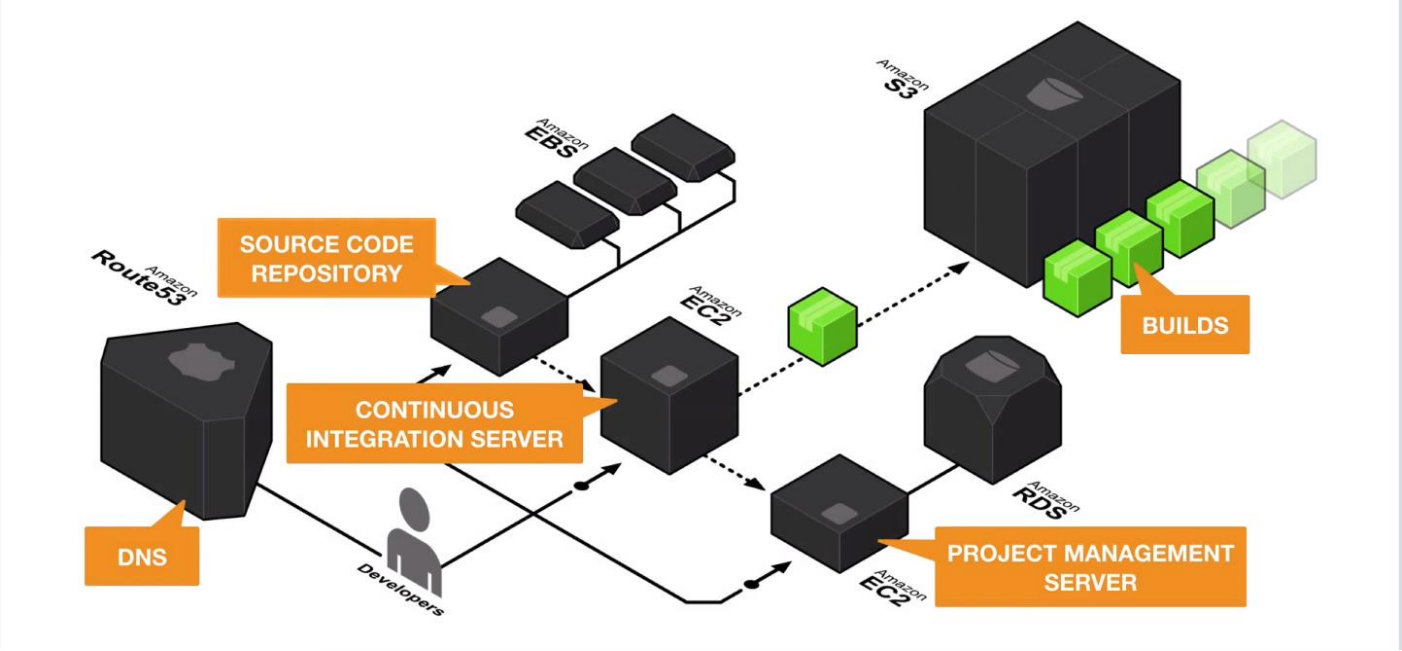
```
{
    "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",
    "Parameters": {
        "KeyPair": {
            "Description": "The EC2 Key Pair to allow SSH access to the instance",
            "Type": "String"
        }
    },
    "Resources": {
        "Ec2Instance": {
            "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

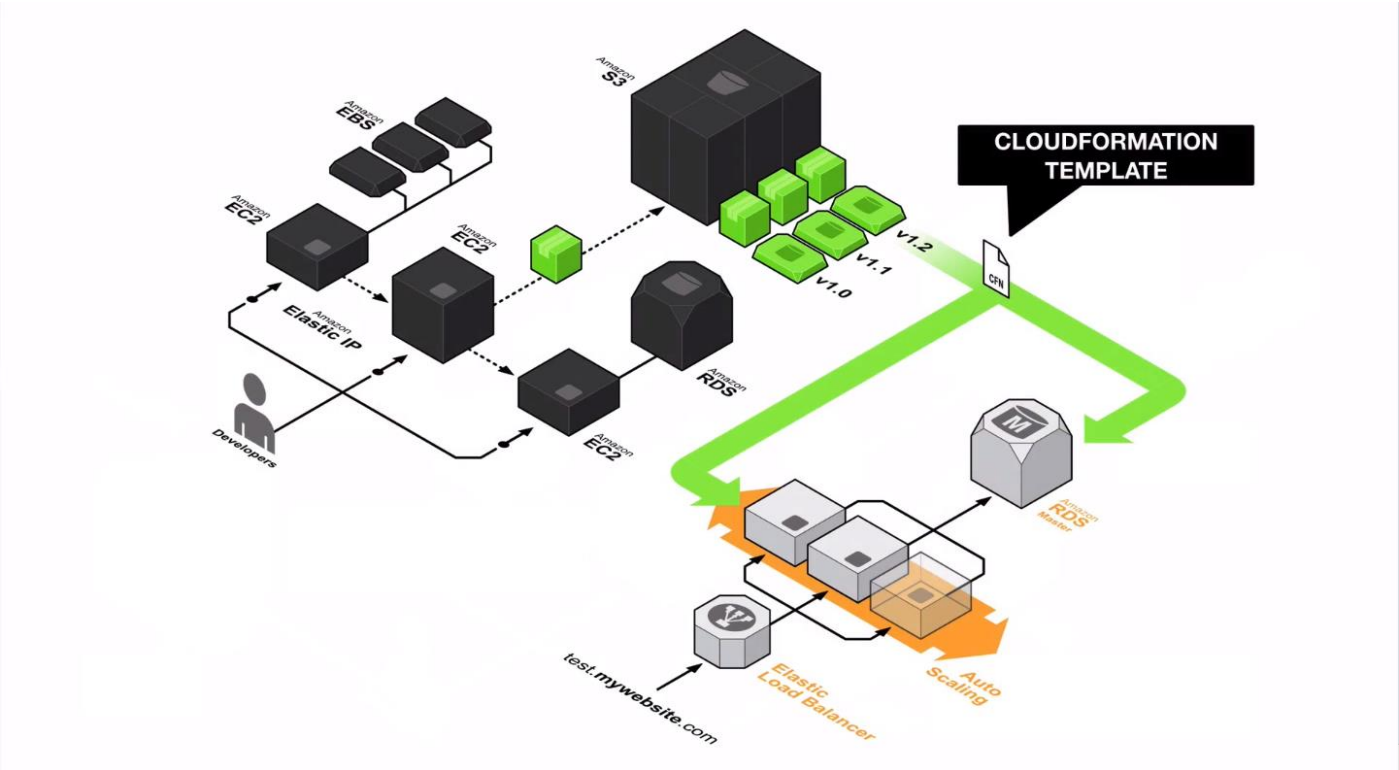
A Simple Template that creates an EC2 Instance

```
{
    "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"
}
```

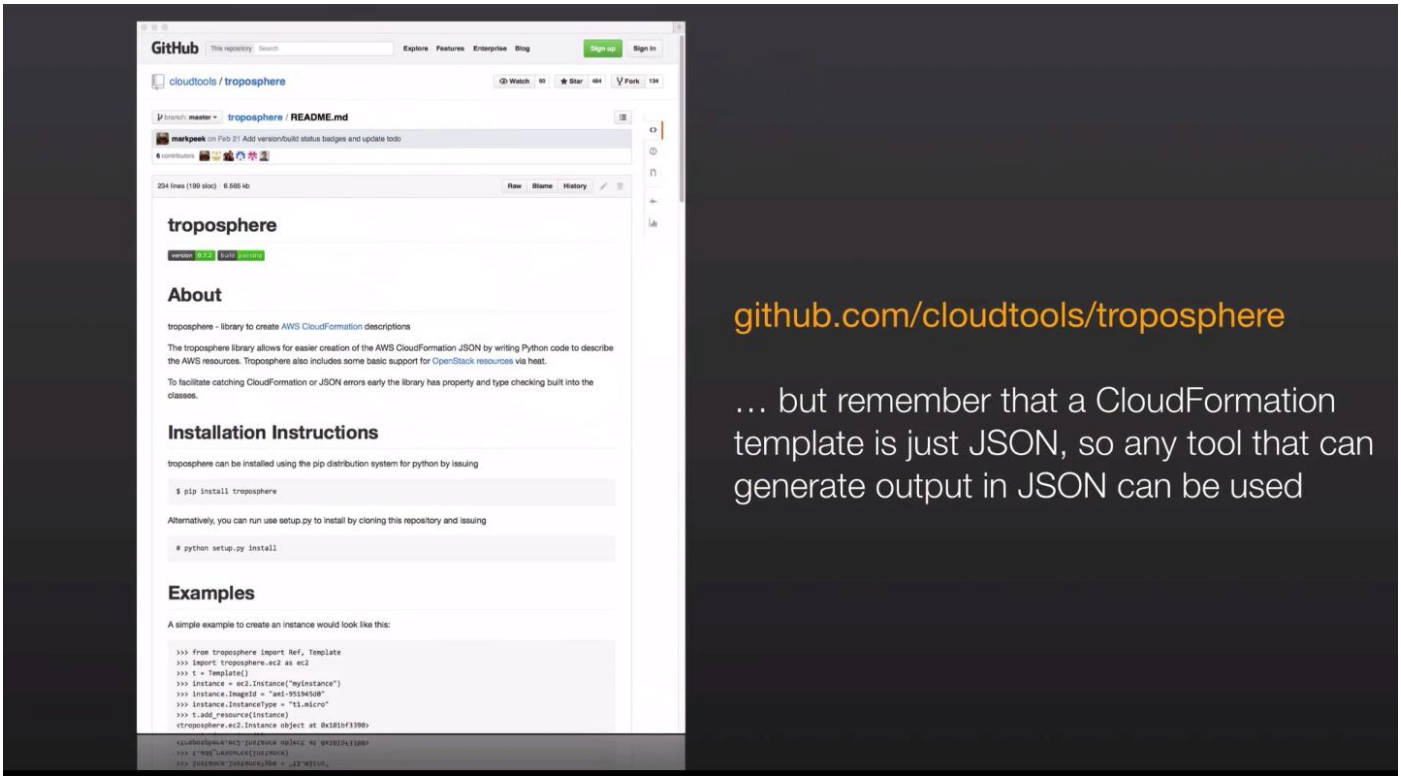
These outputs will be returned once the template has completed execution



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.



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

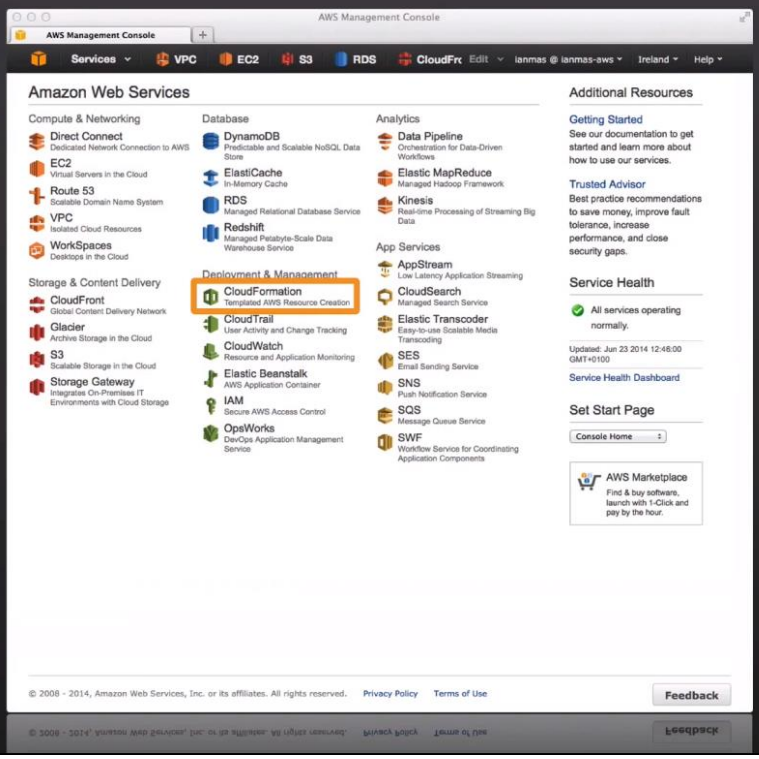
Using a template to create and manage a stack

ec2Instance.template

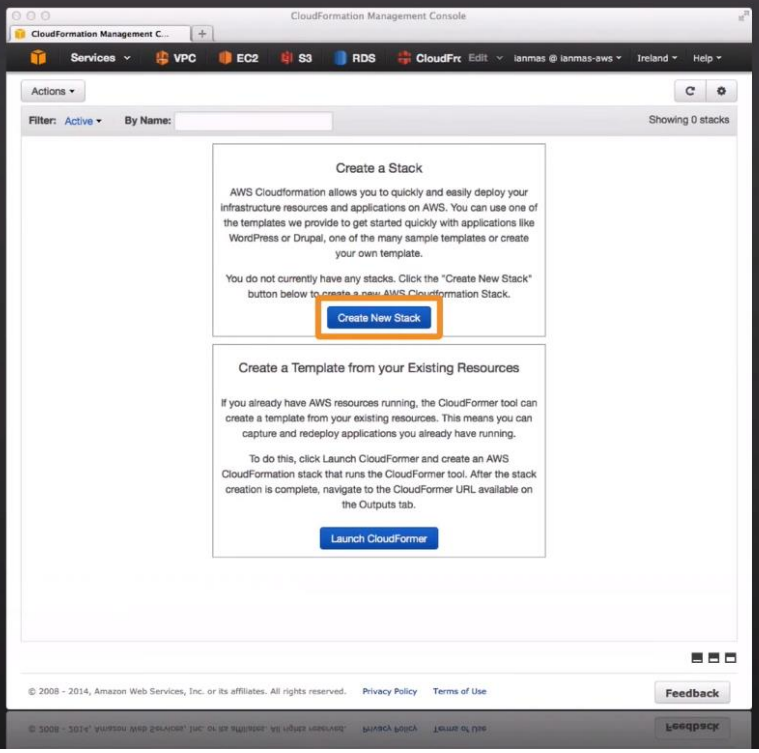
```
{
    "Description": "Create an EC2 instance running the latest Amazon Linux AMI.",
    "Parameters": {
        "KeyPair": {
            "Description": "The EC2 Key Pair to allow SSH access to the instance",
            "Type": "String"
        }
    },
    "Resources": {
        "Ec2Instance": {
            "Properties": {
                "ImageId": "ami-dd925baa",
                "InstanceType" : "m3.medium",
                "KeyName": {
                    "Ref": "KeyPair"
                }
            },
            "Type": "AWS::EC2::Instance"
        }
    },
    "Outputs": {
        "InstanceId": {
            "Description": "The InstanceId of the newly created EC2 instance",
            "Value": {
                "Ref": "Ec2Instance"
            }
        }
    },
    "AWSTemplateFormatVersion": "2010-09-09"
}
```

Line: 17:15 | Plain Text | Tab Size: 4

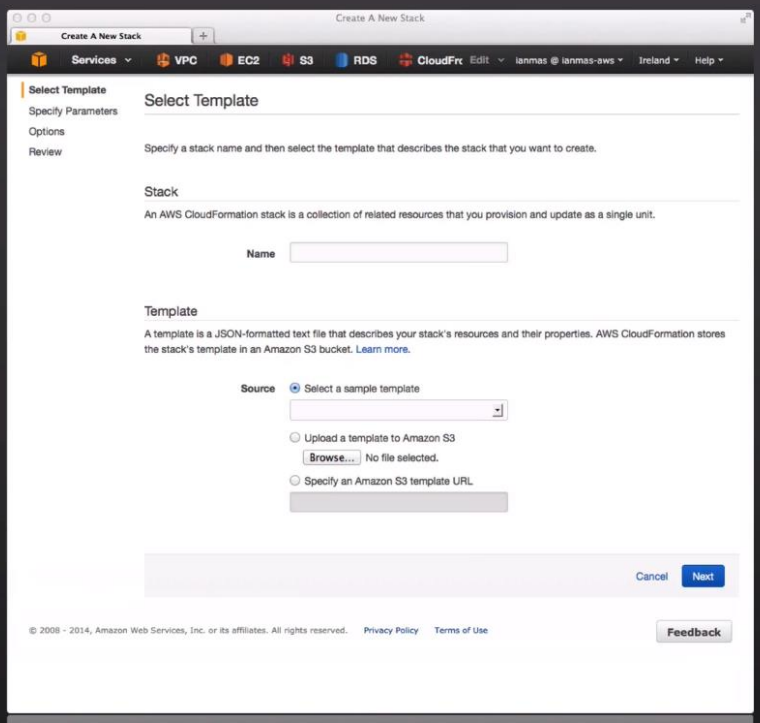
Using a template to create and manage a stack



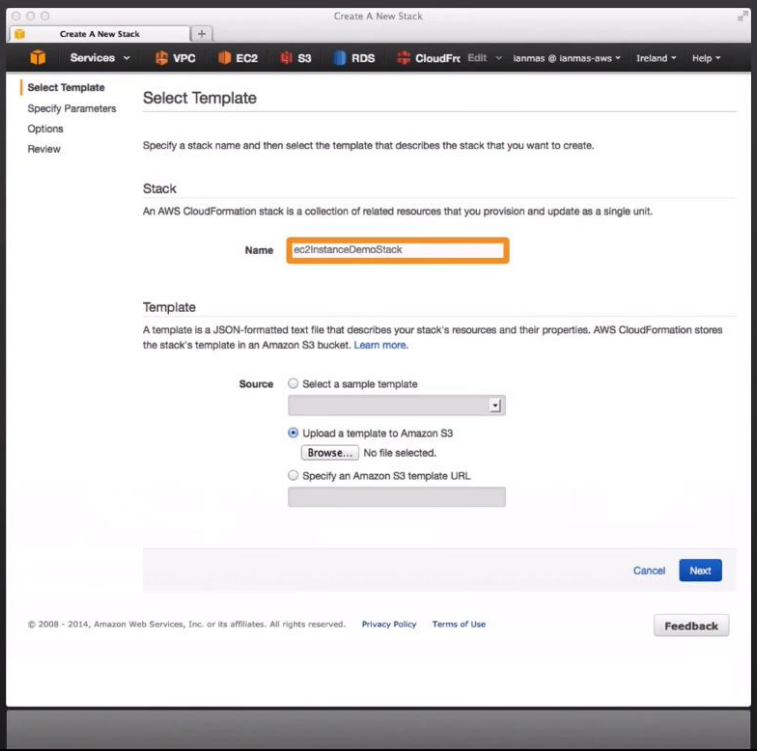
Using a template to create and manage a stack



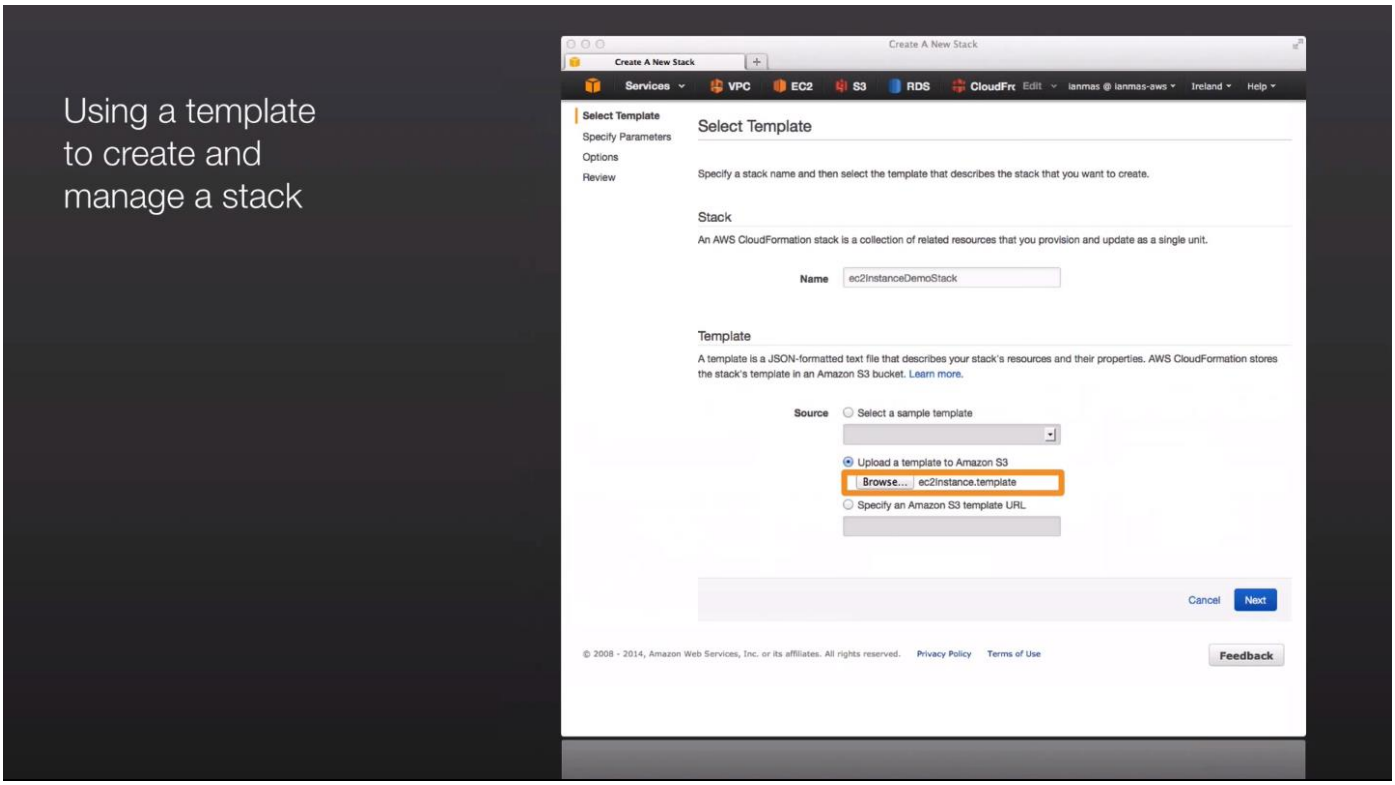
Using a template to create and manage a stack



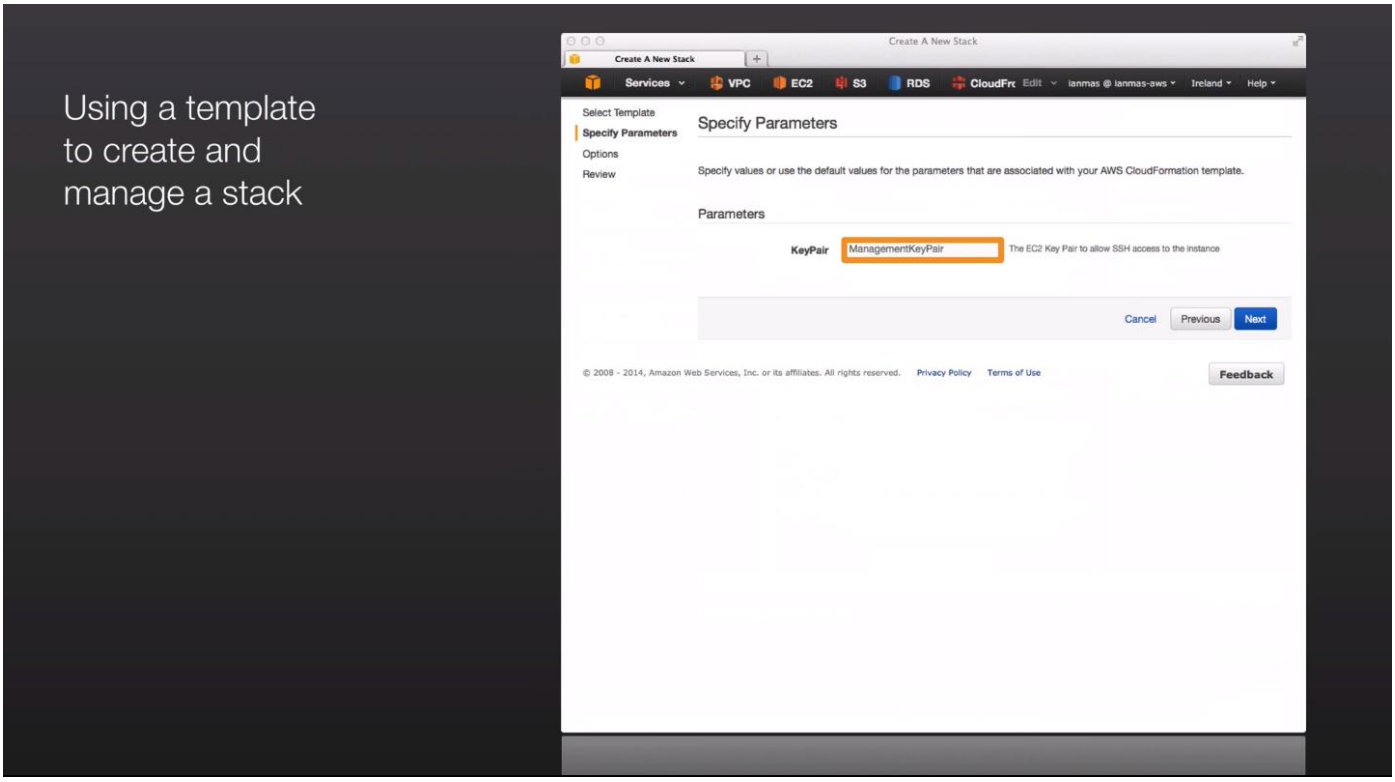
Using a template to create and manage a stack



We give the name of the stack we want to create

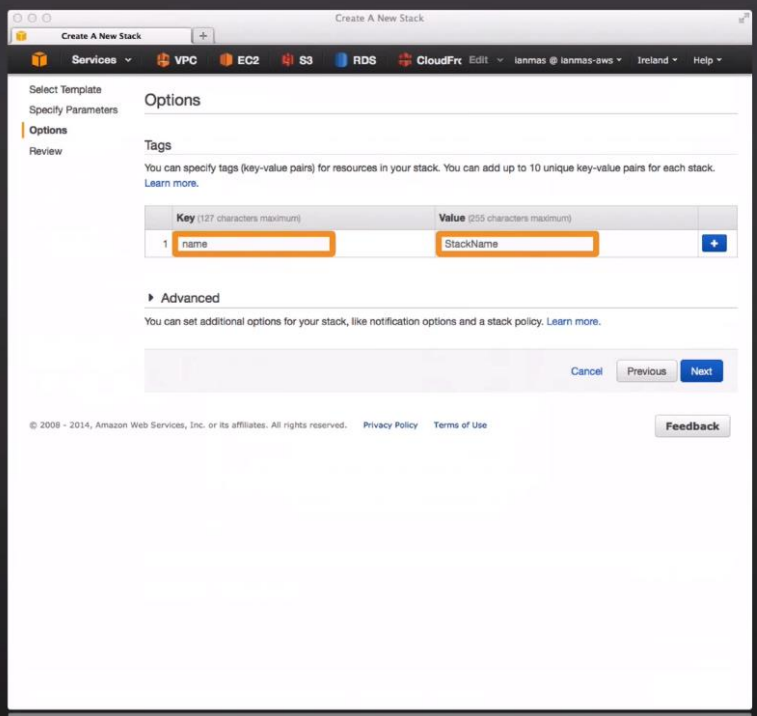


We will be uploading the created template to S3. A best practice will be to use IAM to restrict access to the S3 bucket being used to hold the CF templates.



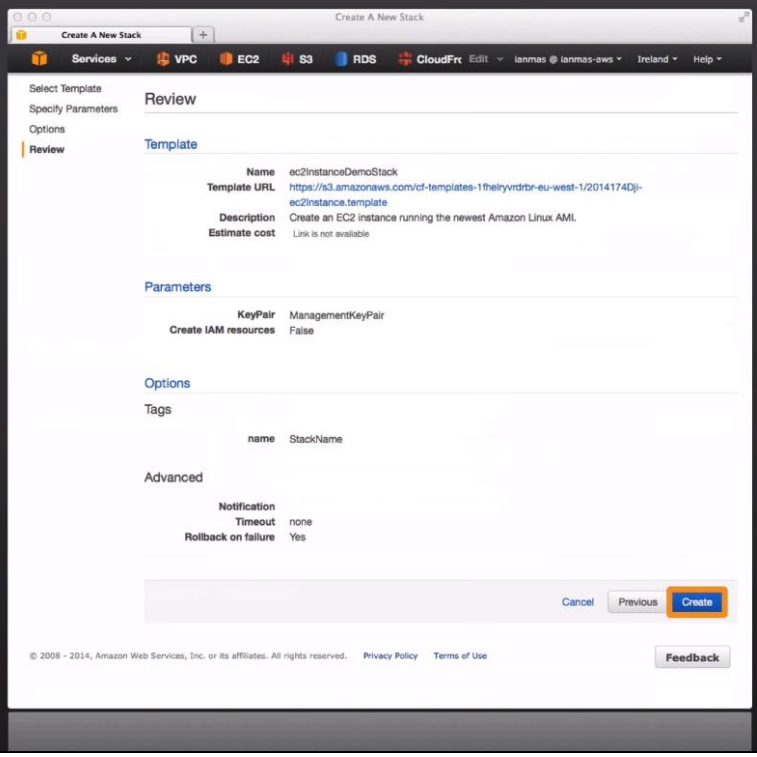
You then will be asked to specify the parameters needed for the CF template

Using a template to create and manage a stack



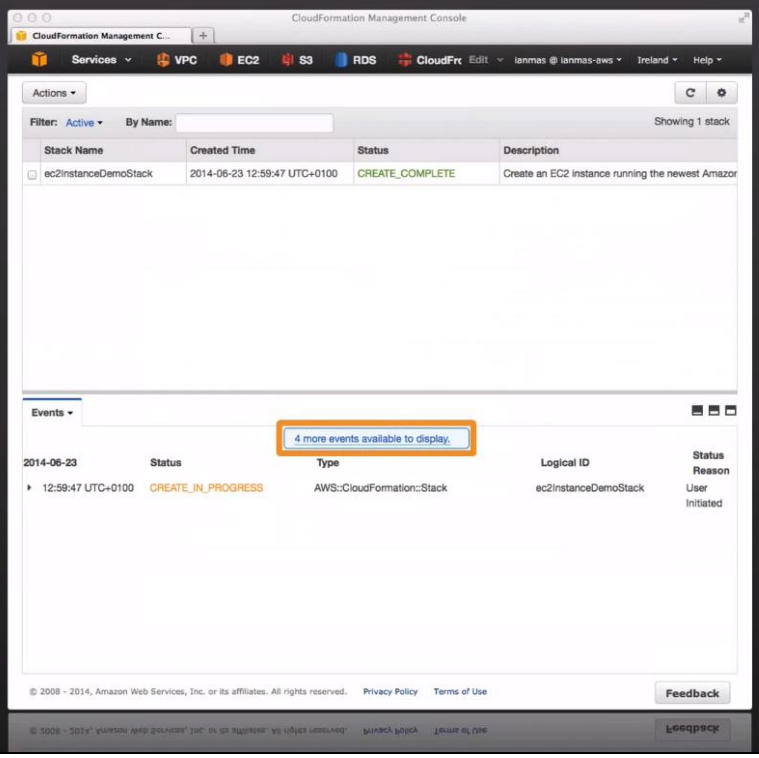
You then will be asked to name your stack for later use

Using a template to create and manage a stack

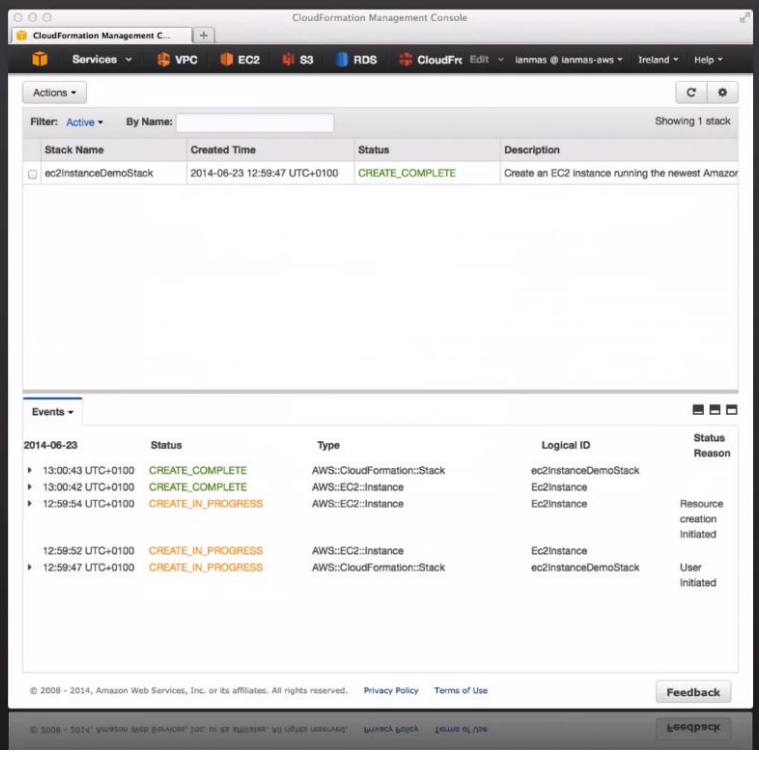


Review and click Create button if okay

Using a template to create and manage a stack

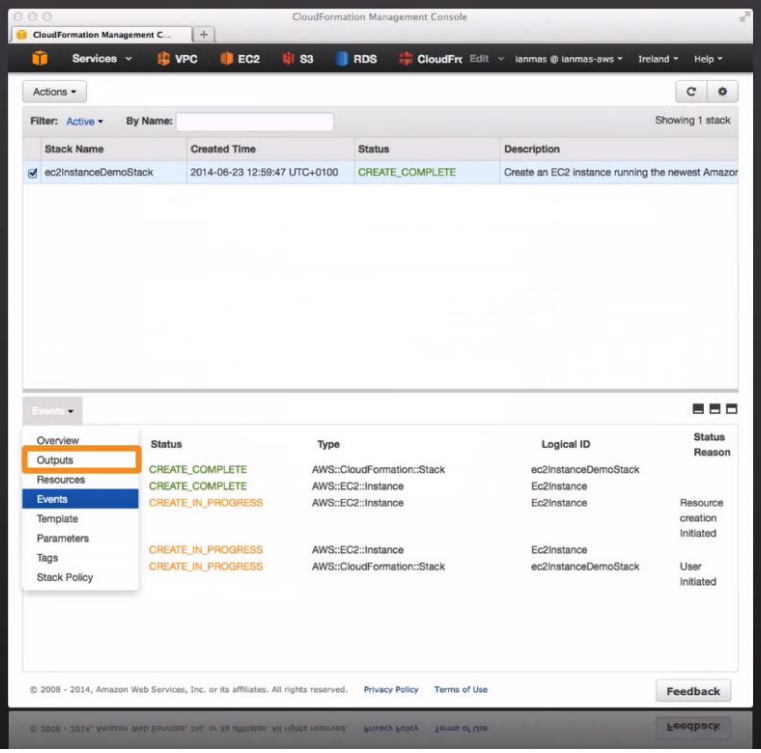


Using a template to create and manage a stack

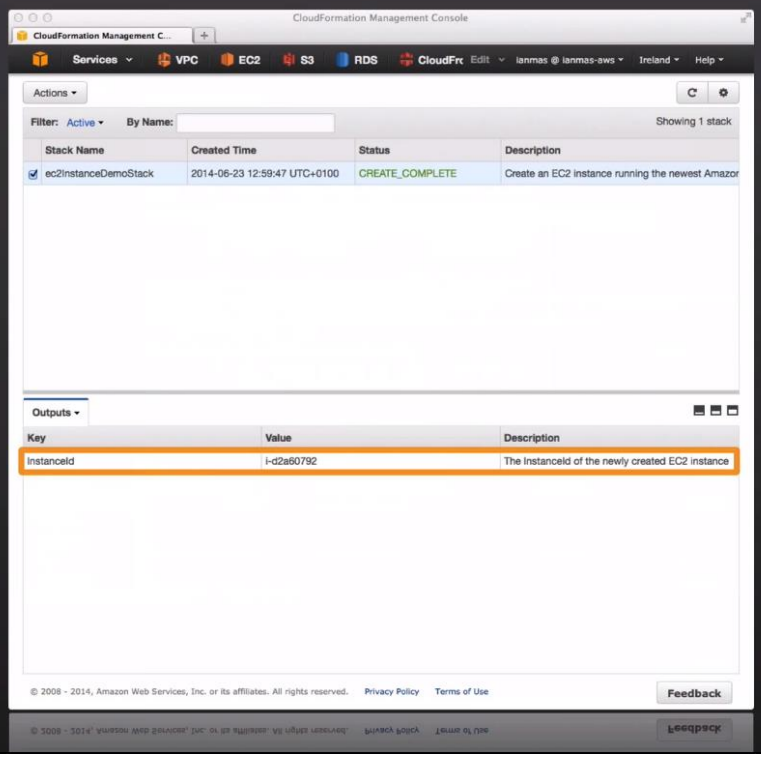


You can now begin to see the event stream for the stack creation process

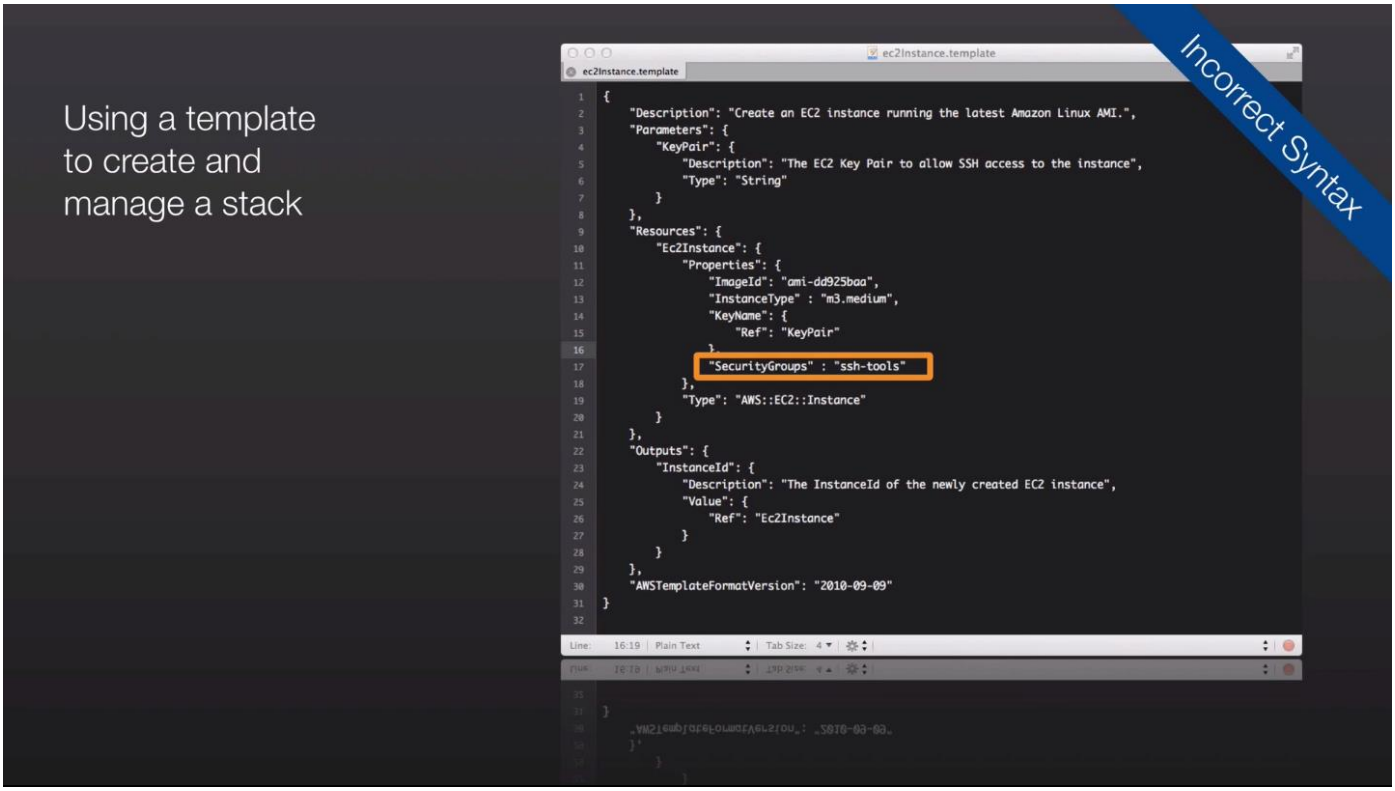
Using a template to create and manage a stack



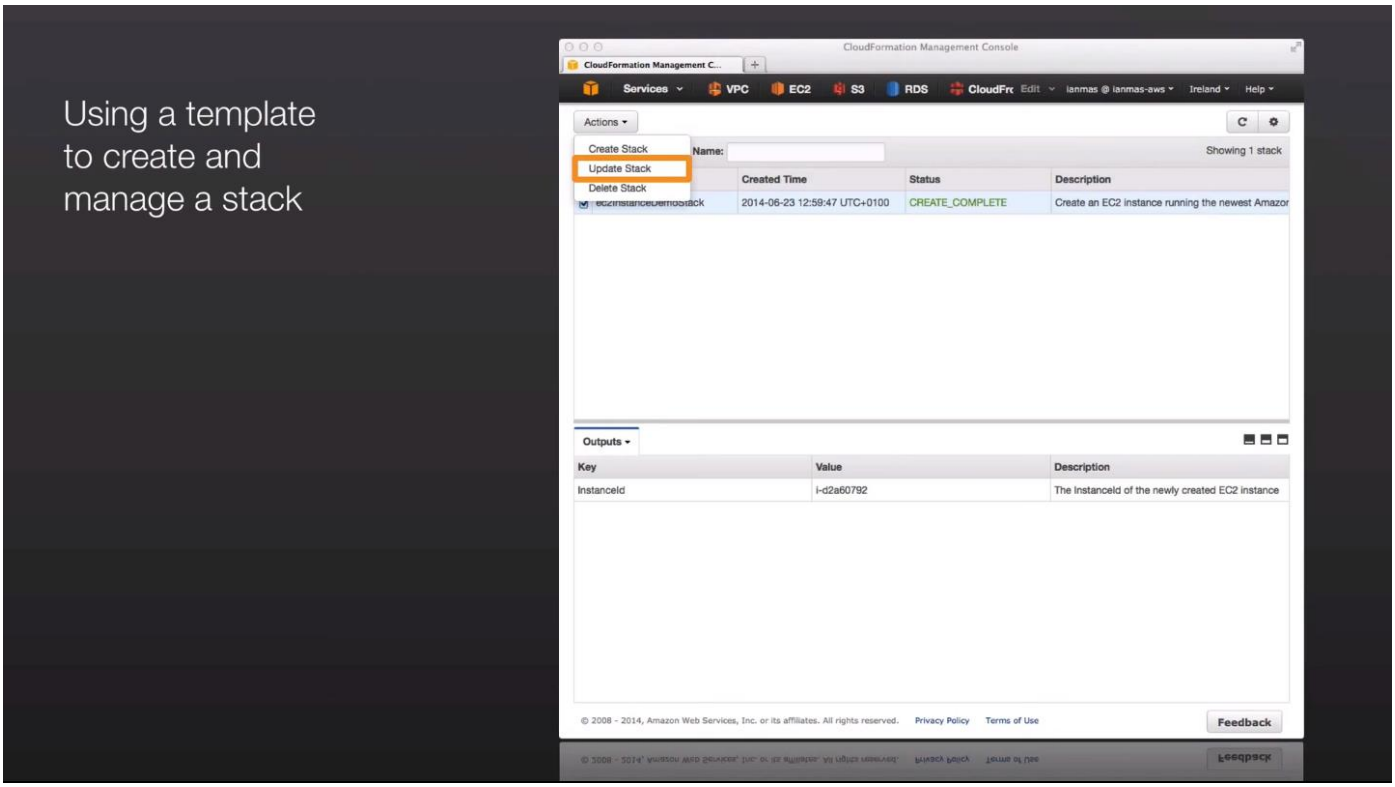
Using a template to create and manage a stack



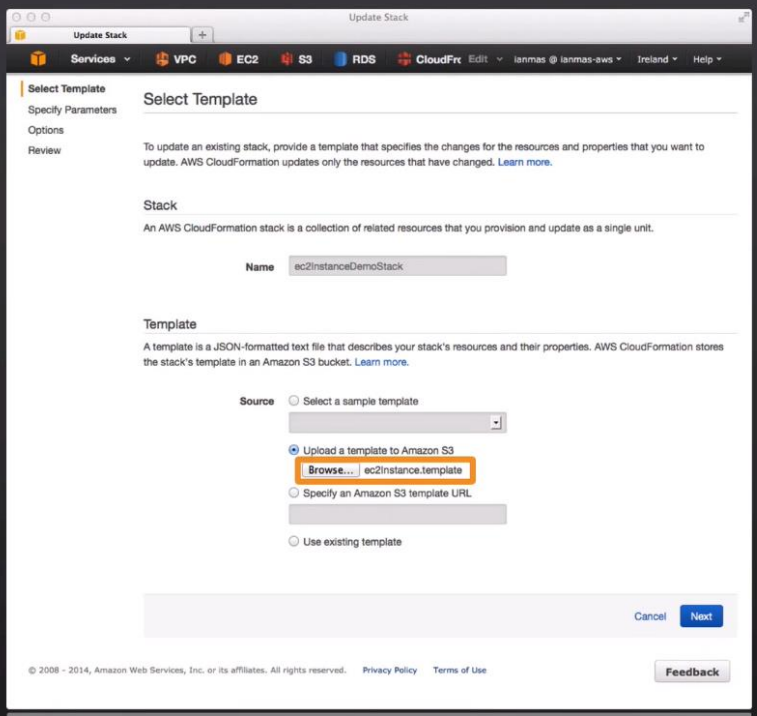
You can now see the outputs that we want from the new stack



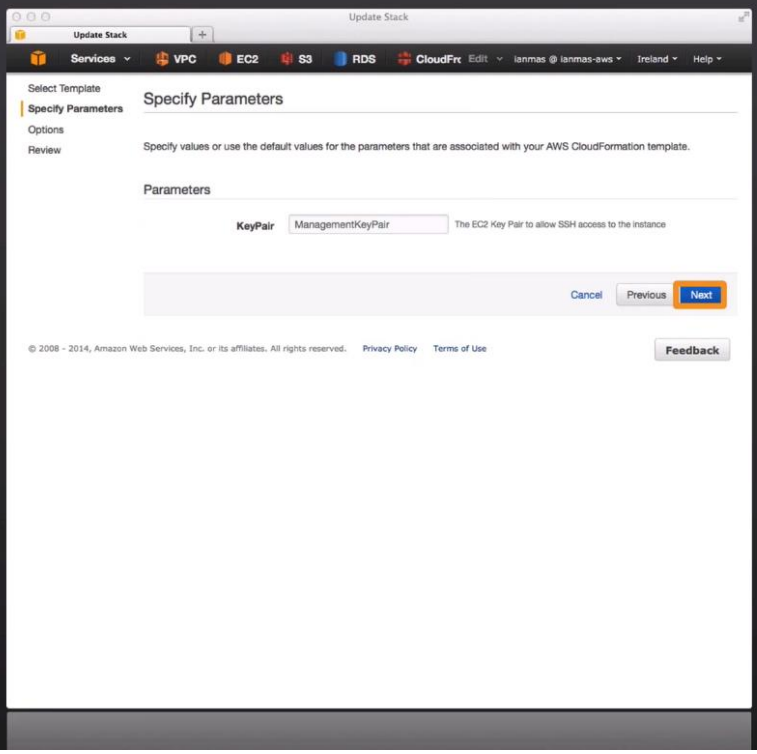
Next, let us see what we need to do when we update our stack definition as above where we have edited the Resources section of the stack template to add a security group access. We have deliberately put in a syntax error in the change above

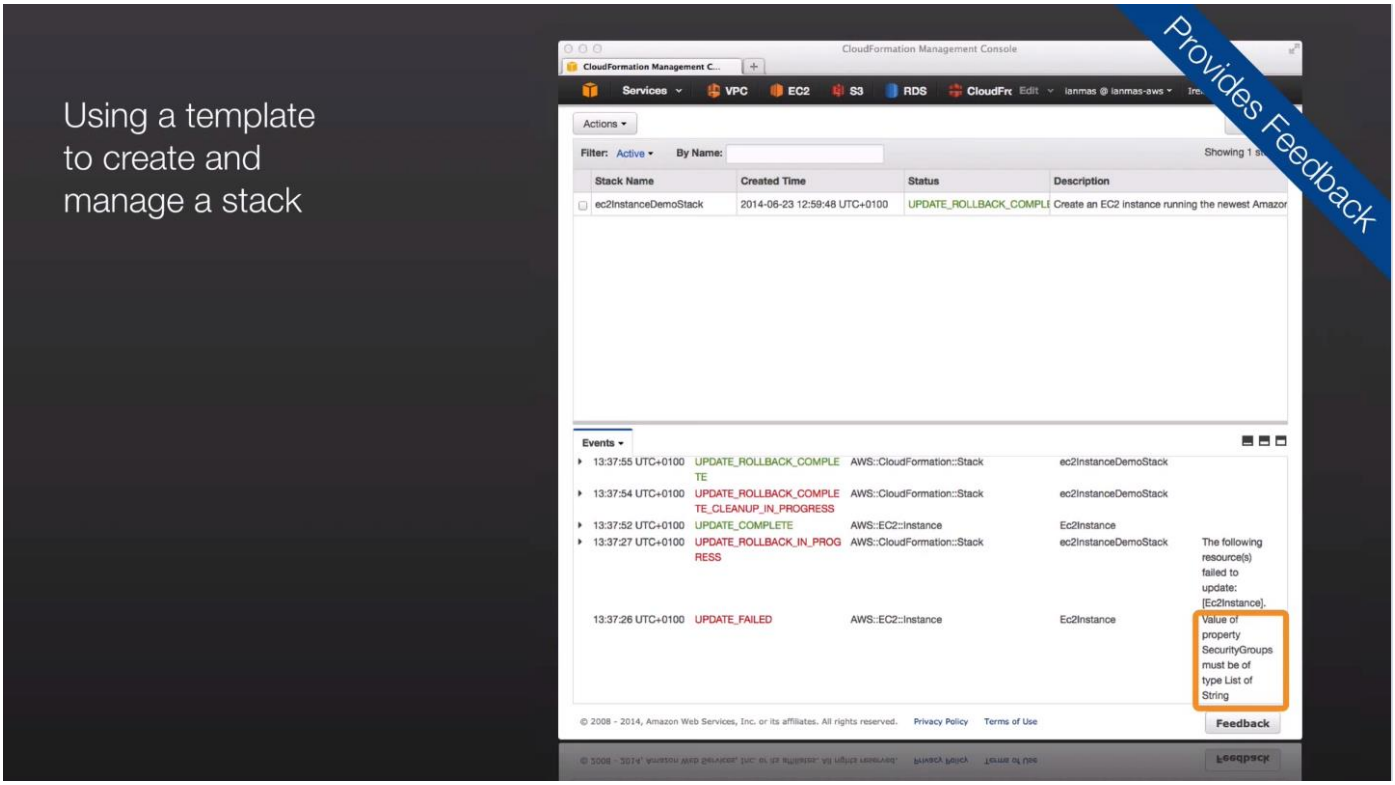


Using a template to create and manage a stack

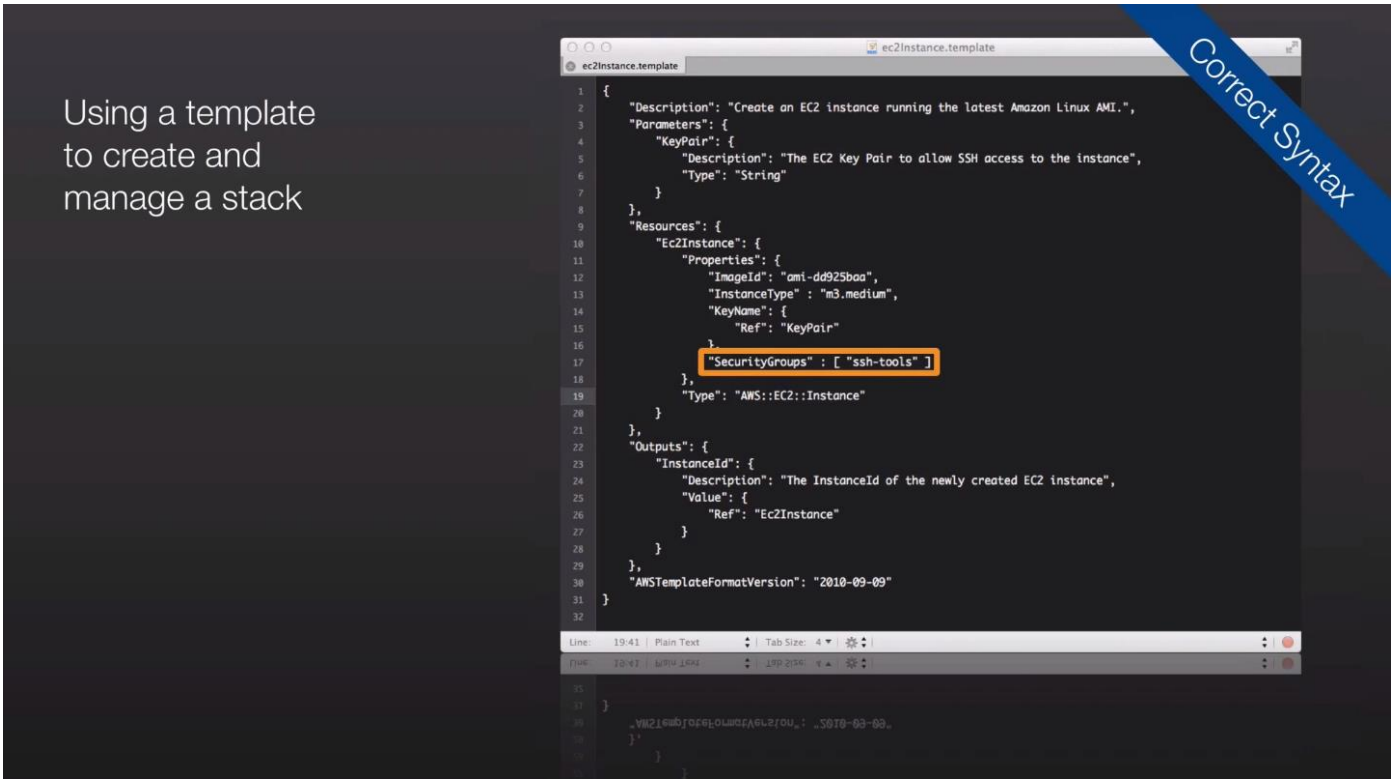


Using a template to create and manage a stack

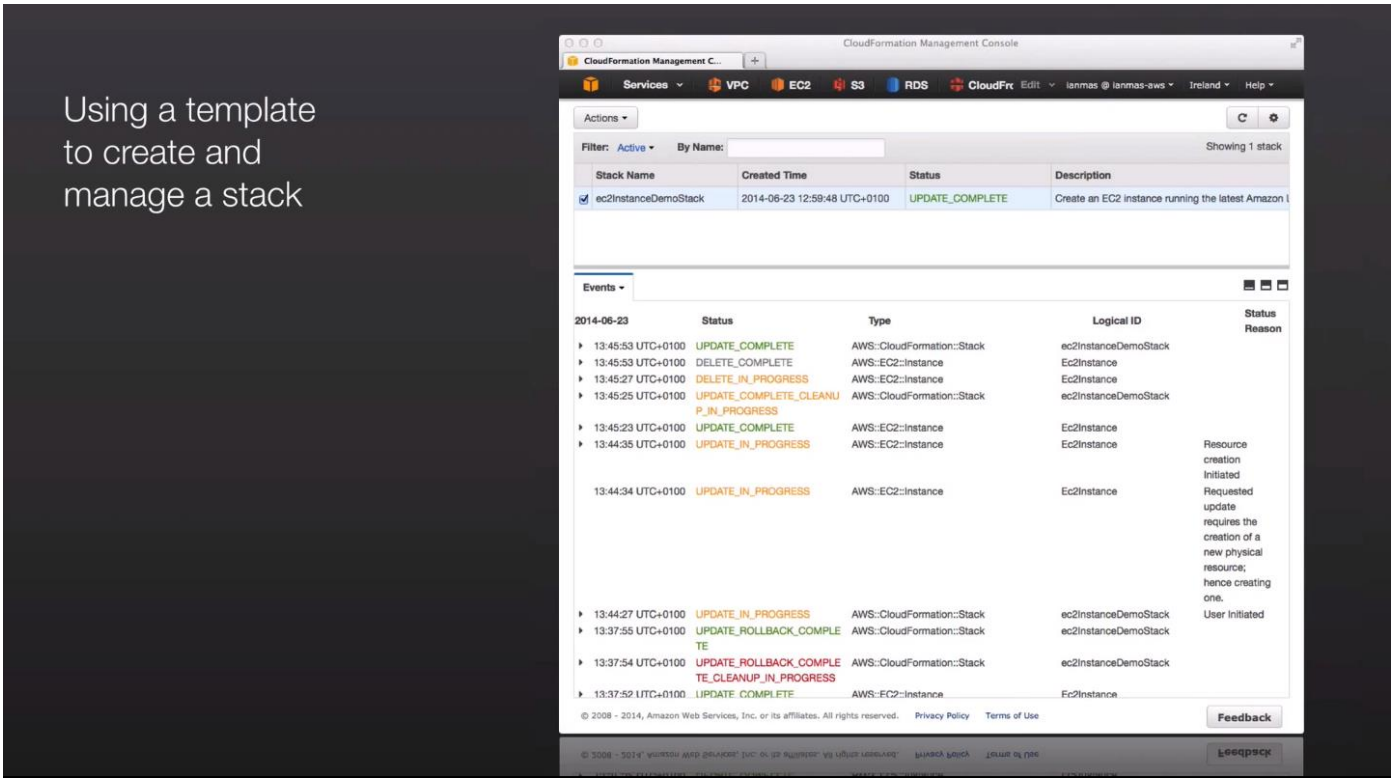




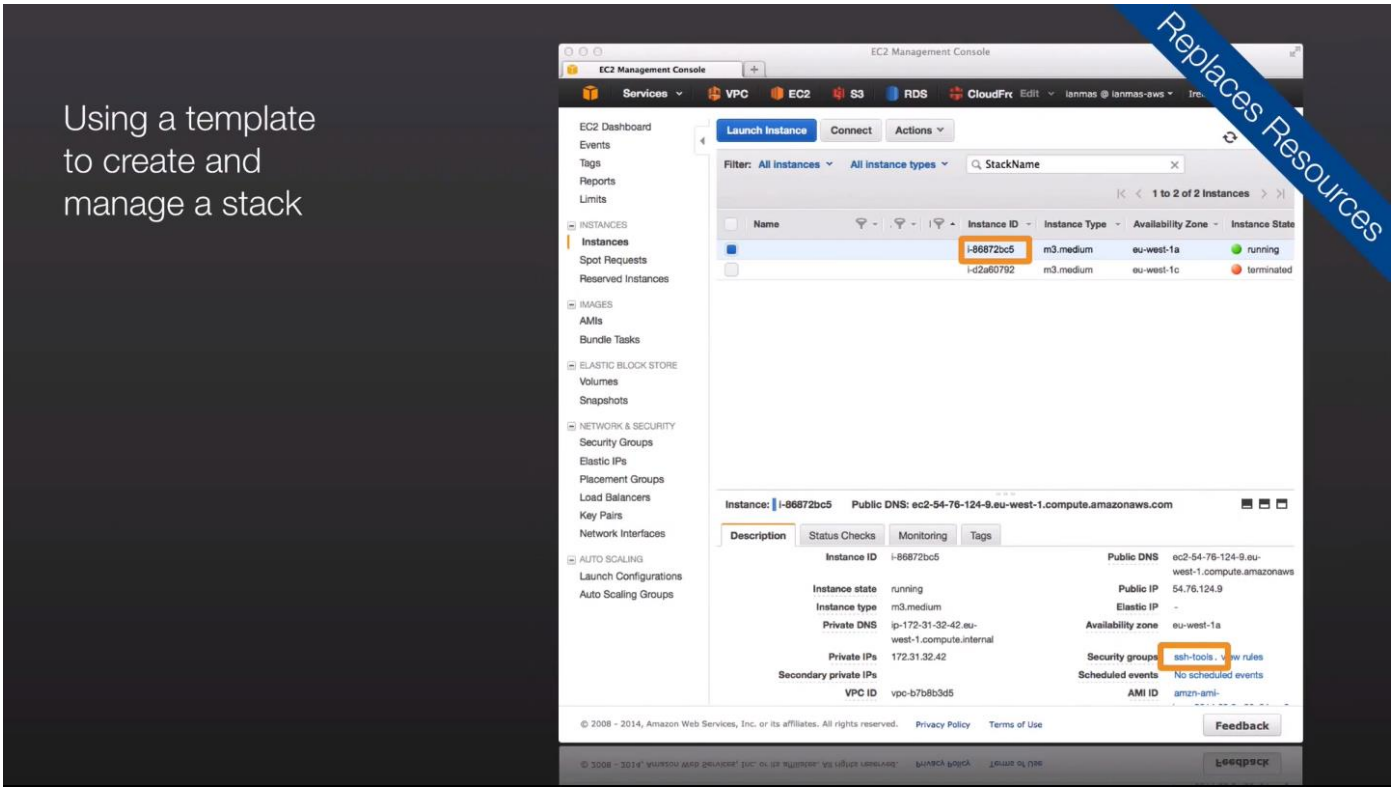
Note that we see details of what happened since our update as failed.



We can then fix the syntax problem

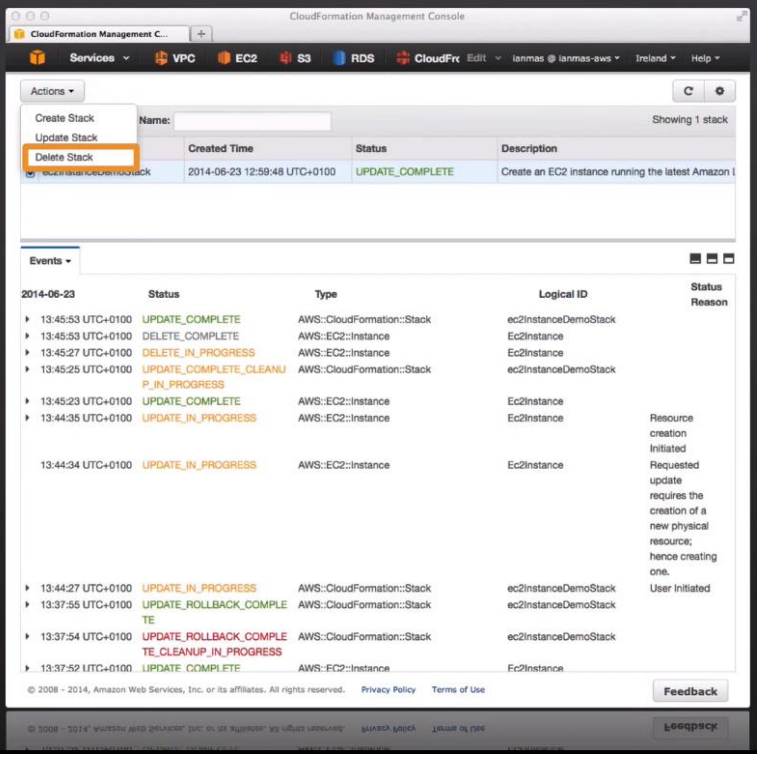


We can now successfully complete the update and recreate the stack, we now have a new EC2 instance with the appropriate security group

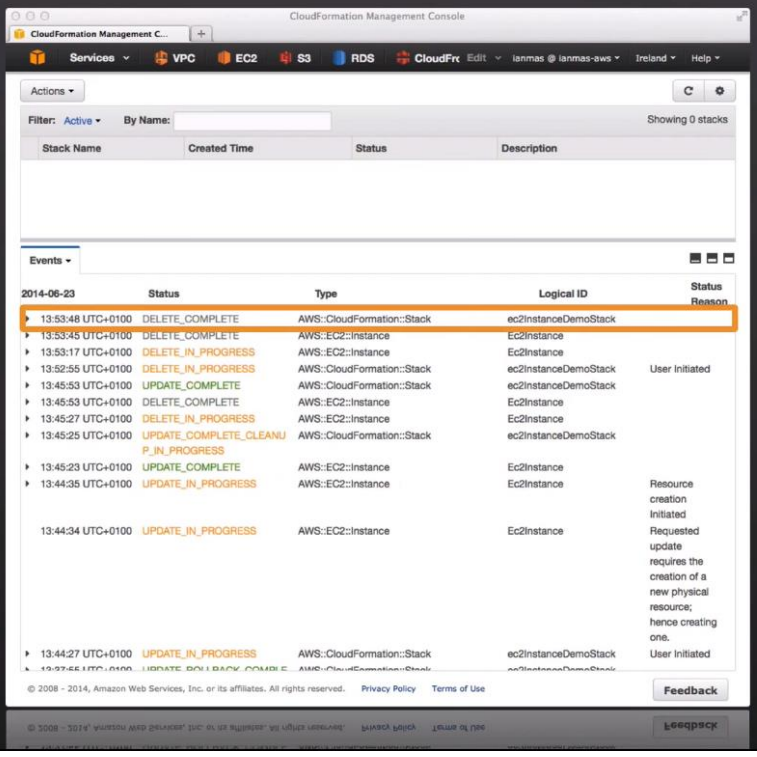


We can see that the old EC2 instance is being terminated and we have a new EC2 instance running from the updateStack operation

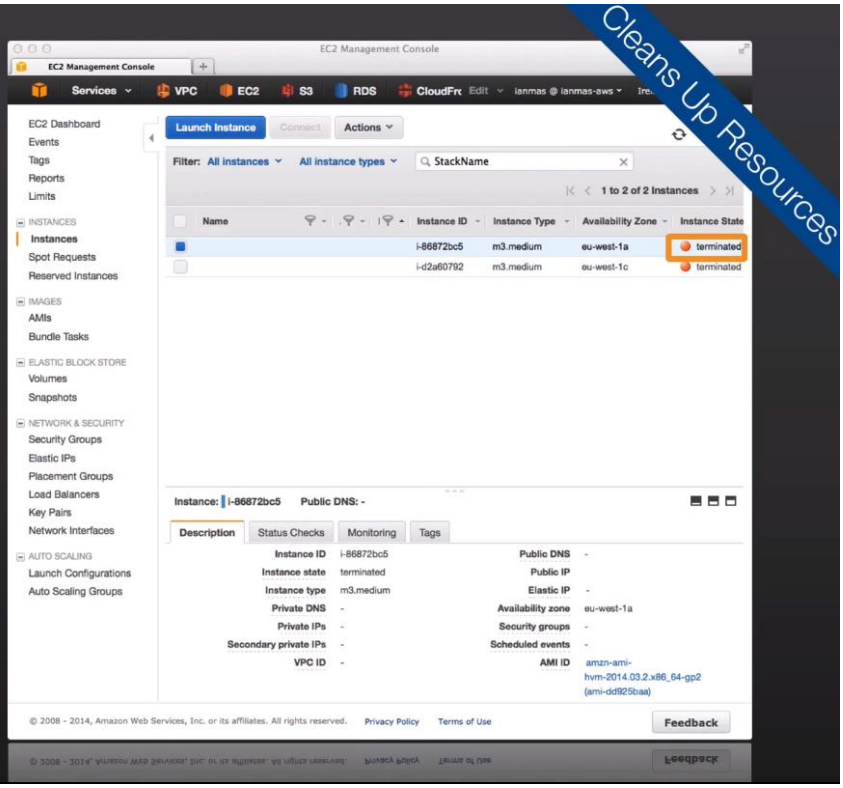
Using a template to create and manage a stack



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 via the AWS CLI

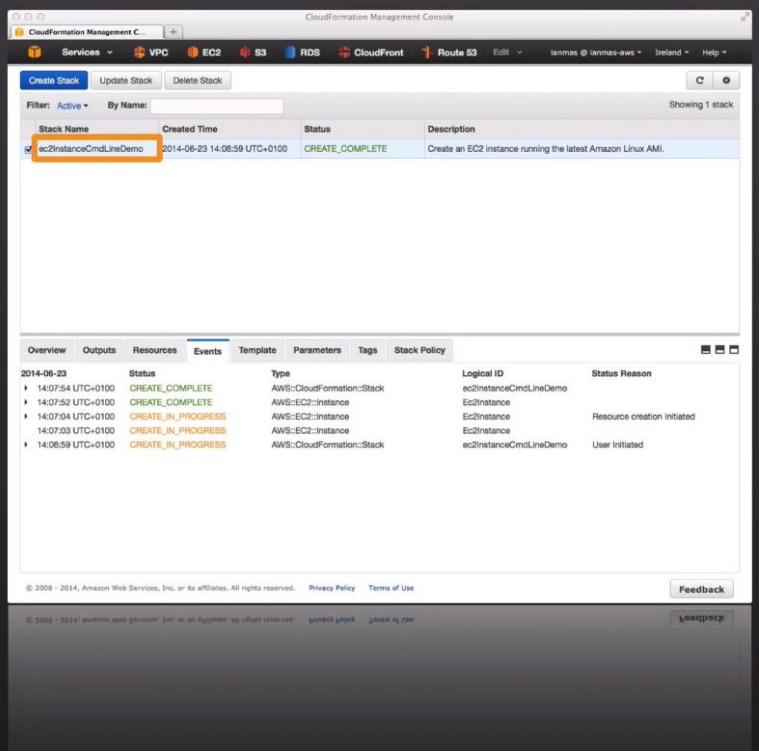
```
aws cloudformation create-stack
    --stack-name ec2InstanceCmdLineDemo
    --template-url https://s3-eu-west-1.amazonaws.com/cf-
templates-1fhelryvrdrbr-eu-west-1/2014174d0r-ec2Instance.template
    --parameters ParameterKey=KeyPair,ParameterValue=ManagementKeyPair
```

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
```

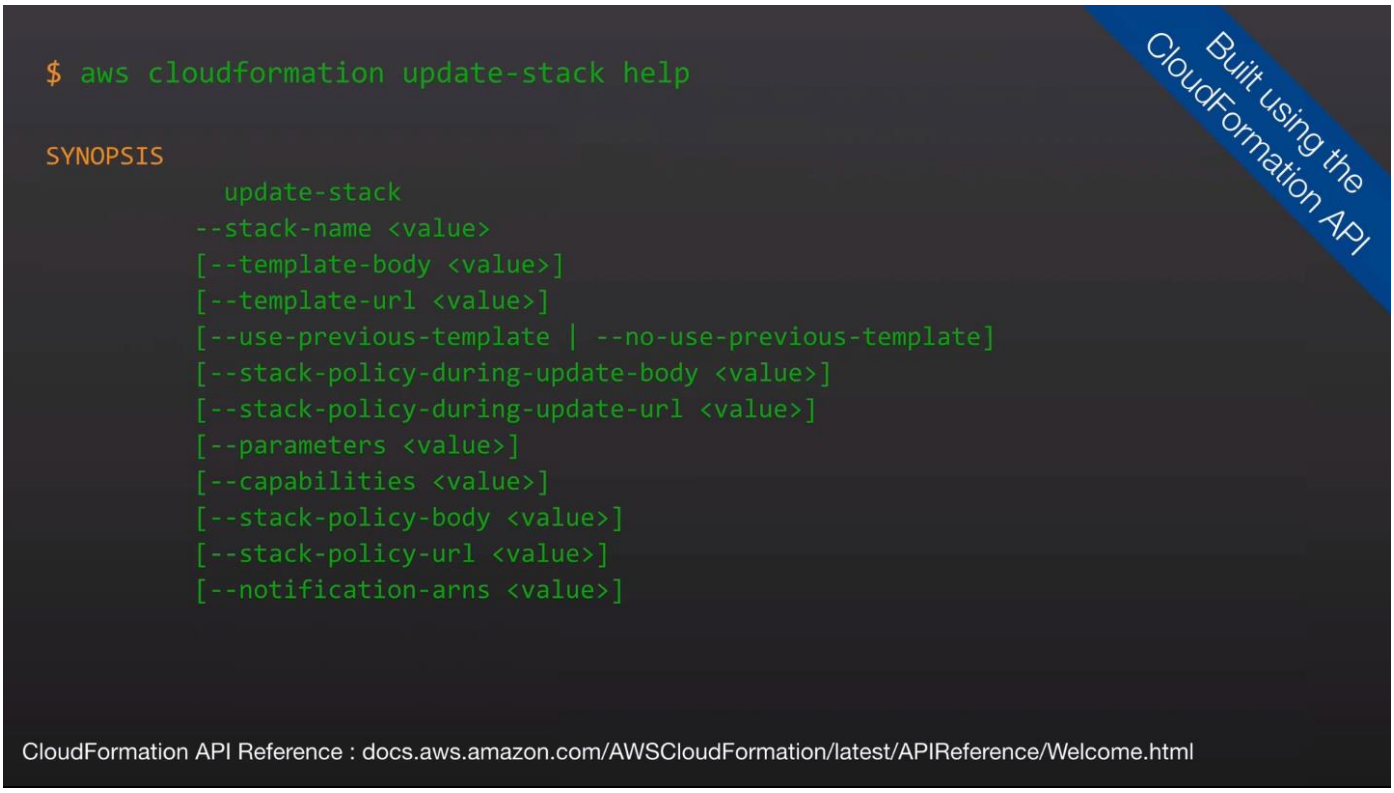
As usual, you can get more details via the AWS CLI

```
$ 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>]
```



That concludes the introduction for creating, updating, and deleting a CF stack from the console and the CLI.

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.

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.

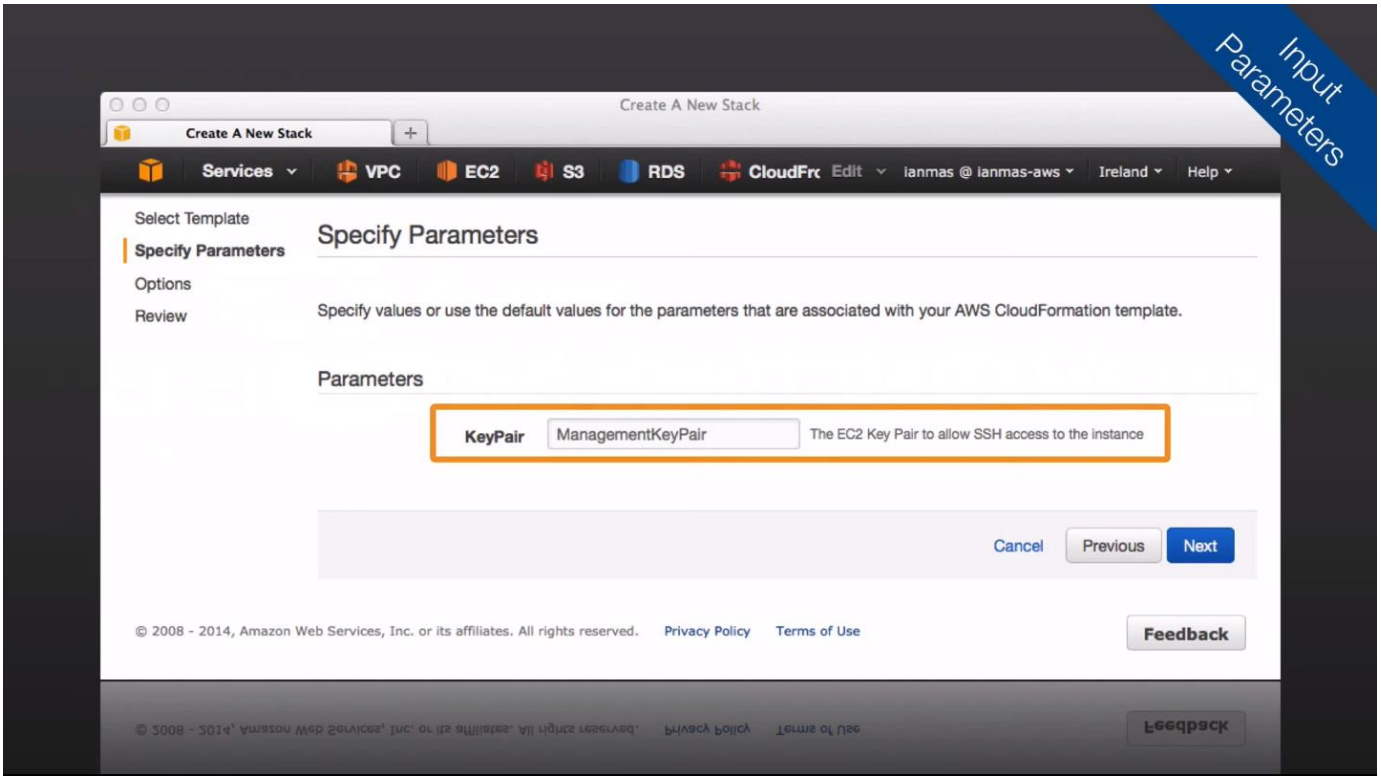
There are also situations where you may need to refer to resources that have been created outside of CF, like if you have pre-existing VPC with security groups in it that you want to make use of or you want to statically find a keypair that you want to use for SSH access. You can simply use literal references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

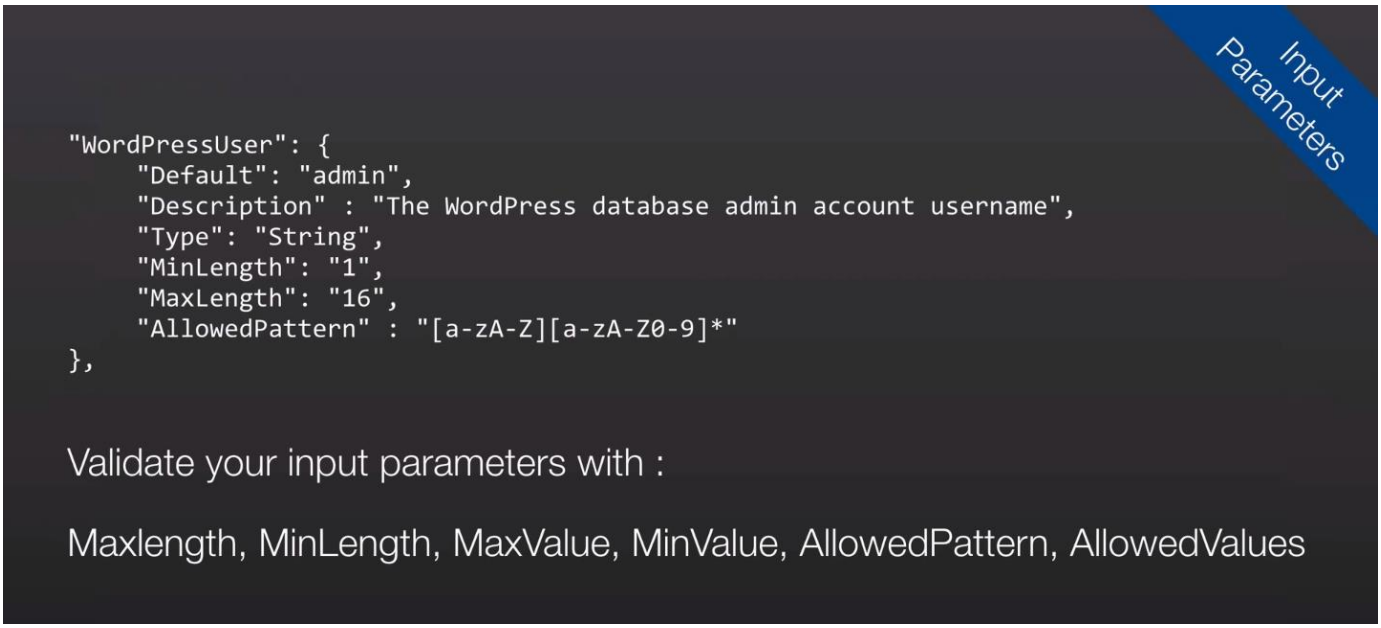
REFERENCING INPUT PARAMETERS

Input Parameters

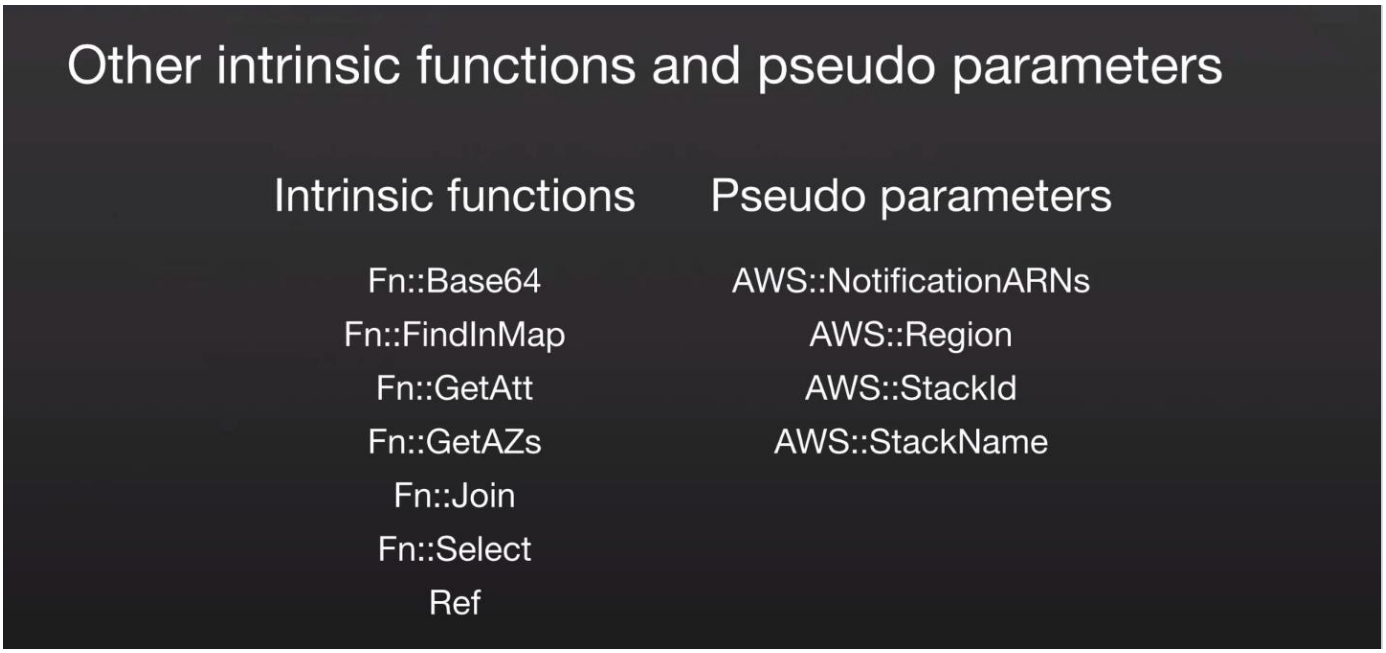
```
{
"Parameters" : {
     "KeyPair" : {
     "Description" : "The EC2 Key Pair to allow SSH access to the instance",
     "Type" : "String"
},
"Resources" : {
          "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
           "Properties" : {
               "SecurityGroups" : [ { "Ref"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



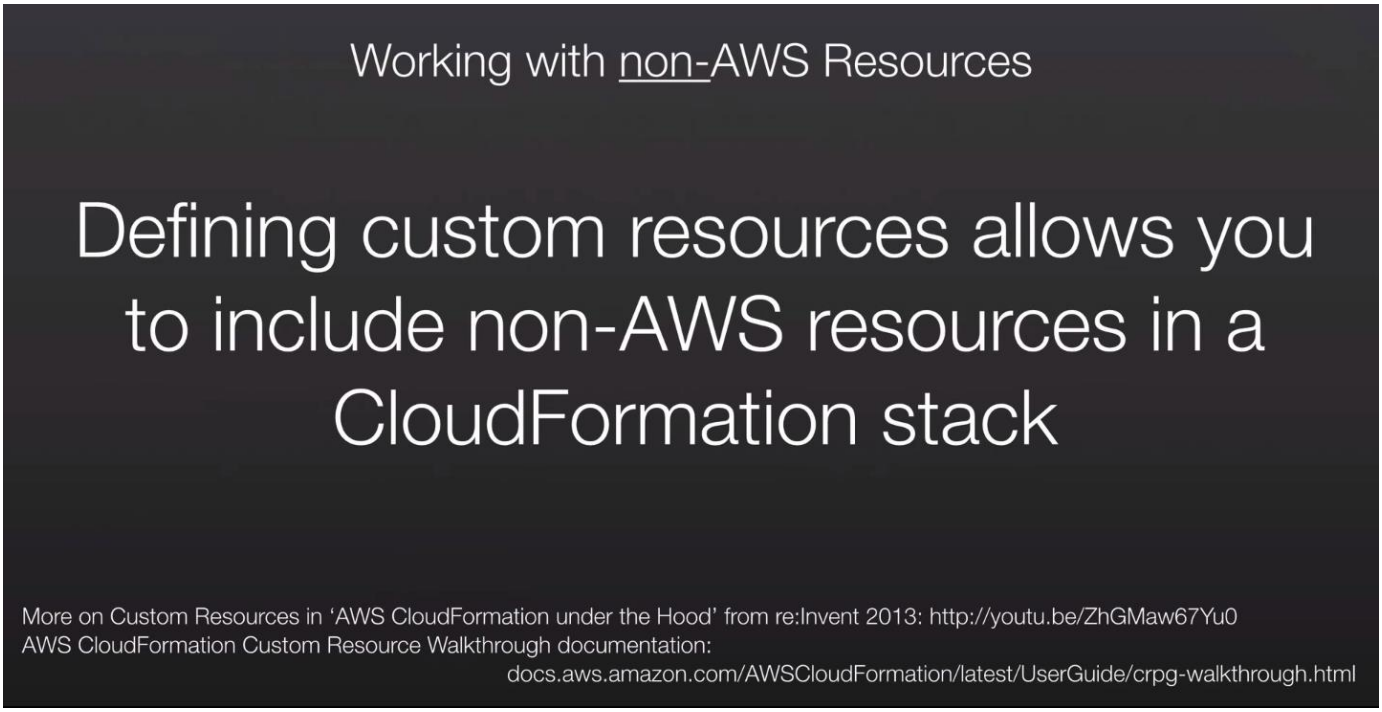


This is important because there might be region-specific parameters that you might require within a template. An example is when we use a CF template to create an EC2 instance in a particular region, then we want to make this template also portable across different regions. We may need different AMI 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***.



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.



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","\n",
            "yum -y install gcc-c++ make","\n",
            "yum -y install mysql-devel sqlite-devel","\n",
            "yum -y install ruby-rdoc rubygems ruby-mysql ruby-devel","\n",
            "gem install --no-ri --no-rdoc rails","\n",
            "gem install --no-ri --no-rdoc mysql","\n",
            "gem install --no-ri --no-rdoc sqlite3","\n",
            "rails new myapp","\n",
            "cd myapp","\n",
            "rails server -d","\n",
            "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"
              ]]}}
     }
   },
```

Creating files on Instance Filesystems

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"
              }
            }
```

Controlling Services

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!

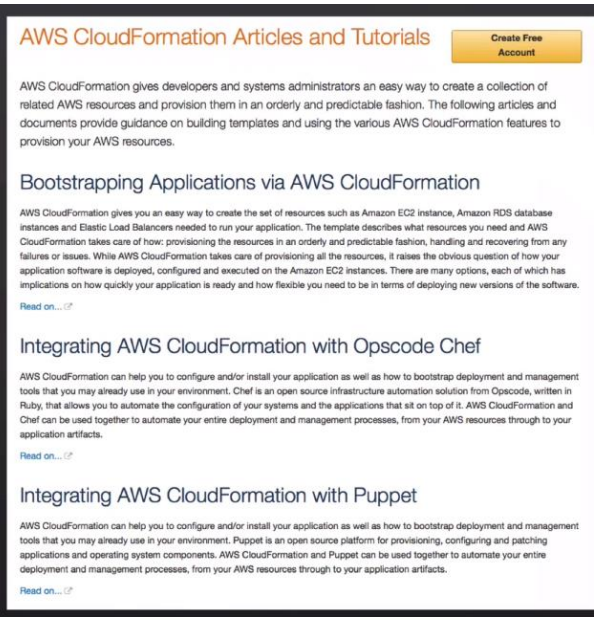
All this functionality is available for Windows instances too!

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:

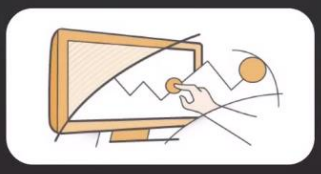
github.com/awslabs/cfncluster/

CloudFormation Template to Deploy Wordpress

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

AWS Training & Certification

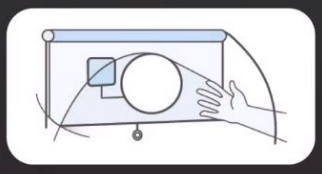
Self-Paced Labs



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

aws.amazon.com/training/self-paced-labs

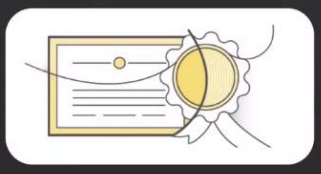
Training



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

aws.amazon.com/training

Certification



Validate your proven skills and expertise with the AWS platform

aws.amazon.com/certification