Learn CloudFormation

Write, deploy, and maintain your AWS infrastructure

Packt

www.packt.com

By Agus Kurniawan

Learn CloudFormation

Write, deploy, and maintain your AWS infrastructure

Agus Kurniawan

BIRMINGHAM - MUMBAI

Learn CloudFormation

Commissioning Editor: Gebin George
Acquisition Editor: Heramb Bhavsar
Content Development Editor: Abhishek Jadhav
Technical Editor: Swathy Mohan
Copy Editor: Safis Editing
Project Coordinator: Kinjal Bari
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Graphics: Tom Scaria
Production Coordinator: Deepika Naik

First published: July 2018

Production reference: 1310718

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78913-432-2

www.packtpub.com

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Agus Kurniawan is a lecturer, researcher, IT consultant, and author. He has more than 16 years of experience in various software and hardware development projects for various companies. He also has been delivering materials in training and workshops, and delivering technical writing. He has been awarded the Microsoft Most Valuable Professional (MVP) award for 13 years in a row. He is currently doing some research related to software engineering, machine learning, networking, and security systems at the Faculty of Computer Science, University of Indonesia.

I would like to thank the Amazon AWS and IoT communities everywhere in the world for contributing and making learning AWS IoT easy. Last, a thank you to my wife, Ela, and my children, Thariq and Zahra, for their great support while I was completing this book.

About the reviewer

Gajanan Chandgadkar has more than 12 years of IT experience. He has spent more than 6 years in the USA, helping large enterprises architect, migrate, and deploy applications in AWS. He is an AWS-certified solutions architect professional and a certified DevOps professional with more than seven certifications in trending technologies. He is also a technology enthusiast who has vast experience in different topics, such as application development, container technology, and continuous delivery.

Currently, he is working with Happiest Minds Technologies as an associate DevOps architect.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

As the Amazon Web Services (AWS) infrastructure is gradually moving towards the cloud, managing cloud-related tasks efficiently continues to be a challenge for system administrators. CloudFormation is a language that was developed for managing infrastructure-related services efficiently on AWS, and its features help secure the AWS resource deployment process. *Learn CloudFormation* serves as a fundamental guide that will kick-start your journey with CloudFormation. We will introduce you to the basic concepts of infrastructure as code (IaC) and the AWS services required to implement automation and infrastructure management. Then, we deep dive into concepts such as CloudFormation mapping, conditions, limits, output, and EC2. In the concluding chapters, you will manage the entire AWS infrastructure using CloudFormation templates.

By the end of this book, you will be up and running with IaC with CloudFormation.

Who this book is for

Learn CloudFormation is for cloud engineers, system administrators, cloud architects, or any stakeholders working in the field of cloud development or cloud administration. Basic knowledge of AWS is necessary.

What this book covers

`Chapter 1`, *Introducing AWS CloudFormation*, will introduce AWS CloudFormation and explain how to get started with AWS CloudFormation.

`Chapter 2`, *Building Your First AWS CloudFormation Project*, will explain how to get started building your first AWS CloudFormation project. The project will be implemented with a step-by-step practical approach.

`Chapter 3`, *Developing AWS CloudFormation Templates*, will explore how to develop AWS CloudFormation templates. Readers will learn how to create AWS CloudFormation templates using JSON and YAML. Readers also will be introduced to using AWS CloudFormation Designer to develop AWS CloudFormation templates in GUI mode.

`Chapter 4`, *AWS CloudFormation StackSets*, will explore the basics of AWS CloudFormation StackSets. By the end of this chapter, readers will know how to build and manage AWS CloudFormation StackSets.

`Chapter 5`, *Building Lambda Functions Using AWS CloudFormation*, will take readers through the process of deploying Lambda Functions using AWS CloudFormation.

`Chapter 6`, *AWS CloudFormation Security*, will deal with securing resources that are deployed using AWS CloudFormation.

Chapter 7, *Managing and Testing Production Infrastructure for AWS CloudFormation*, will explain how to manage and deploy testing and production infrastructure using AWS CloudFormation. The chapter is not present in the book, but it is available for the download in the following link: `https://www.packtpub.com/sites/default/files/downloads/Managing_and_Testing_Production_Infrastructure_for_AWS_CloudFormation.pdf`.

To get the most out of this book

As the practical examples involve the use of AWS, an AWS account is required.

Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Learn-CloudFormation`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/LearnCloudFormation_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "We can use the `Parameters` attribute from the CloudFormation template."

A block of code is set as follows:

```
exports.handler = (event, context, callback) => {
    var data = event['msg'];
    callback(null, 'Received: ' + data);
};
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
{
 "Type" : "AWS::S3::Bucket",
 "Properties" : {
 "AccessControl" : String,
 "AccelerateConfiguration" : AccelerateConfiguration,
 "AnalyticsConfigurations" : [ AnalyticsConfiguration, ... ],
 "BucketEncryption" : BucketEncryption,
 "BucketName" : String,
}
```

Any command-line input or output is written as follows:

```
$ aws cloudformation create-stack --stack-name my-simple-stack
 --template-body file://home/user/templates/simple-s3.json
 --parameters YourBucketName=my-simple-s3
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "On the management console, click on the **Roles** in the left menu."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

1
Introducing AWS CloudFormation

Cloud technology enables IT staff to build virtual data centers easily enabling them to set up a virtual machine with specific configurations. Now, **Infrastructure as Code** (**IaC**) enables building a virtual data center by scripting. In this chapter, we'll briefly look at IaC and some terms in AWS CloudFormation.

The following topics will be covered in this chapter:

- Introducing IaC
- AWS CloudFormation
- CloudFormation templates
- CloudFormation stacks
- CloudFormation StackSets
- Exploring the AWS CloudFormation console

Introducing IaC

Cloud technology enables you to manage servers virtually. We can set memory, CPU, and storage in the virtual model. To build a data center based on the cloud platform, you set up your resources, such as the virtual machine, virtual network, and applications. Sometimes, you need to reconfigure your data center design or rebuild it in another location. You probably perform the same actions to build your data center. These tasks probably make you unhappy.

IaC can be described as a programmable infrastructure that enables you to manage resource configurations using scripts and to automate the provisioning of the infrastructure in addition to deployments.

IaC is designed to reduce the complexity that kills efficiency in manual configuration. You can deploy your infrastructure code to development, testing, staging, and production environments. The following diagram shows the processes in IaC:

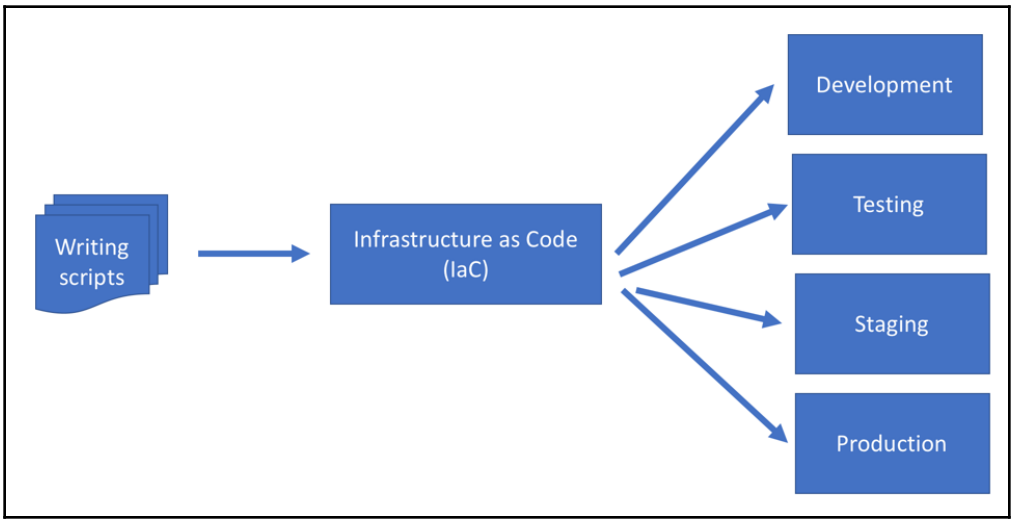


Figure 1.1: Processes in IaC

Imagine you have deployed some resources on several machines, and then some machines crash due to a disaster such as a power failure. Technically, you should deploy all resources manually. IaC is designed to help you deploy the infrastructure automatically. IaC can minimize your risks while deploying a modern infrastructure.

We can deploy our infrastructure on several environments with a single IaC script template. This approach enables us to minimize problems on deployment. We don't need to worry if we want to redeploy our infrastructure, since we have our infrastructure scripts.

AWS CloudFormation

Amazon AWS is a cloud-based platform that provides cloud services. There are a lot of cloud services, so we can deploy any server and any program for general or specific purposes. AWS CloudFormation is designed to implement IaC. You can write scripts to build a custom infrastructure. Once you have done so, you can you can deploy a template which represents Infrastructure as Code for development, testing, staging, and production environments.

AWS CloudFormation is a solution for dynamic infrastructures. You can write your own infrastructure without worrying about reconfiguring and redeploying. AWS CloudFormation allows you to model your entire infrastructure in a text file. This approach can standardize your infrastructure resources across your organization and speed up the troubleshooting process.

AWS CloudFormation helps to build and rebuild your infrastructure and applications, without having to perform manual actions or write custom scripts.

For further information about AWS CloudFormation, you can visit its official website at `https://aws.amazon.com/cloudformation/`.

You will find a lot of information, such as features, pricing, and documentation, as seen in the following screenshot:

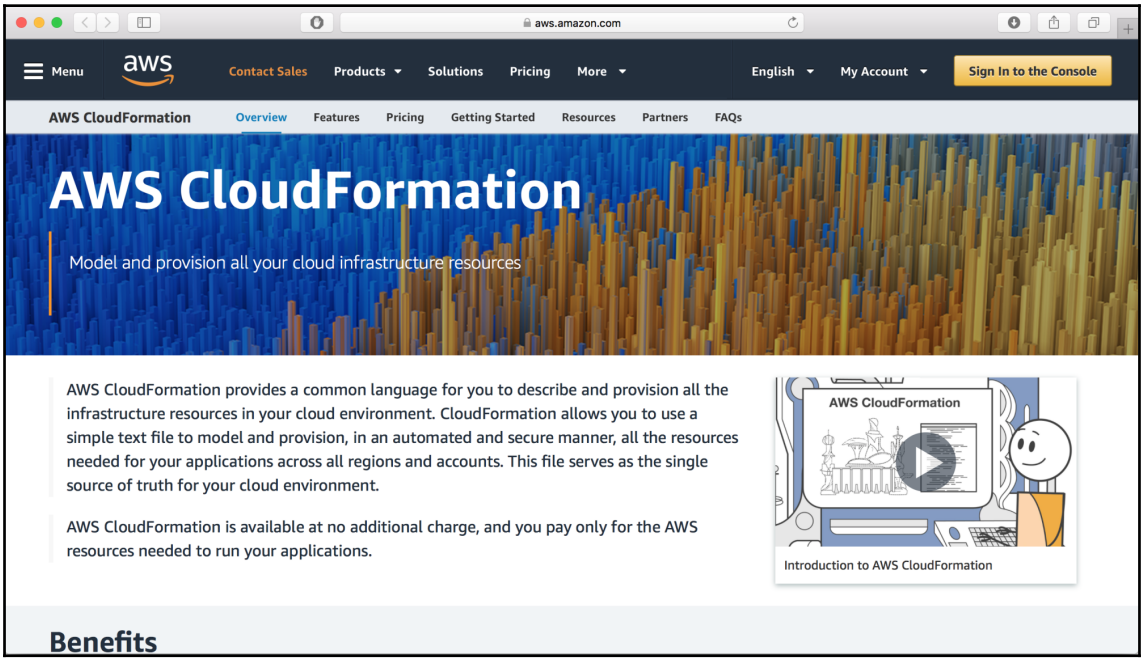


Figure 1.2: The official AWS CloudFormation website

Currently, AWS CloudFormation is free, but the resources that you use in IaC scripts usually aren't. You can read the details about pricing at `https://aws.amazon.com/cloudformation/pricing/`.

How does it work?

AWS CloudFormation provides solutions to build infrastructure from scripts. You can set several resources for infrastructure design. In general, see the following flowchart for how to build IaC:

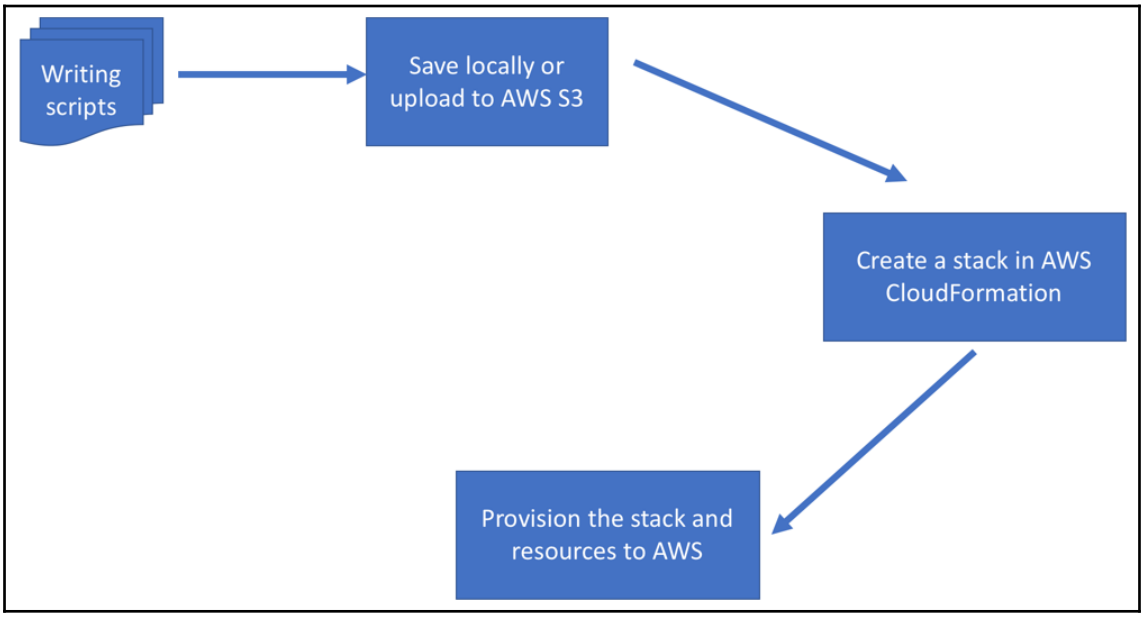


Figure 1.3: A flow for building AWS CloudFormation

Start by writing scripts to build infrastructure using AWS CloudFormation; you should write IaC scripts in JSON, YAML, or text. After completing the scripts, you can upload that script file to AWS CloudFormation. You can also put it on AWS S3 first, so AWS CloudFormation will download it directly.

AWS CloudFormation provisions a stack and then creates AWS resources based on your scripts. You also can set the target region for your CloudFormation stack.

You can deploy and replicate AWS resources on some regions with a single CloudFormation template. We can build AWS StackSets if we want to work with AWS resources in different regions in a single CloudFormation template.

Controlling IaC source scripts

Since we write IaC in JSON, YAML, and text formats, we can save these script files and they can be stored to source control servers. You can manage script versions for editing based on your needs.

You can use Git to manage IaC scripts. Once the scripts are released, you can push them to AWS CloudFormation to provision your AWS resources. The following diagram shows IaC scripting with source control:

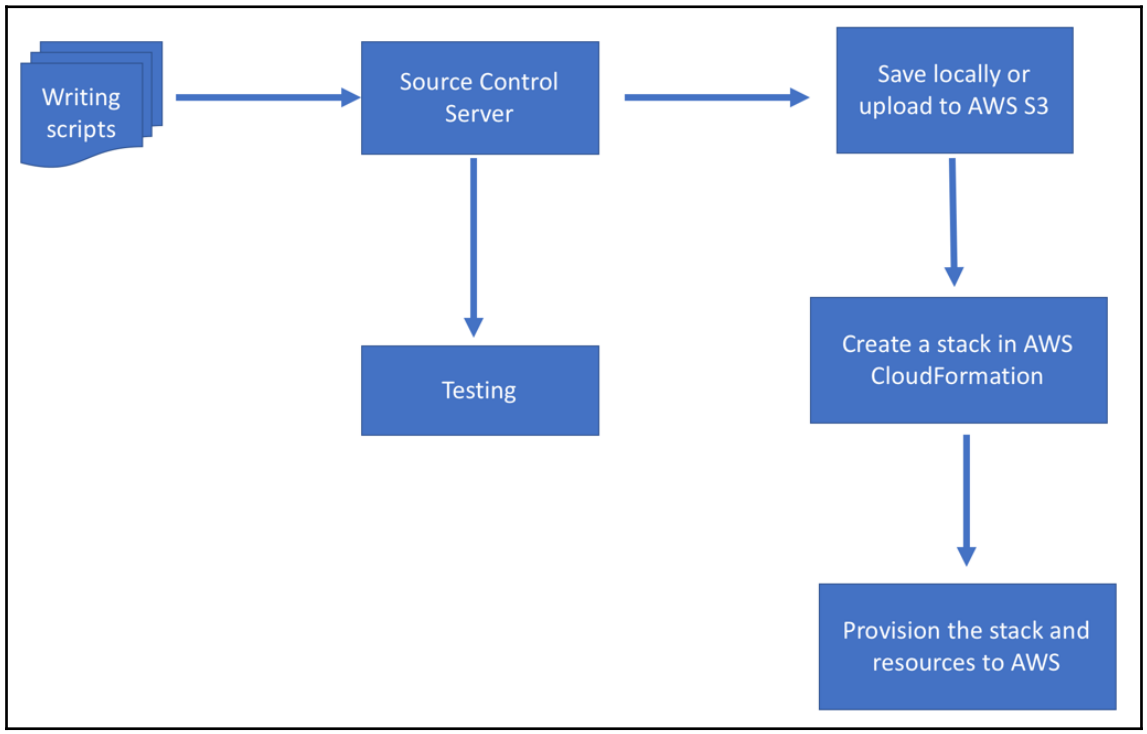


Figure 1.4: IaC scripting with source-control

The preceding diagram describes how to combine source-control with AWS CloudFormation. We write IaC scripts on the local computer. Once done, check the script into the source control server. If you have a testing team, your team can test the script with AWS CloudFormation.

CloudFormation templates

We can develop IaC in AWS CloudFormation easily. AWS already provides a list of templates that we can use for our IaC design. You can write AWS CloudFormation scripts in JSON, YAML, and text.

For instance, we use AWS S3 on AWS CloudFormation. We can write AWS CloudFormation scripts in JSON as follows:

```
{
 "Resources" : {
   "HelloBucket" : {
    "Type" : "AWS::S3::Bucket",
    "Properties" : {
        "AccessControl" : "PublicRead"
    }
   }
 }
}
```

We also can write these IaC scripts in YAML:

```
Resources:
 HelloBucket:
    Type: AWS::S3::Bucket
    Properties:
      AccessControl: PublicRead
```

You can use either JSON or YAML to write IaC scripts and you should learn the AWS CloudFormation API's included parameter when we use those resources.

Sometimes, we prefer to build AWS CloudFormation visually, when we don't know much about JSON or YAML. Fortunately, AWS provides the AWS CloudFormation template designer. We can click and drag any resource to the editor. You can see this in the following screenshot:

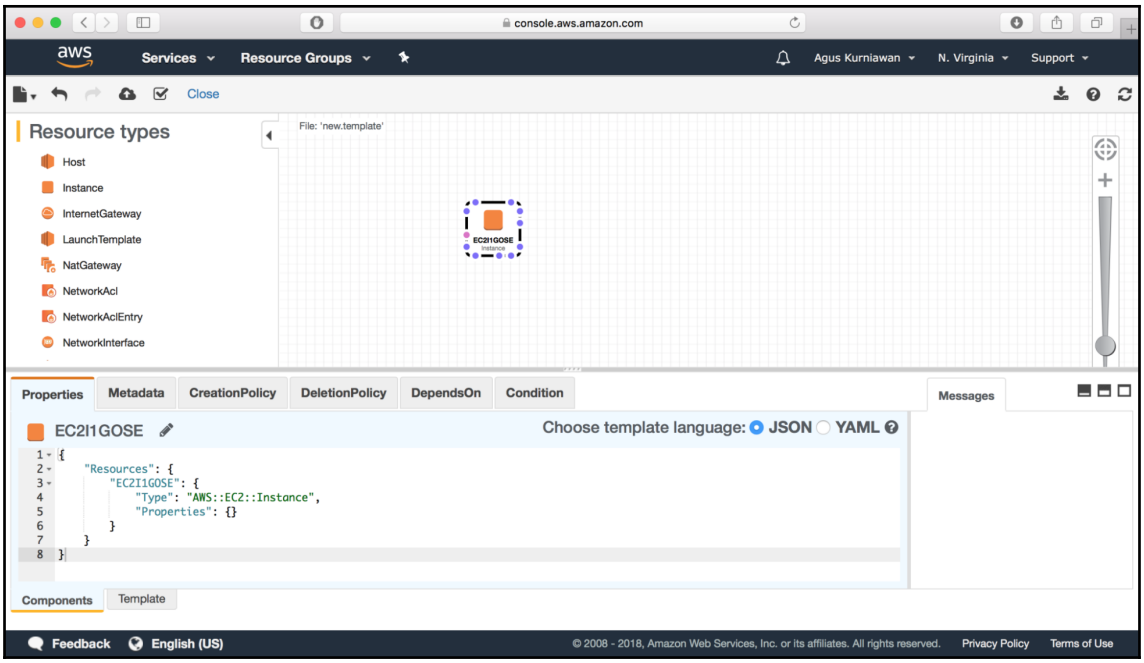


Figure 1.5: Template designer in AWS CloudFormation

If you want to work in Terminal mode, AWS provides the AWS CLI, which enables you to build AWS CloudFormation templates using the Terminal.

AWS also provides AWS CloudFormation templates, so you don't need to spend extra efforts in developing CloudFormation templates. You can download these template samples at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html`.

We will learn more about the IaC scripting included in the AWS CloudFormation templates in `Chapter 3`, *Developing AWS CloudFormation Templates*. We'll use JSON and YAML scripts to develop IaC. We'll also learn how to use the AWS CloudFormation template designer.

CloudFormation stacks

All the resources that you want to build in the AWS environment should be defined in a CloudFormation template. Then, we put this template into a CloudFormation stack. A stack is a collection of AWS resources that you can manage as a single unit. You can create, update, or delete a collection of resources by creating, updating, or deleting stacks.

When you develop an AWS CloudFormation stack, ensure all resources with CloudFormation templates meet with AWS regions. Not all regions have the same resources. If not, you will have problems when provisioning resources.

Sometimes, we use various resources in our infrastructure design, such as database clustering. In this scenario, we can develop CloudFormation nested stacks. This means our CloudFormation stack consists of several stacks. For instance, we draw a design like the one in the following diagram. The CloudFormation stack consists of two stacks: **B1** and **B2**. Inside the **B1 Stack**, we use three CloudFormation stacks: **C1**, **C2**, and **C3**. Then, we build two stacks within the **C3 Stack**, as shown in the following example model of AWS CloudFormation nested stacks:

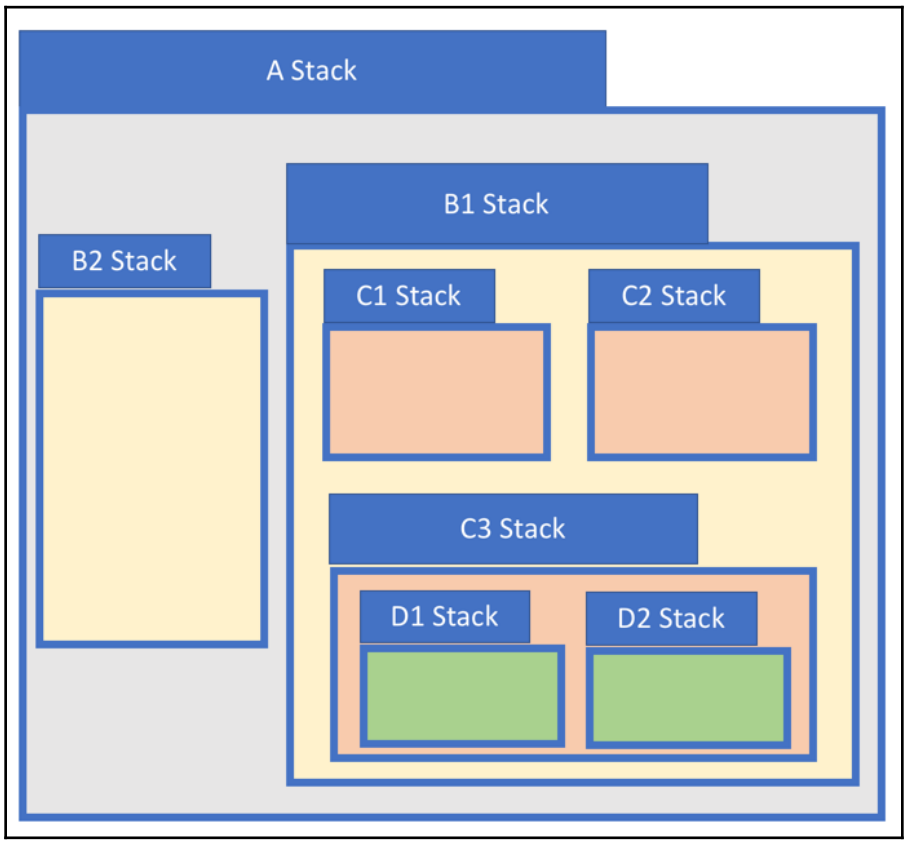


Figure 1.6: An example model of AWS CloudFormation nested stacks

To understand AWS CloudFormation stacks, we will explore this topic in `Chapter 2`, *Building Your First AWS CloudFormation Project* and `Chapter 3`, *Developing AWS CloudFormation Templates*. We'll review some scenarios to show you how to work with AWS CloudFormation stacks.

CloudFormation StackSets

If you plan to deploy AWS resources in multiple regions, you can use a CloudFormation StackSet. Technically, a CloudFormation StackSet is designed to enable you to create CloudFormation stacks in AWS accounts across regions by using a single AWS CloudFormation template.

We illustrate the processing model in a CloudFormation StackSet in the following diagram. In this scenario, if a StackSet is to be executed, this generates a StackSet instance. When the StackSet is created, it will generate stacks depending on your CloudFormation template. Since this is a StackSet, AWS will build stacks within its predefined region. The following diagram shows the processing model for a StackSet:

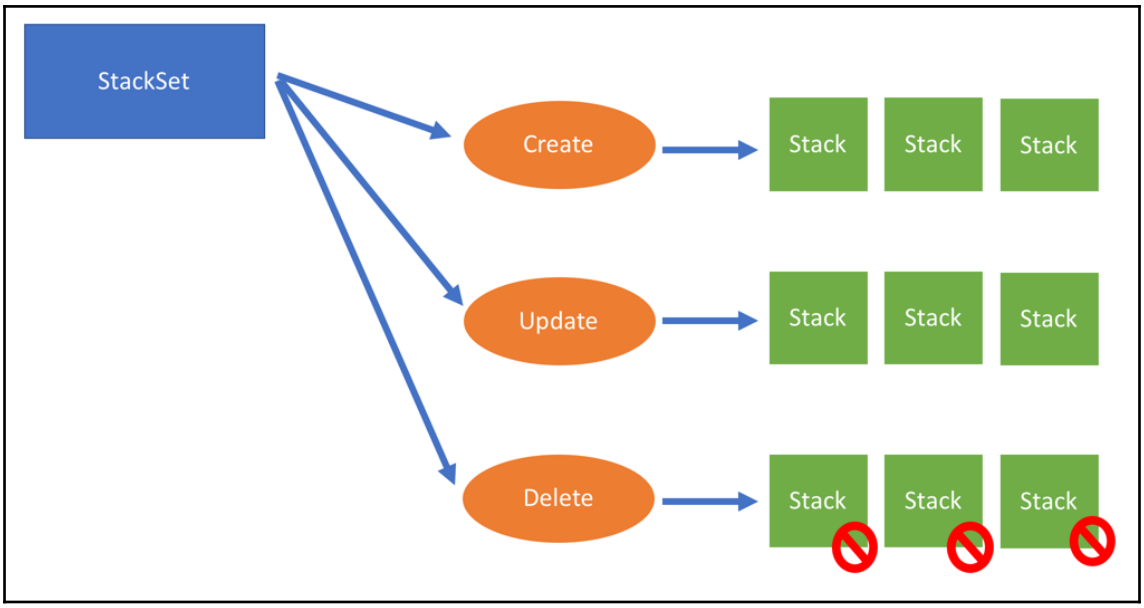


Figure 1.7: Processing model for StackSet

When we edit our StackSet, AWS CloudFormation will update all the stacks. When we perform a deletion operation on a StackSet, AWS CloudFormation will delete all the stacks related to that StackSet.

We will learn more about AWS CloudFormation StackSets in `Chapter 4`, *AWS CloudFormation StackSets*. Some scenarios will be provided to show you how AWS CloudFormation StackSets work.

Exploring the AWS CloudFormation management console

In this section, we'll familiarize ourselves with the AWS CloudFormation management console. To work with AWS CloudFormation, you should have an active AWS account. You can access the AWS CloudFormation management console by opening a browser and navigating to `https://console.aws.amazon.com/cloudformation/`.

Now you should see AWS CloudFormation Management Console, which is shown in the following screenshot. You can click on the **Create new stack** button to create a stack. If you want to create a StackSet, you can click on the **Create new StackSet** button:

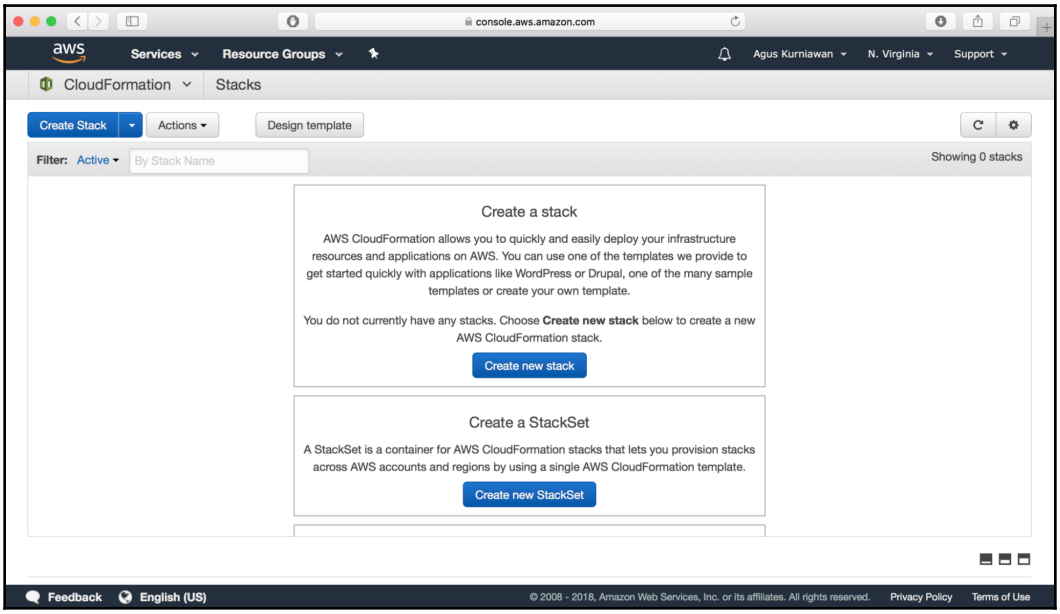


Figure 1.8: AWS CloudFormation Management Console

If you want to build a CloudFormation script template, you can click on the **Design template** button. AWS CloudFormation will provide the CloudFormation designer editor to enable you to build your own infrastructure. You can see this screen shown in *Figure 1.5*.

You should set your working region with AWS CloudFormation. In a CloudFormation StackSet, we can work with different regions for the various stacks.

In the AWS CloudFormation Management Console, we can manage all the stacks and StackSets, including editing and deleting their resources.

Managing CloudFormation using the AWS CLI

If you usually prefer to work in Terminal mode, AWS provides the AWS CLI to manage AWS CloudFormation. You can create a stack and a StackSet from the Terminal. You also can update and delete your CloudFormation projects.

Some commands are defined to manage CloudFormation from the AWS CLI. You should use those commands if you want to work with CloudFormation. For more information about the AWS CLI commands for CloudFormation, check out `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/index.html`.

The following screenshot shows you how to create the CloudFormation stack from the AWS CLI. We will learn how to do this in `Chapter 2`, *Building Your First AWS CloudFormation Project*:

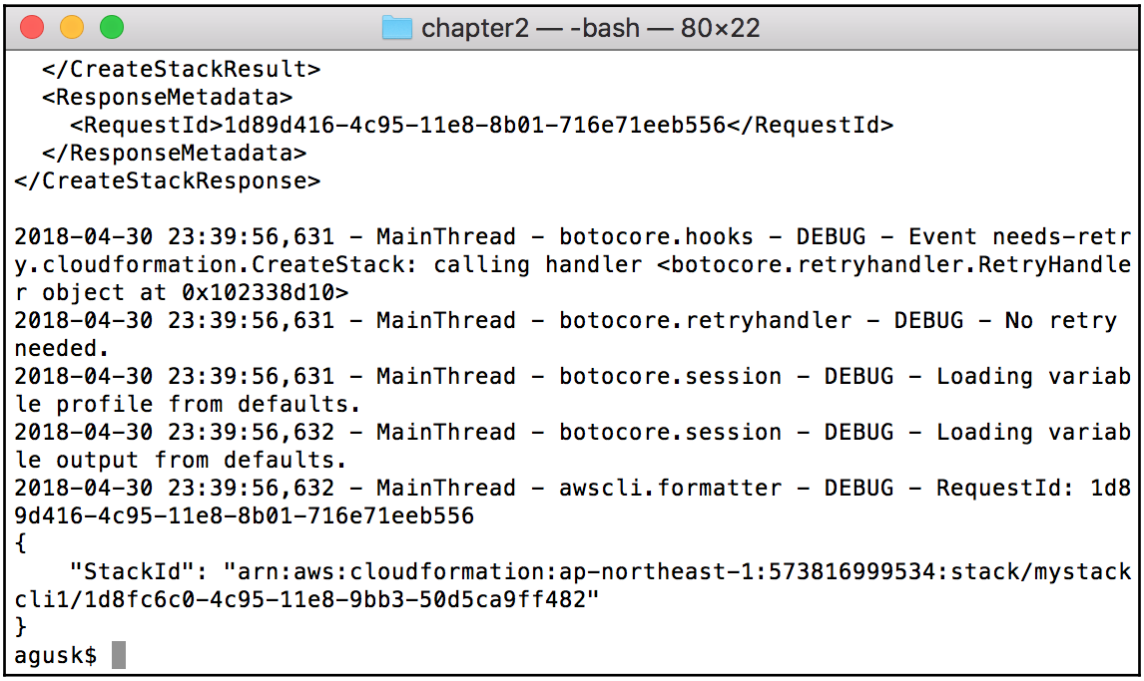


Figure 1.9: Managing CloudFormation using the AWS CLI

This is the end of the first chapter. We reviewed some terms in AWS CloudFormation. We will learn how to use this console in the next chapter.

Summary

We learned a bit about AWS CloudFormation terms, to better understand what AWS CloudFormation is. Next, we will look at how to implement a simple program for IaC and provision it using AWS.

Questions

1. What is IaC?
2. What are the benefits of IaC?
3. What is the main objective of AWS CloudFormation?
4. How does AWS CloudFormation work?
5. How do you build an AWS CloudFormation template?
6. What are infrastructure-nested stacks?
7. Why do we implement AWS CloudFormation StackSets?

2
Building Your First AWS CloudFormation Project

We can build various infrastructure models using AWS CloudFormation. We can add, update, and delete AWS resources in our infrastructure design as IaC and then deploy our IaC scripts to AWS CloudFormation. In this section, we will build a simple project to show how to develop IaC using AWS CloudFormation.

The following topics will be covered in this chapter:

- Preparing a project
- CloudFormation implementation using the management console
- CloudFormation implementation using the AWS CLI
- Managing an AWS CloudFormation project

CloudFormation project scenario

In the previous chapter, we learned what AWS CloudFormation is. In this chapter, we continue our exploration of AWS CloudFormation. We will focus on how to build a simple IaC using CloudFormation.

To minimize problem complexity, we will use a simple AWS resource such as Amazon S3, and create a bucket in Amazon S3 through CloudFormation. To build a CloudFormation project, we will follow the steps shown in the following diagram:

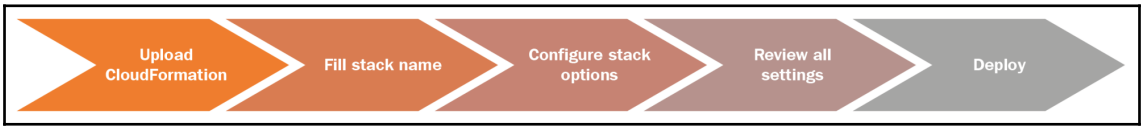


Figure 2.1: Project steps for deploying AWS CloudFormation

The preceding diagram shows how to build and deploy a CloudFormation project. In this scenario, we will use a CloudFormation stack to build an IaC design. The steps are explained as follows:

1. Build IaC scripts in a CloudFormation template to define the AWS resources that are used to build an infrastructure
2. Upload the template file to the CloudFormation server
3. Create a CloudFormation stack
4. Configure all options from the CloudFormation stack
5. Review all settings before deploying to the CloudFormation server
6. Deploy a CloudFormation stack

We will apply these steps to our CloudFormation project. Each step will be executed in the next section.

Preparation

Technically, we don't need any special preparation. You should have an active AWS account in order to build CloudFormation projects, and you also need an internet connection since we build and manage AWS through the internet network.

Lastly, you should be aware of pricing. Currently, CloudFormation is free, but the AWS resources used in a CloudFormation project are probably chargeable. Some AWS resources are free with certain limitations.

Implementing a CloudFormation project using the management console

In this section, we are going to build our first CloudFormation project graphically through the CloudFormation management console. Follow these steps:

1. Open a browser and navigate to `https://console.aws.amazon.com/cloudformation`.

2. Log in with your active AWS account. After successful login, you should get the AWS CloudFormation management console, as shown in the following screenshot:

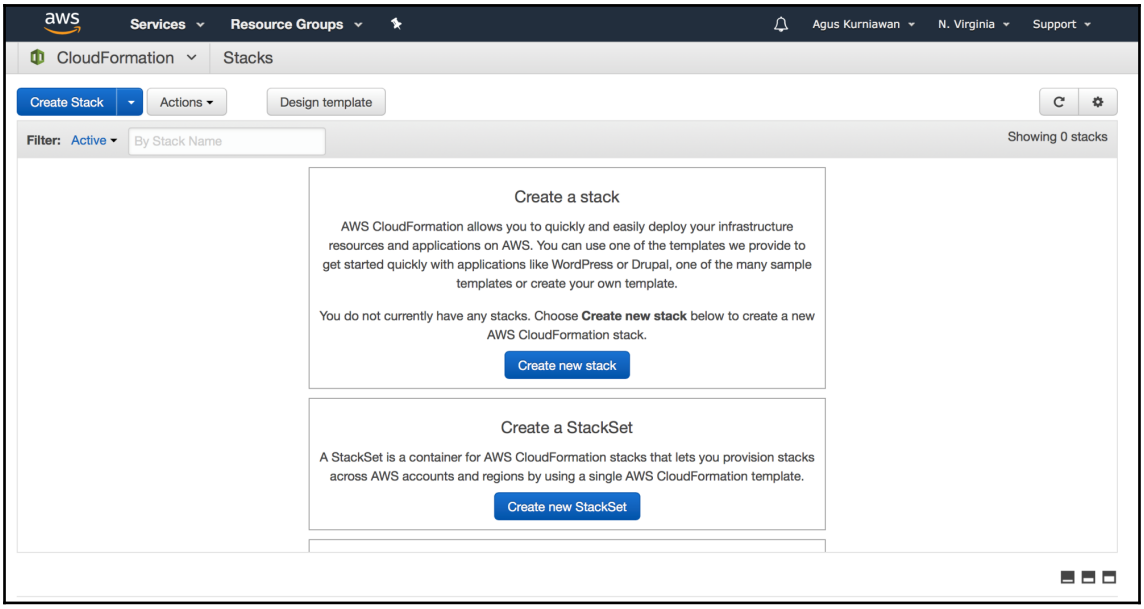


Figure 2.2: AWS CloudFormation management console

3. To start creating a simple CloudFormation project, first create a stack.
4. Click on the **Create new stack** button, shown in *Figure 2.2*.

5. You should now see the screen shown in the following screenshot:

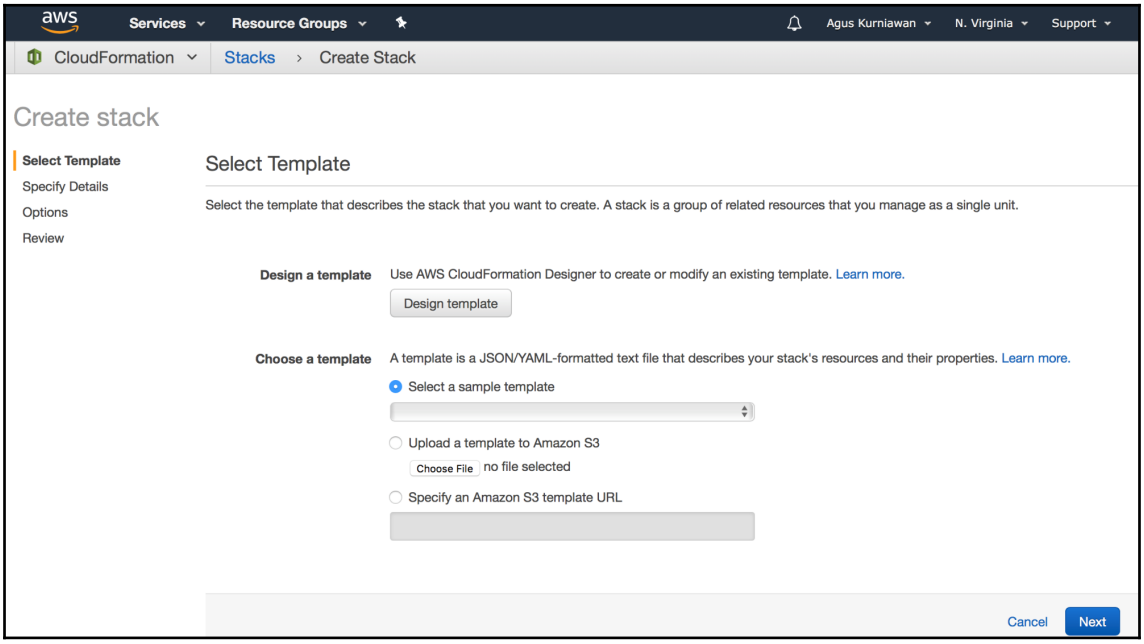


Figure 2.3: Selecting a template for CloudFormation

6. From the screen shown in the preceding screenshot, upload a CloudFormation template file. You can upload the template file or create a template using the **Design template** option. If you have a CloudFormation template file that has already been uploaded to Amazon S3, you can specify the file path from Amazon S3.
7. For this demo, we uploaded a CloudFormation template file from local to Amazon S3. The template file is in JSON and YAML file formats.
8. Now, we can create a template file in JSON and YAML. You can select a template file in JSON (`hello-cloudformation.json`) or YAML (`hello-cloudformation.yaml`).

The following is the content of the `hello-cloudformation.json` file:

```
{
 "Resources" : {
    "MySimpleBucket" : {
       "Type" : "AWS::S3::Bucket"
    }
 }
}
```

The following is the content of the `hello-cloudformation.yaml` file:

```
Resources:
 MySimpleBucket:
   Type: AWS::S3::Bucket
     Properties:
       AccessControl: PublicRead
```

9. After creating a CloudFormation template file in JSON or YAML, you can upload it to the CloudFormation management console.
10. Select the **Upload a template to Amazon S3** option shown in *Figure 2.3*.
11. Then, click on the **Choose File** option to select your template file from the local computer.
12. Wait until your template file is uploaded to Amazon S3.
13. After it is uploaded, click on the **Next** button to continue the steps of the project.
14. You should now see the screen shown in the following screenshot:

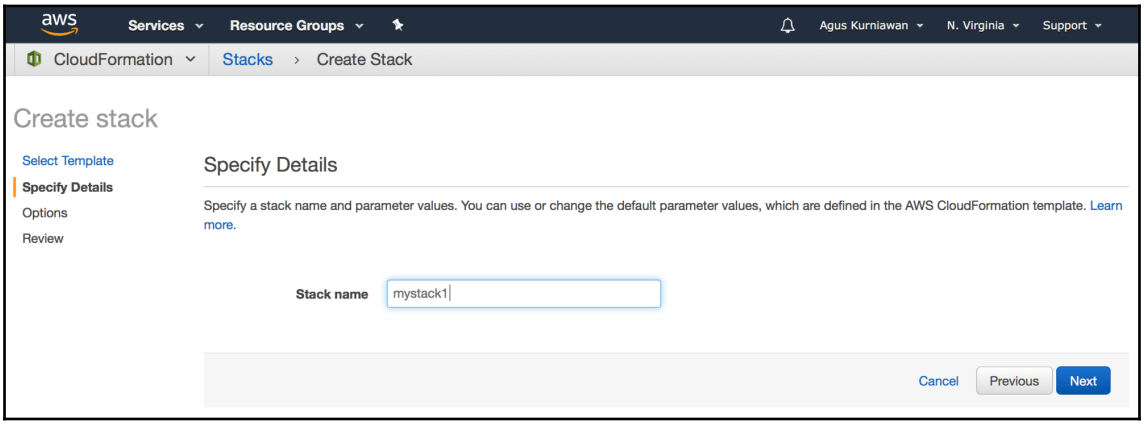


Figure 2.4: Adding a stack name

15. Fill in your stack name. This is used for stack identity in CloudFormation.
16. After you have filled in the stack name, click on the **Next** button to continue.
17. You will see the following screenshot for configuring stack options such as tags and permissions:

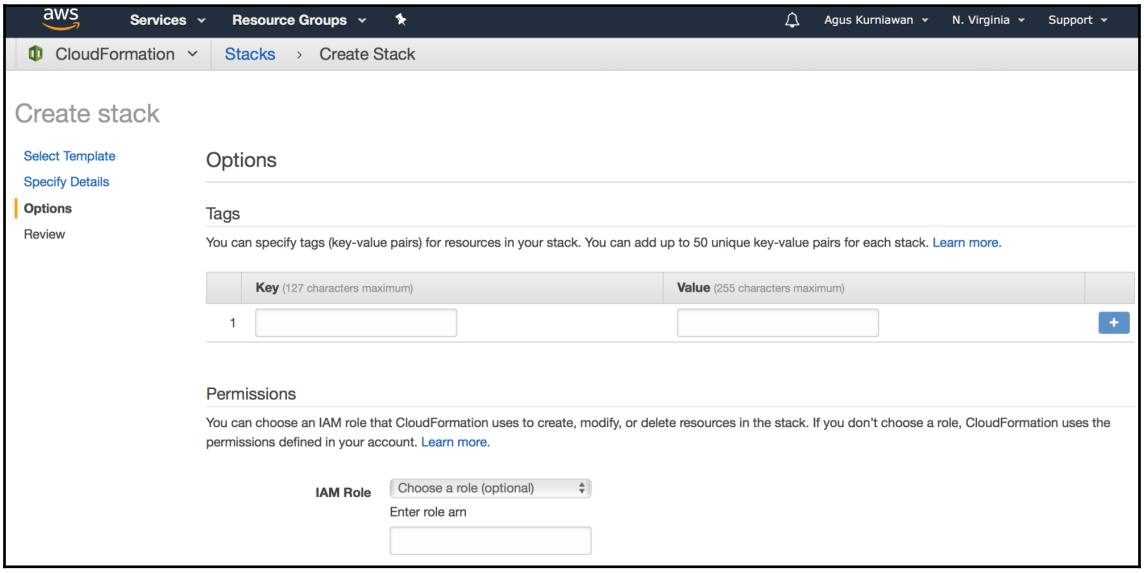


Figure 2.5: Configuring stack options

18. In this demo, we won't configure any options in the CloudFormation stack. We can skip this step. Click on the **Next** button to continue.
19. You can now see the review screen, as shown in the following screenshot:

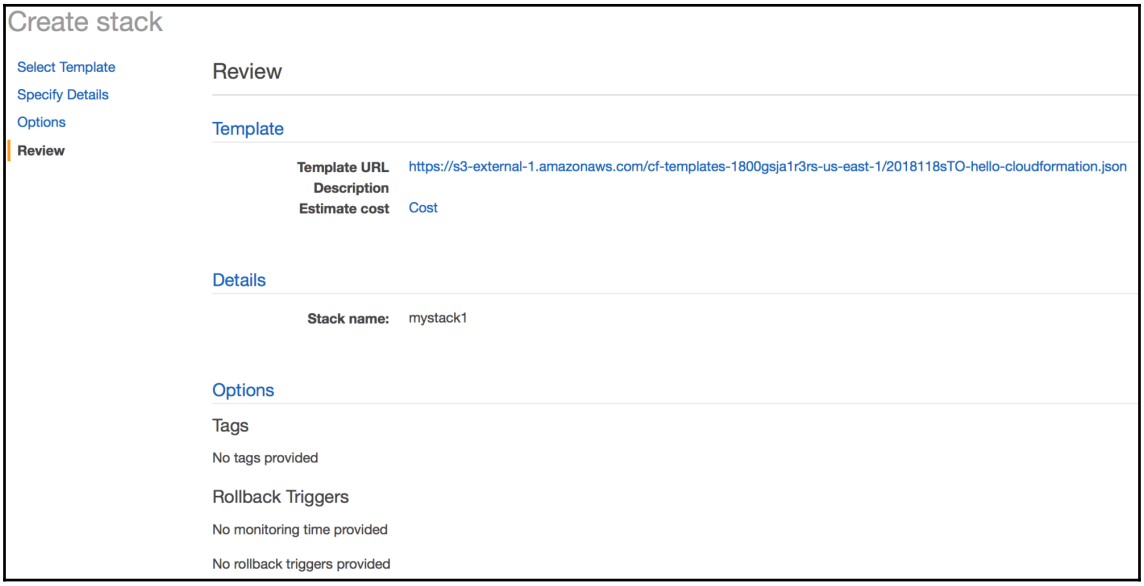


Figure 2.6: Confirmation before executing and deploying to Amazon AWS

20. Review all the information that you have set.
21. Once done, click on the **Create** button to execute your template and then deploy all the resources from the predefined template to Amazon AWS. Your browser should be redirected to the CloudFormation dashboard. You should now see the status of the stack being created, as shown in the following screenshot:

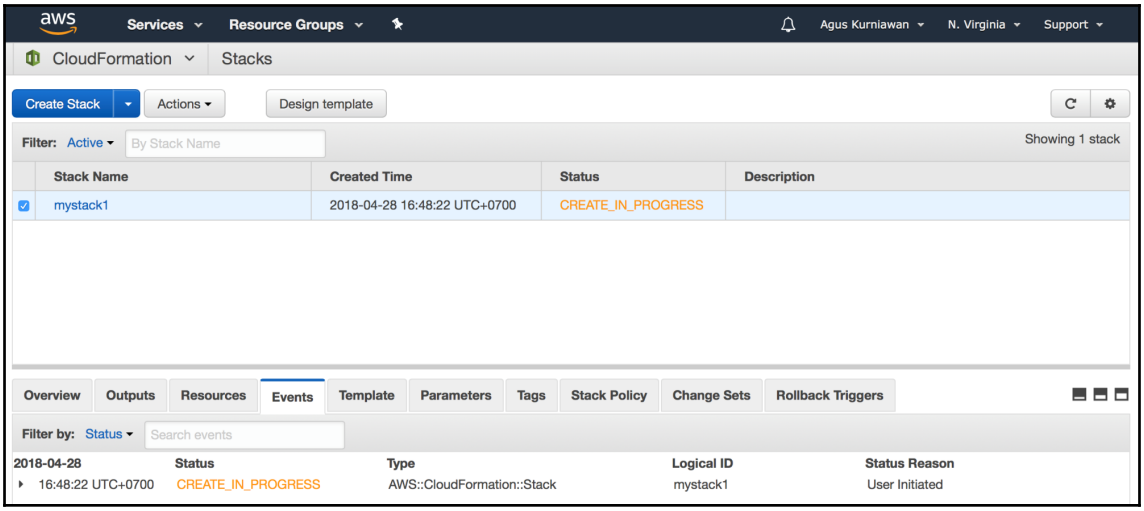


Figure 2.7: A stack being created in CloudFormation

22. If you want to get an update on the progress of the creation, click on the circle-arrow at the top right.
23. Once the process is done, you should see it in the **Status** column, in the list shown in the following screenshot:

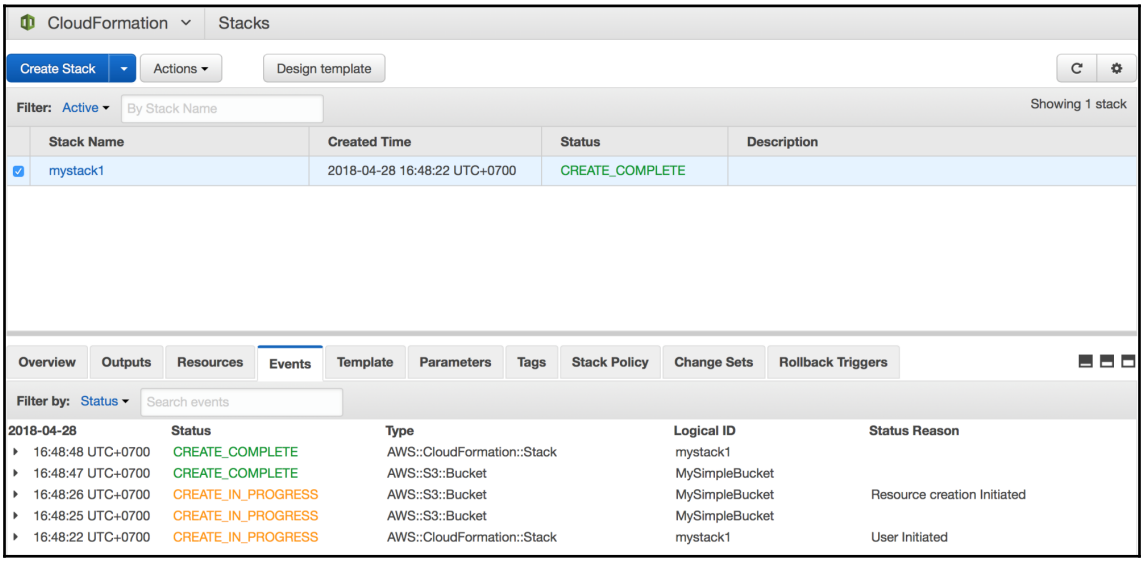


Figure 2.8: A stack was completely created

24. If you want to see the details of your created stack, you can click on your stack name, as shown in the preceding screenshot.
25. Then, you will see details from the selected CloudFormation stack. The following screenshot shows an example of a detailed stack:

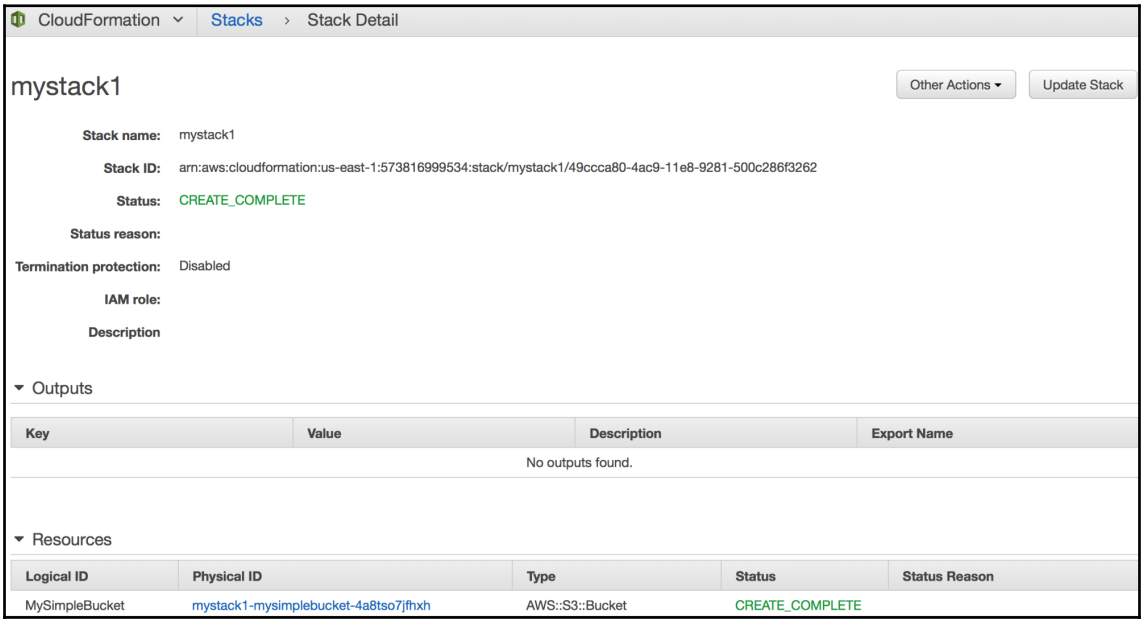


Figure 2.9: Getting details from the stack

26. AWS CloudFormation creates all the resources that you have already defined in the template file.
27. In our demo, we create Amazon S3. We can verify it by opening the Amazon S3 management console at `https://console.aws-amazon.com/s3`.
28. On the Amazon S3 management console, you should see a bucket with the name that is defined in your CloudFormation stack, as shown in the following screenshot:

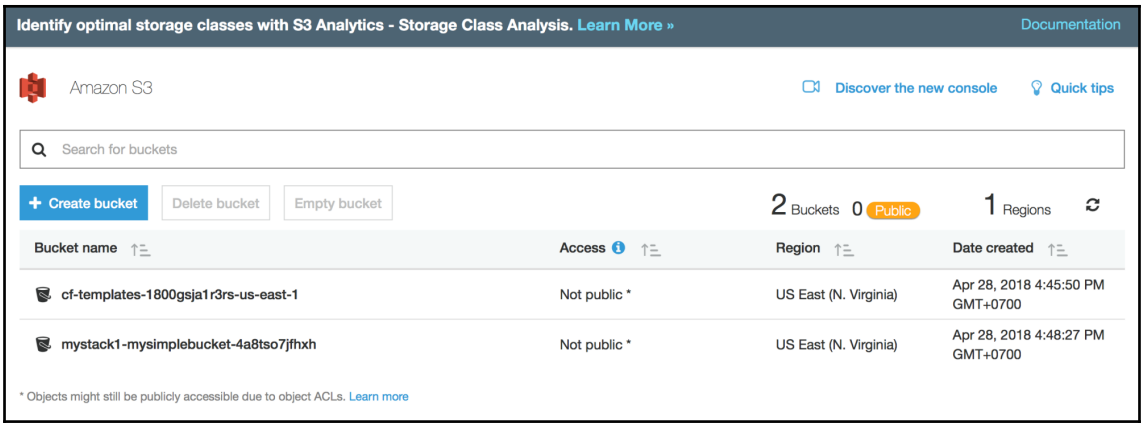


Figure 2.10: A S3 bucket was created in AWS CloudFormation

This is the end of the section on how to build AWS CloudFormation graphically through the CloudFormation management console.

Next, we will work through a similar scenario using the AWS CLI.

Implementing a CloudFormation project using the AWS CLI

In this section, we'll explore how to work with the AWS CLI on CloudFormation. In the previous section, we built and deployed CloudFormation graphically through the CloudFormation management console. We will perform a similar action to build and deploy CloudFormation using the AWS CLI.

Let's explore!

Setting up the AWS Command Line Interface (CLI)

The AWS CLI is a tool from Amazon for managing Amazon AWS in Terminal mode. We manage all AWS resources from this Terminal. This tool supports Windows, Linux, and macOS. If you work with the Windows platform, you can download this tool from the following website, selecting your Windows version:

- Windows 64-bit: `https://s3.amazonaws.com/aws-cli/AWSCLI64.msi`
- Windows 32-bit: `https://s3.amazonaws.com/aws-cli/AWSCLI32.msi`

For Linux and Mac, you can install the AWS CLI through `pip` with the Python runtime. Type the following command:

```
$ pip install awscli
```

If you want to upgrade your AWS CLI through `pip`, type the following command:

```
$ pip install awscli --upgrade --user
```

If you have installed or upgraded the CLI already, you can verify AWS CLI by checking its version. Type the following command in the Terminal:

```
$ aws --version
```

You should see the AWS CLI version. For instance, you can see my AWS CLI version in the following screenshot:

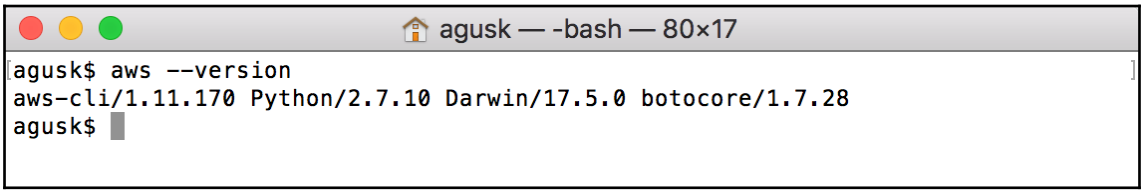


Figure 2.11: Checking the AWS CLI version

To configure the AWS CLI with your current AWS account, type the following command:

```
$ aws configure
```

You should prepare all the required access and secret keys. I recommend reading the guidelines on this at `https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html`.

The next step is to configure security access on the AWS CLI to enable working with CloudFormation.

Configuring security access for CloudFormation

To work with CloudFormation from the AWS CLI, we need to configure our security access and rights. In this demo, we will build and deploy CloudFormation with Amazon S3, so we need to configure security settings for CloudFormation and Amazon S3 on the AWS CLI.

The following are the steps for adding security access to CloudFormation and Amazon S3:

1. Open a browser and navigate to the AWS IAM management console at `https://console.aws.amazon.com/iam`.
2. Once you're at the IAM management console, click on the **Policies** option in the left-hand menu. You should see the screen shown in the following screenshot:

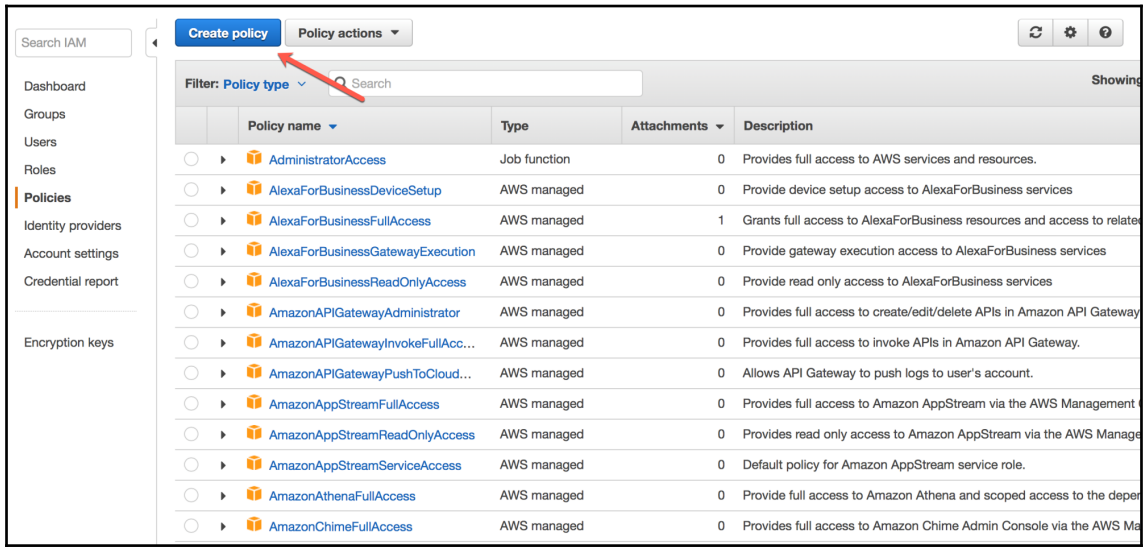


Figure 2.12: Add a new policy for CloudFormation

3. Now, create a custom policy for CloudFormation.
4. Click on **Create policy**, which is indicated by the arrow in *Figure 2.12*. Then, you should have a policy form, as shown in *Figure 2.13*.

5. We will add policy scripts in JSON format, so click on the **JSON** tab on the creation form, as shown in the following screenshot:

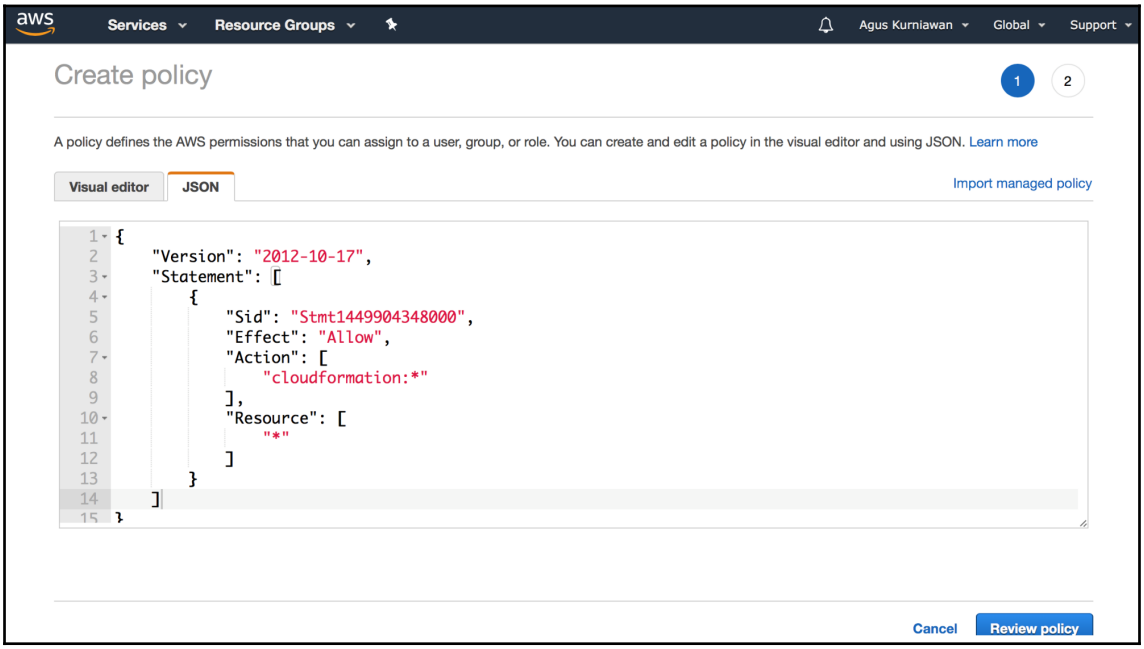


Figure 2.13: Adding a policy to access CloudFormation resources

6. In this scenario, we give full access to CloudFormation. You can write these scripts and paste them to the **JSON** tab shown in *Figure 2.13*:

```
{
  "Version": "2012-10-17",
  "Statement": [
                {
                  "Sid": "Stmt1449904348000",
                  "Effect": "Allow",
                  "Action": [
                      "cloudformation:*"
                  ],
                  "Resource": [
                      "*"
                  ]
                }
   ]
}
```

7. Once you have pasted the scripts, you can click on the **Review policy** button and you should see the form shown in *Figure 2.14*.
8. Fill in the name and description of your policy. For instance, the name of the review policy in the following screenshot is `AWSCloudFormationLRW`:

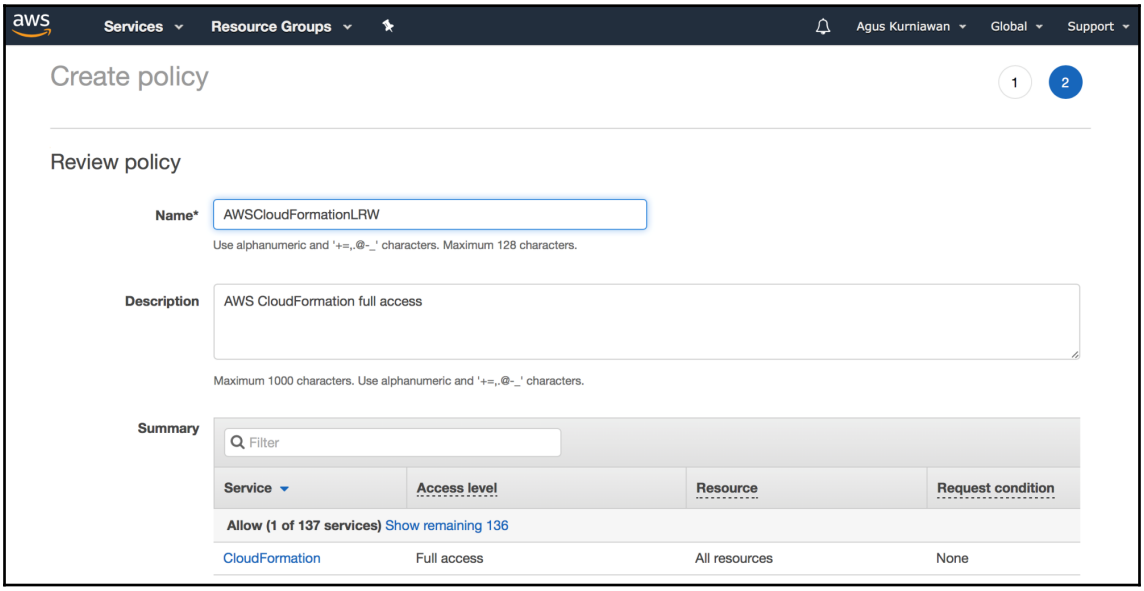


Figure 2.14: Fill in the policy name and description

9. Once done, click on **Create policy** to start creating a policy. Now, we continue by adding that policy to our account.
10. Click on the **Users** section in the menu on the left and select the user account that is used in the AWS CLI.
11. In the **Summary** section of your IAM user, you should have the following screenshot:

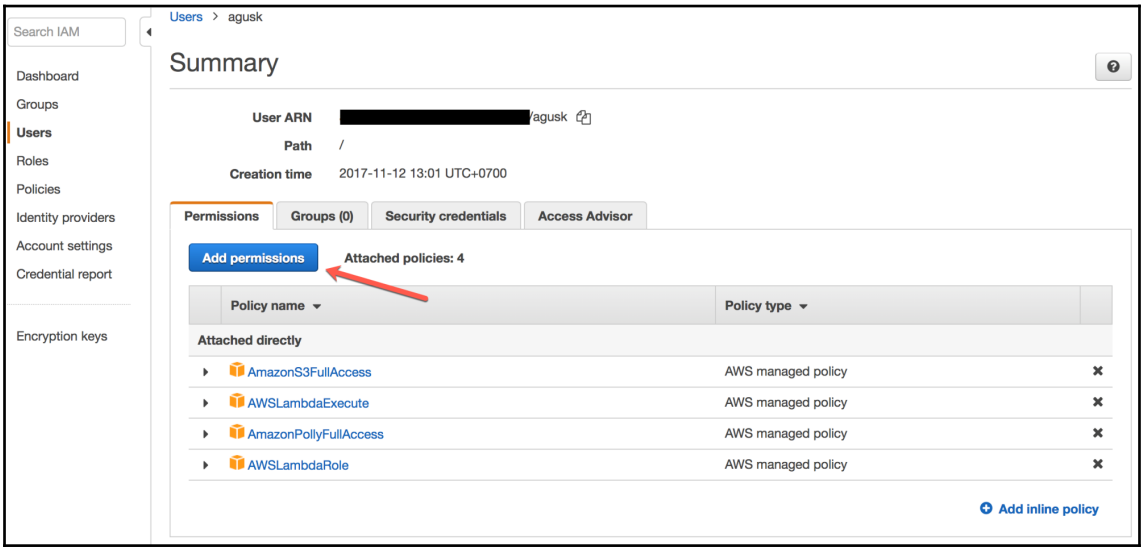


Figure 2.15: Adding a policy to a user

12. To add your own policy to your IAM account, click on **Add permissions** on the **Permissions** tab; see *Figure 2.15*. You should then see the screen shown in *Figure 2.16*.
13. Select the **Attach existing policies directly** option in the **Grant permissions** section and type your policy name, for instance, `AWSCloudFormationLRW`.

14. You should see your own policy. Click on it, as shown in the following screenshot:

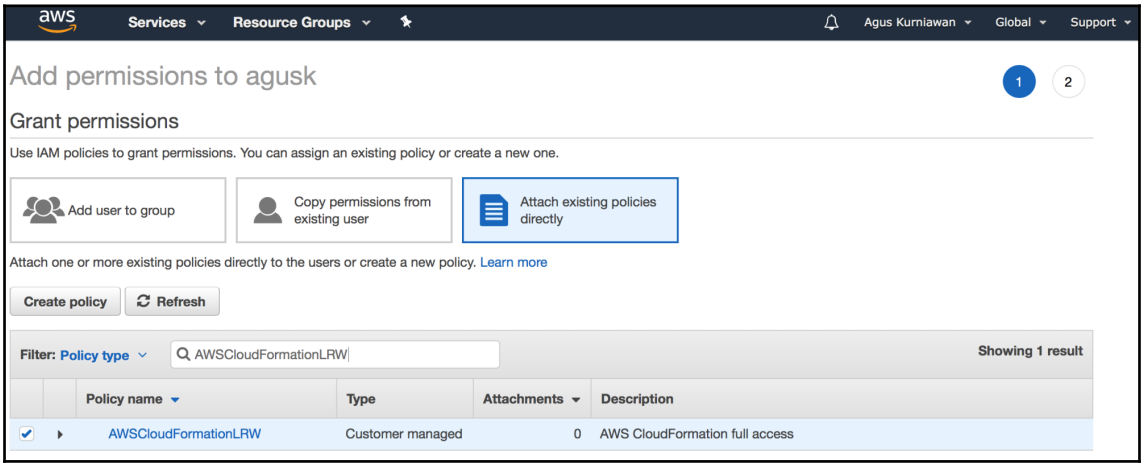


Figure 2.16: Adding permissions from your own policy

15. After selecting your policy, click on **Next:Review** at the bottom. Now you should see the form shown in *Figure 2.17*.
16. Once done, click on **Add permissions** to add that policy to your IAM account:

Figure 2.17: Confirmation of adding permission

17. Now, your IAM user has a CloudFormation policy with full access.
18. Using the same approach, we also need to add the **AmazonS3FullAccess** policy to your IAM account.
19. Add an existing policy with **AmazonS3FullAccess,** as shown in the following screenshot:

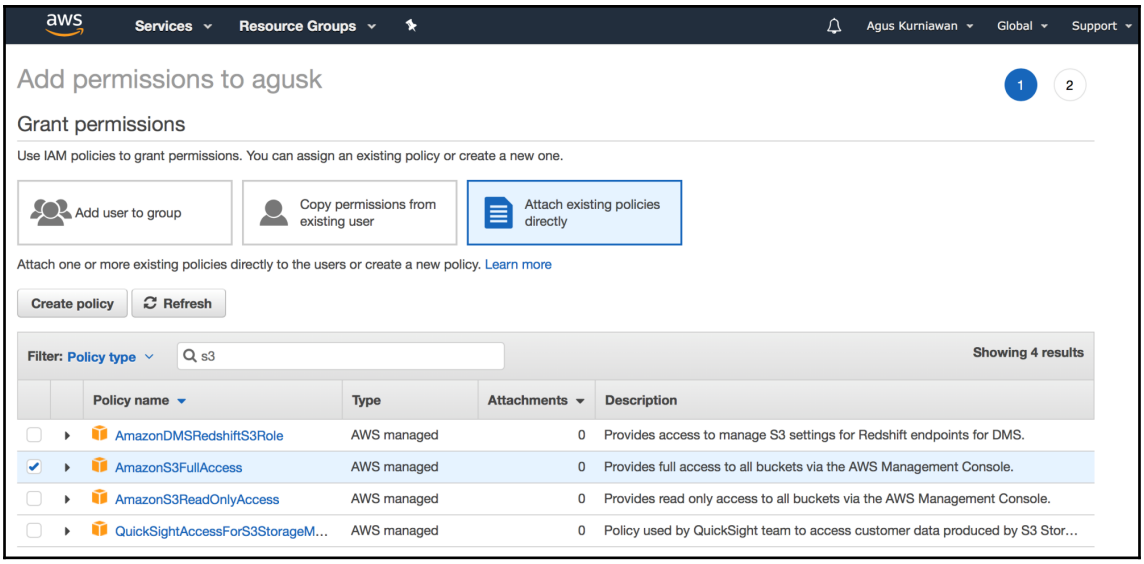


Figure 2.18: Adding the AmazonS3FullAccess policy to a user

20. Select the **AmazonS3FullAccess** policy and then add it to your IAM user.
21. Once done, your IAM should have CloudFormation and **AmazonS3FullAccess** policies, as shown in the following screenshot:

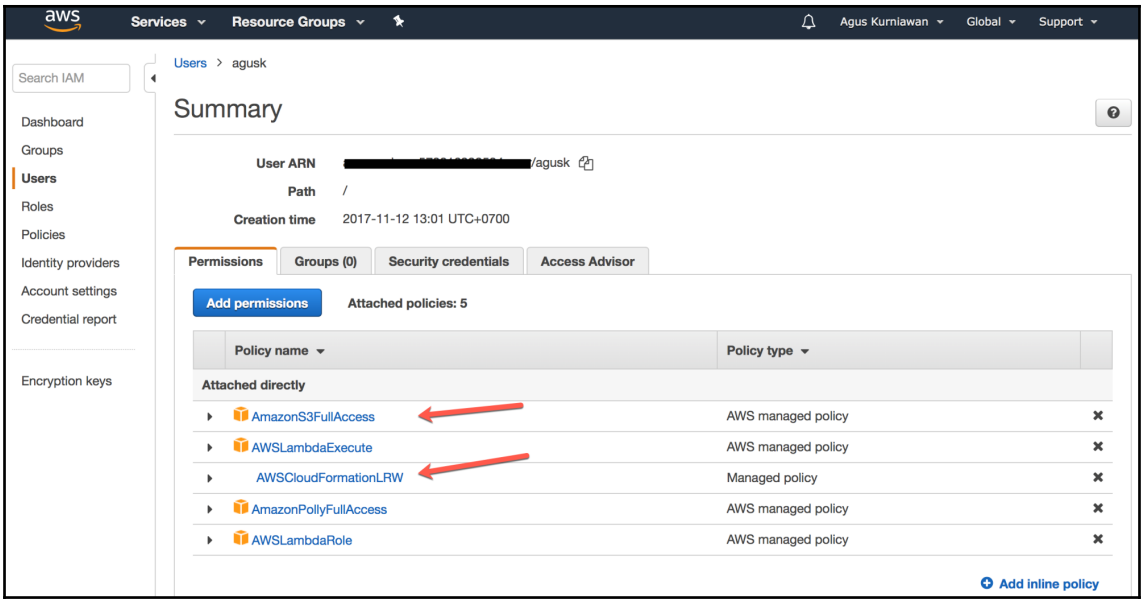


Figure 2.19: CloudFormation and Amazon S3 policies have been added

Now, your IAM user has CloudFormation and **AmazonS3FullAccess** policies. Now, you can manage CloudFormation from the AWS CLI.

The next step is to build CloudFormation and then deploy it to AWS CloudFormation.

Building and deploying CloudFormation

In this section, we will use the AWS CLI to deploy AWS CloudFormation. To work with CloudFormation in the AWS CLI, we need to know some CloudFormation commands. You can find all the commands for CloudFormation at `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/index.html`.

We will use the same template as in the first demo, `hello-cloudformation.json`. We will upload this template to CloudFormation and then deploy it.

Now open the Terminal and navigate to the directory where `hello-cloudformation.json` is located. To create a stack, we can use the `cloudformation create-stack` command. Type the following command to upload the template and create a stack:

```
$ aws cloudformation create-stack --stack-name mystackcli1 --template-body
file://./hello-cloudformation.json --debug
```

The preceding command is explained as follows:

- `--stack-name mystackcli1`: This defines the stack name. In this case, the stack name is `mystackcli1`.
- `--template-body file://./hello-cloudformation.json`: This is a template file. You should use `file://` with the full template file path. In this demo, Terminal has already been navigated to a directory where the `./hello-cloudformation.json` file is located.
- `--debug`: This is a parameter to enable verbosity so we can see all the verbose messages from the CLI.

For information about the CloudFormation `create-stack` command, I recommend you read the document at: `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/create-stack.html`.

If this operation is executed successfully, you should get **StackId** on the Terminal. You can see my program output in the following screenshot:

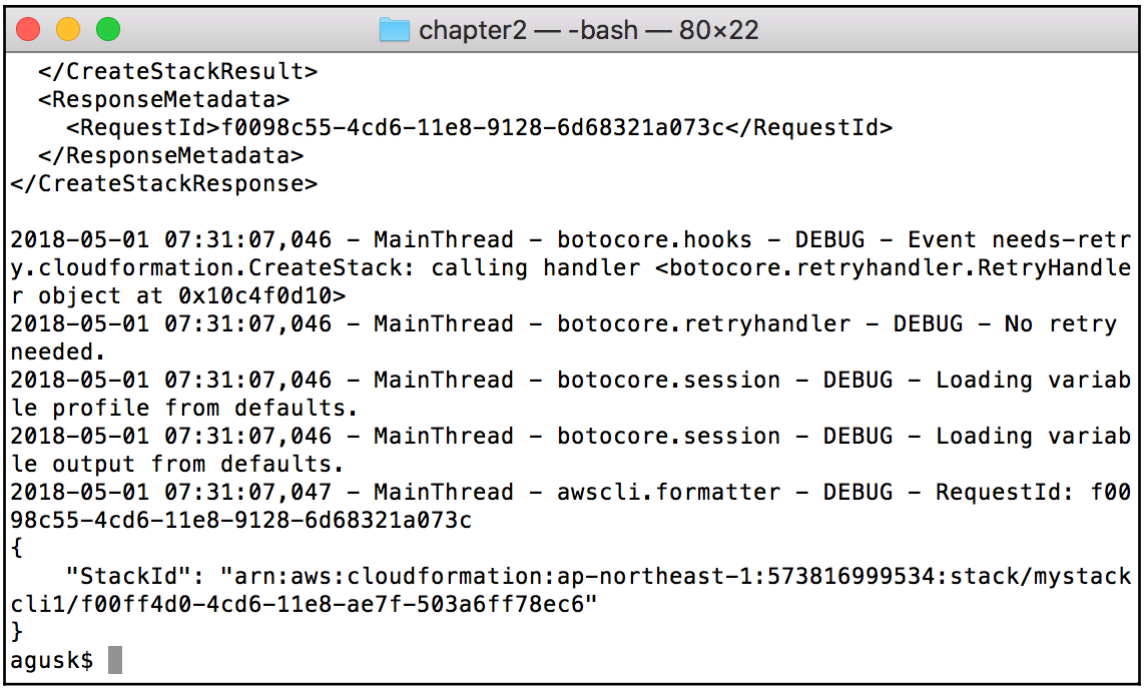


Figure 2.20: Creating a CloudFormation solution using the AWS CLI

To verify your operation is complete, you can use the `list-stacks` command from the CloudFormation CLI. For further information about the `list-stacks` command, go to `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/list-stacks.html`. Now, type the following command:

```
$ aws cloudformation list-stacks
```

You should see a status from the stack operation, as you can see in the following screenshot, for example. A stack status can be found in the **StackStatus** option, which is indicated by the rectangle:

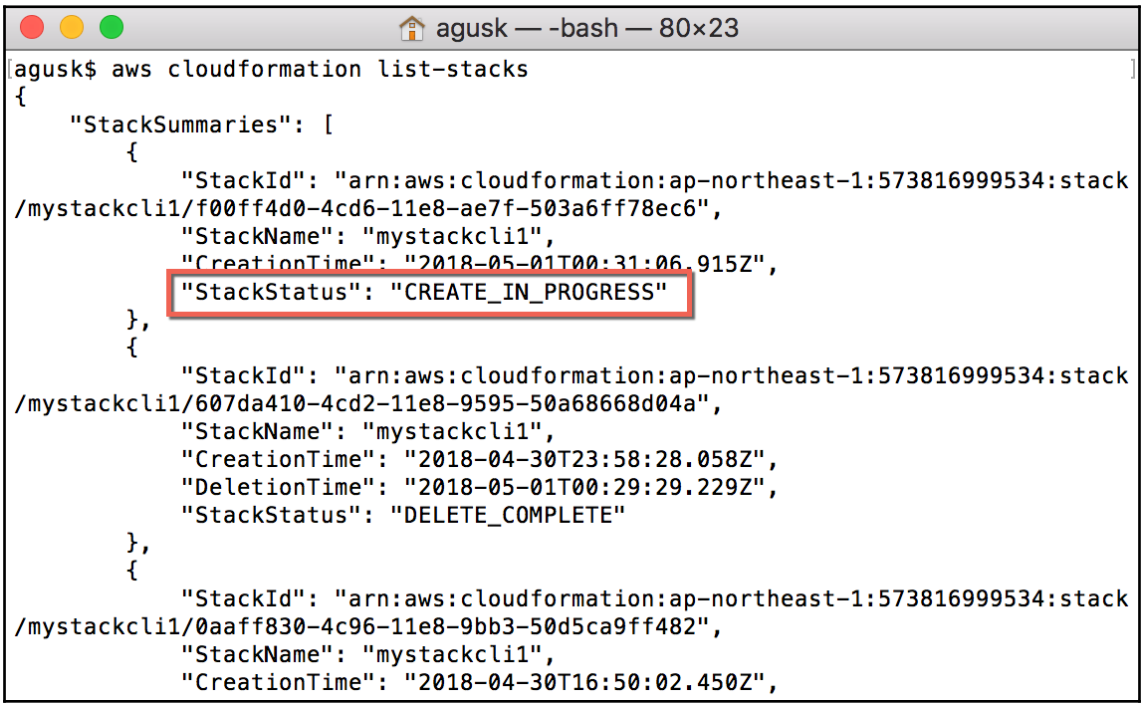


Figure 2.21: Showing all CloudFormation stacks from the CLI

We can also find out stack details using the `describe-stacks` command by passing in the stack name. Information about the `describe-stacks` command can be found at `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/describe-stacks.html`. For the demo, I want to see the details of the stack with the name `mystackcli1`:

```
$ aws cloudformation describe-stacks --stack-name mystackcli1
```

After execution, you should see the stack status in the **StackStatus** option, as shown in the following screenshot:

```
agusk$ aws cloudformation describe-stacks --stack-name mystackcli1
{
    "Stacks": [
        {
            "StackId": "arn:aws:cloudformation:ap-northeast-1:573816999534:stack
/mystackcli1/f00ff4d0-4cd6-11e8-ae7f-503a6ff78ec6",
            "Tags": [],
            "EnableTerminationProtection": false,
            "CreationTime": "2018-05-01T00:31:06.915Z",
            "StackName": "mystackcli1",
            "NotificationARNs": [],
            "StackStatus": "CREATE_COMPLETE",
            "DisableRollback": false,
            "RollbackConfiguration": {}
        }
    ]
}
agusk$
```

Figure 2.22: Checking the stack status through the CLI

If the status of the stack is **CREATE_COMPLETE**, stack creation has succeeded. You can verify this on the CloudFormation management console. Select the region used by your AWS CLI. The following screenshot shows that my stack was created by the AWS CLI:

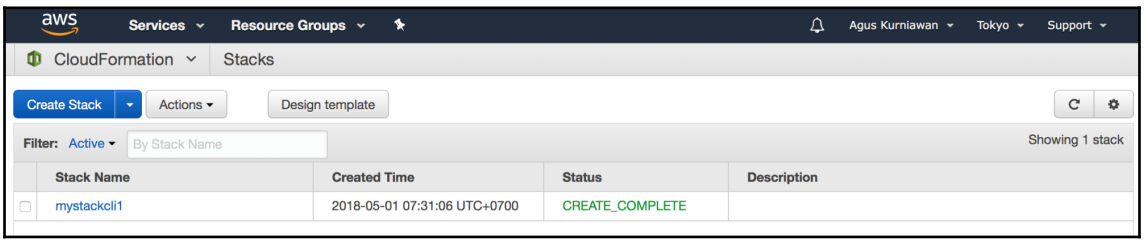


Figure 2.23: A CloudFormation stack has been created from the AWS CLI

This is the end of the section on how to create a CloudFormation stack using the AWS CLI. Next, we will modify an existing CloudFormation stack.

Editing a CloudFormation project

In this section, we will update our CloudFormation project. For a simple update scenario, we will modify a bucket name from Amazon S3. We will modify the CloudFormation template and then apply this template to the CloudFormation server.

We will perform project updates using the CloudFormation management console and the AWS CLI. Each method will be explained in the next section.

Editing CloudFormation using the management console

We will modify an existing CloudFormation stack. For a simple scenario, we will change a bucket title on Amazon S3. We will use the modified template file, `hello-cloudformation-v2.json`, from the previous demo.

The following are the steps for modifying an existing stack using the CloudFormation management console:

1. Open a browser and navigate to AWS CloudFormation at `https://console.aws.amazon.com/cloudformation/`.
2. On the AWS CloudFormation dashboard, select the stack that you want to modify.
3. Then, click on **Actions** and select the **Update Stack** option from the sub-menu. You can see this in the following screenshot:

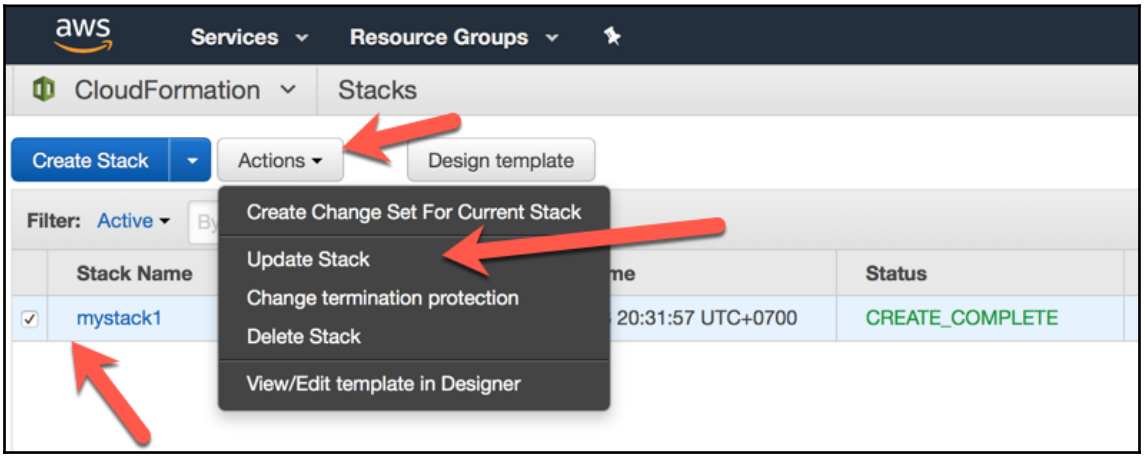


Figure 2.24: Editing a selected stack in the CloudFormation management console

4. After selecting a stack for editing, select the **Upload a template to Amazon S3** option.
5. Click on the **Next** button to read the CloudFormation template. You should now see the following screen:

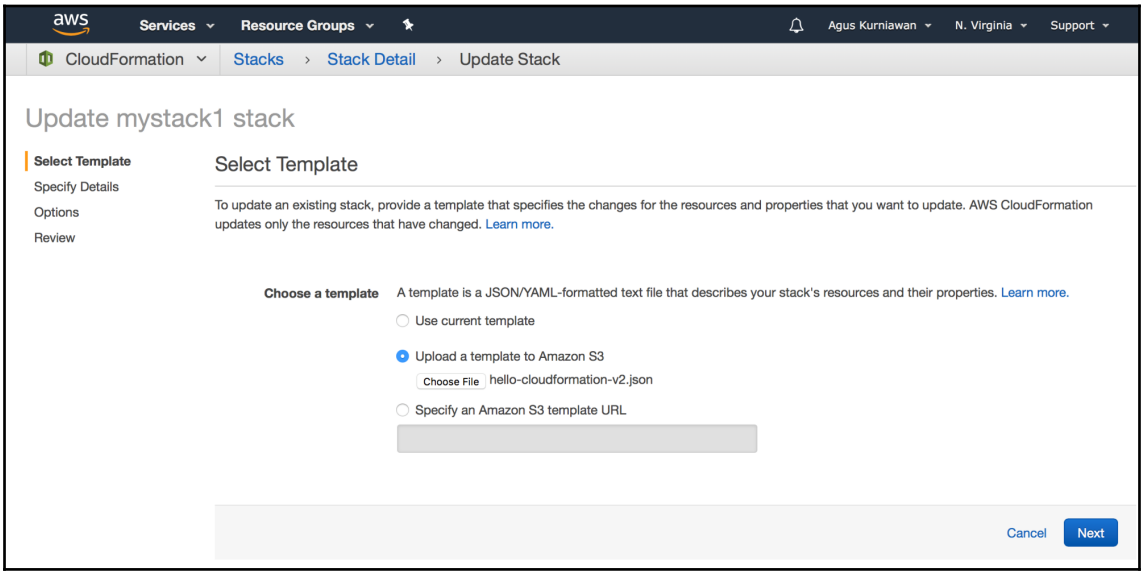


Figure 2.25: Uploading an updated template

6. Upload the updated template, for instance, `hello-cloudformation-v2.json`. When done, click on the **Next** button.
7. CloudFormation will check your template. If the content of the template file is similar, CloudFormation will not perform the next process.

8. If there is a change, CloudFormation will continue the process. You should see the screen shown in *Figure 2.26*. We can't change the stack name. Now, click on the **Next** button:

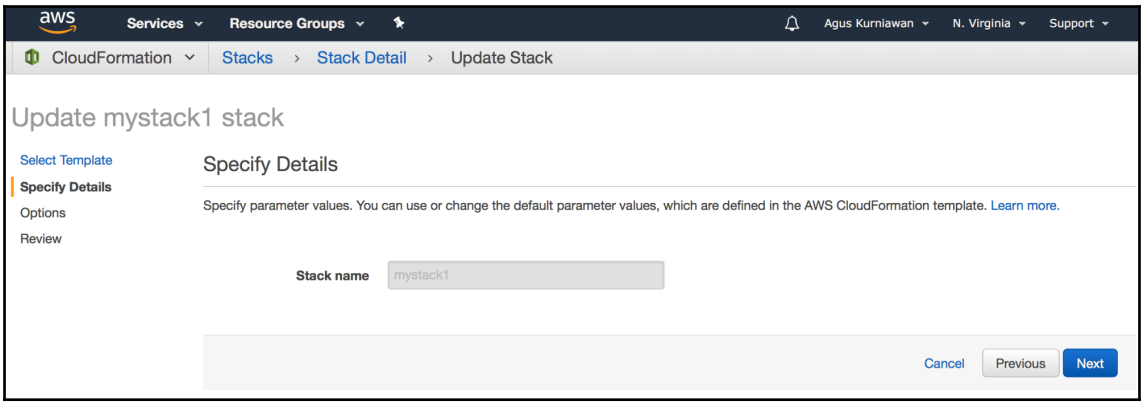


Figure 2.26: Showing a stack for updating

9. You should now see the screen shown in *Figure 2.27*. Change your stack options and click on the **Next** button:

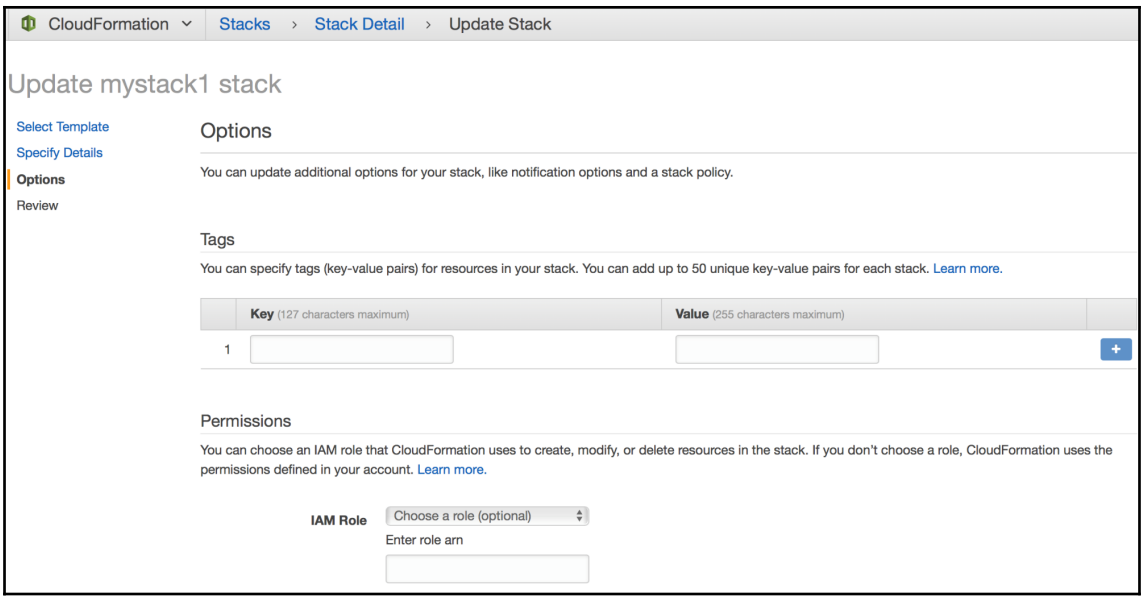


Figure 2.27: Setting options for a stack

10. You should now see the review screen, as shown in *Figure 2.28*. Check all your changes and click on **Update** to deploy the changes to CloudFormation:

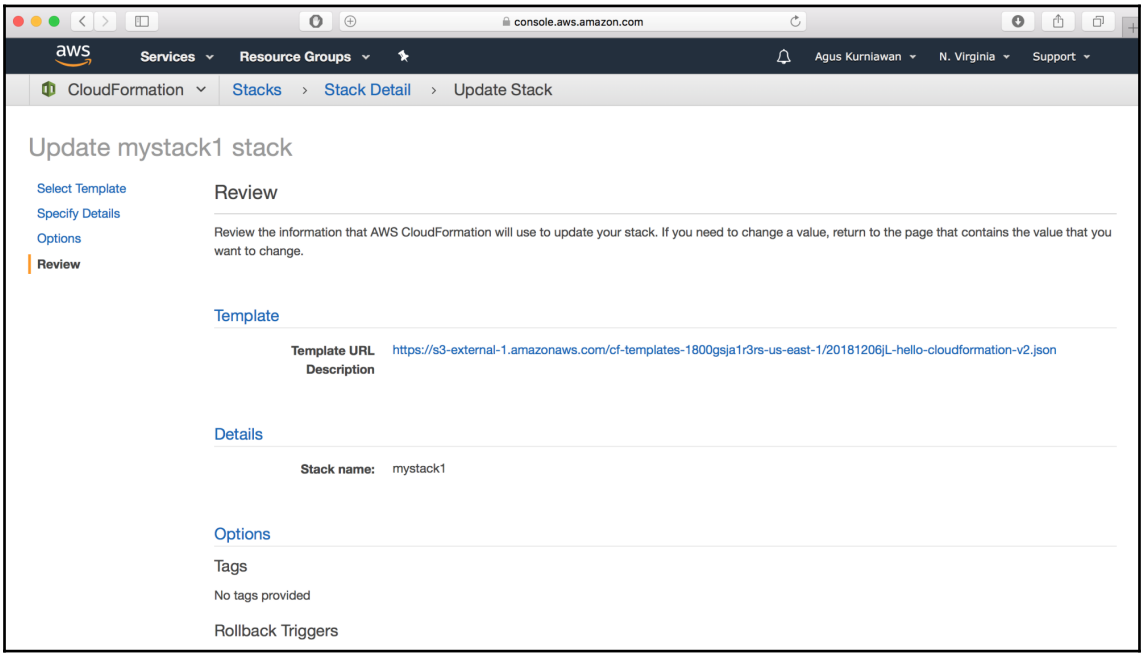


Figure 2.28: Reviewing all changes before deploying

11. CloudFormation will apply all changes to your stack. You can see the progress of the process in the **Status** column on the **Stacks** dashboard.

12. Once done, you should see **UPDATE_COMPLETE** in the **Status** column, as shown in the following screenshot:

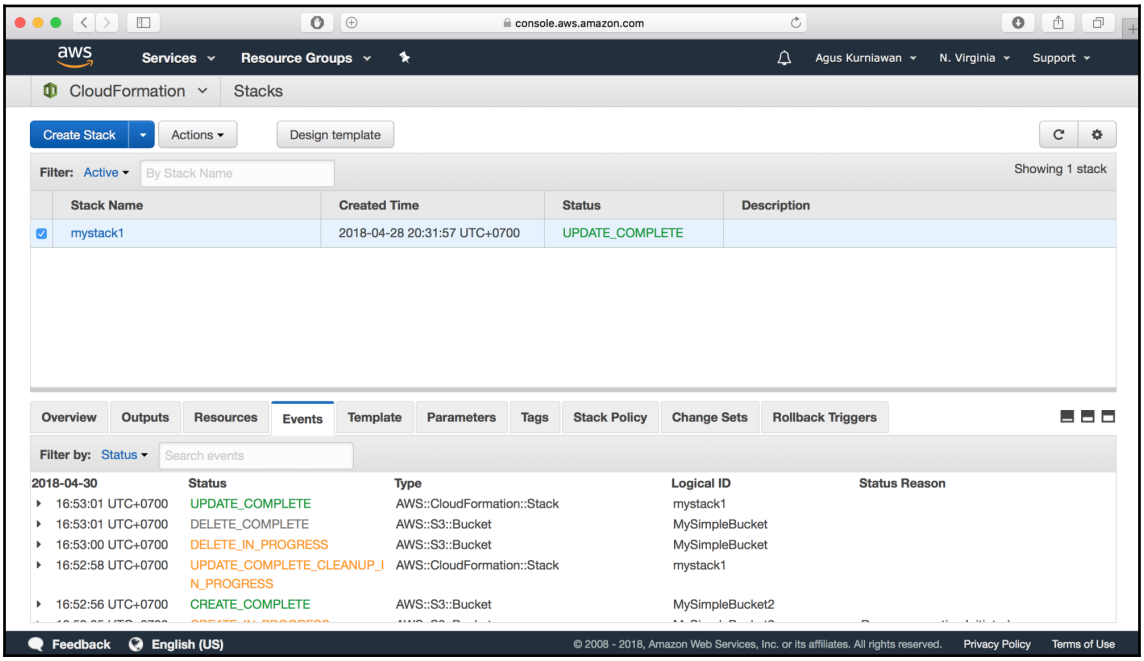


Figure 2.29: Stack update is complete

This is the end of the section on editing a CloudFormation stack using the management console. Next, we will perform the same process using the AWS CLI.

Editing CloudFormation using the AWS CLI

In this section, we will focus on editing an existing CloudFormation stack using the AWS CLI. We have the modified template file, `hello-cloudformation-v2.json`. We will update an existing CloudFormation stack by replacing it with the `hello-cloudformation-v2.json` file.

Now open the Terminal and navigate to the directory where `hello-cloudformation-v2.json` is located. To update a stack, we can use `cloudformation update-stack`. Type the following command to upload the template and create a stack:

```
$ aws cloudformation update-stack --stack-name mystackcli1 --template-body
file://./hello-cloudformation-v2.json --debug
```

The preceding command is explained as follows:

- `--stack-name mystackcli1`: This is the stack that will be updated. In this case, the stack name is `mystackcli1`.
- `--template-body file://./hello-cloudformation-v2.json`: This is a changed template file. You should use `file://` with the full template file path. In this demo, Terminal has already been navigated to the directory where the `./hello-cloudformation-v2.json` file is located.
- `--debug`: This is a parameter to enable verbosity, so we can see all verbose messages from the CLI.

For further information about the CloudFormation `update-stack` command, I recommend reading the document at `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/update-stack.html`.

If this operation is executed successfully, you can check the current operation status using the `describe-stacks` command. Type the following command:

```
$ aws cloudformation describe-stacks --stack-name mystackcli1
```

You should see the status of the **StackStatus** option, as shown in the following screenshot:

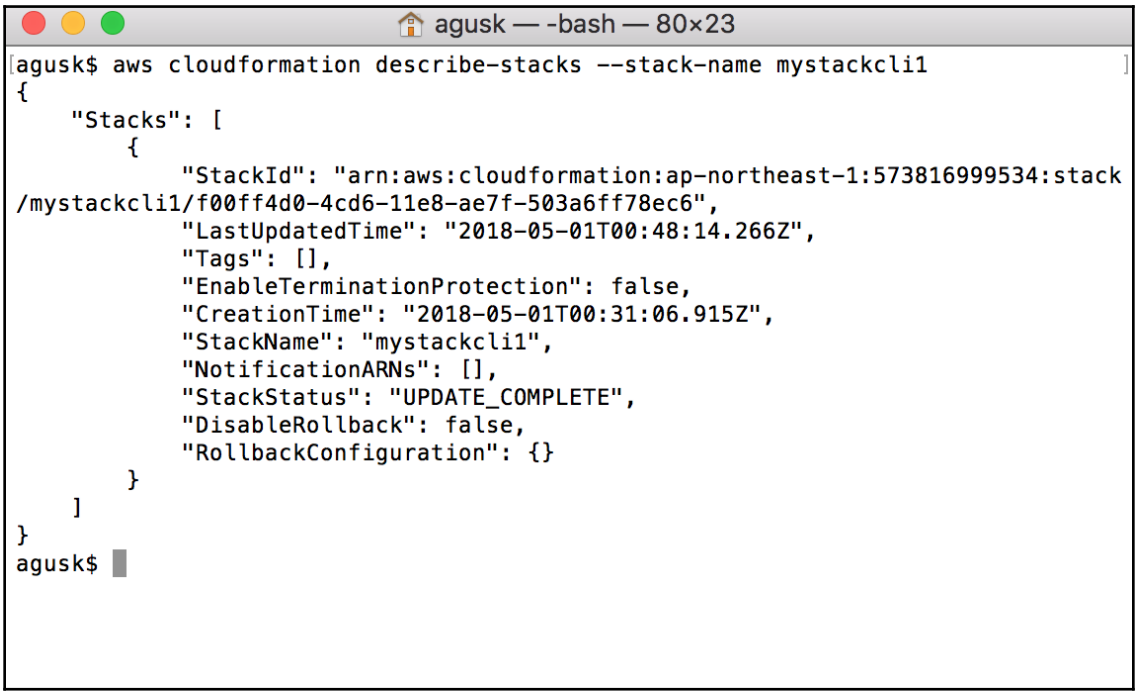


Figure 2.30: Editing a CloudFormation stack from the CLI

If successful, you should see the changes on CloudFormation and the AWS resources that you have modified.

In the next section, we will learn how to delete an existing CloudFormation project.

Deleting a CloudFormation project

If you don't use a CloudFormation stack, you can remove it from CloudFormation. Since you are probably charged for all of the AWS resources that you use, I recommend removing all resources related to the CloudFormation stack. We can also remove a CloudFormation stack because of a design error in IaC. Removing a CloudFormation stack will delete all the resources related to that stack.

In this section, we will learn how to remove a CloudFormation stack using the management console and the AWS CLI.

Deleting CloudFormation using the management console

If you want to delete your CloudFormation stack, you can visit the CloudFormation management console, where you should see your CloudFormation stacks.

To remove a CloudFormation stack, select the stack to be removed. Then, click on **Actions** and you should see the menu shown in the following screenshot. Select the **Delete Stack** option to delete the selected stack:

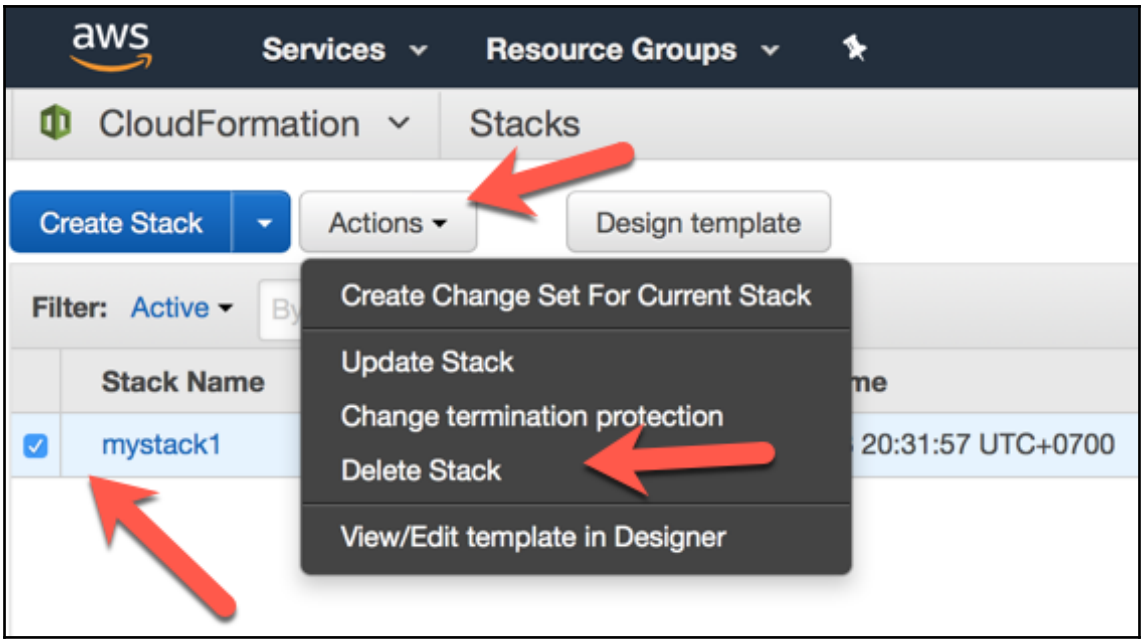


Figure 2.31: Deleting a stack

After selecting the **Delete Stack** option from the menu, you should see a confirmation dialog, as shown in the following screenshot. Click on **Yes, Delete** to continue the deletion of the stack:

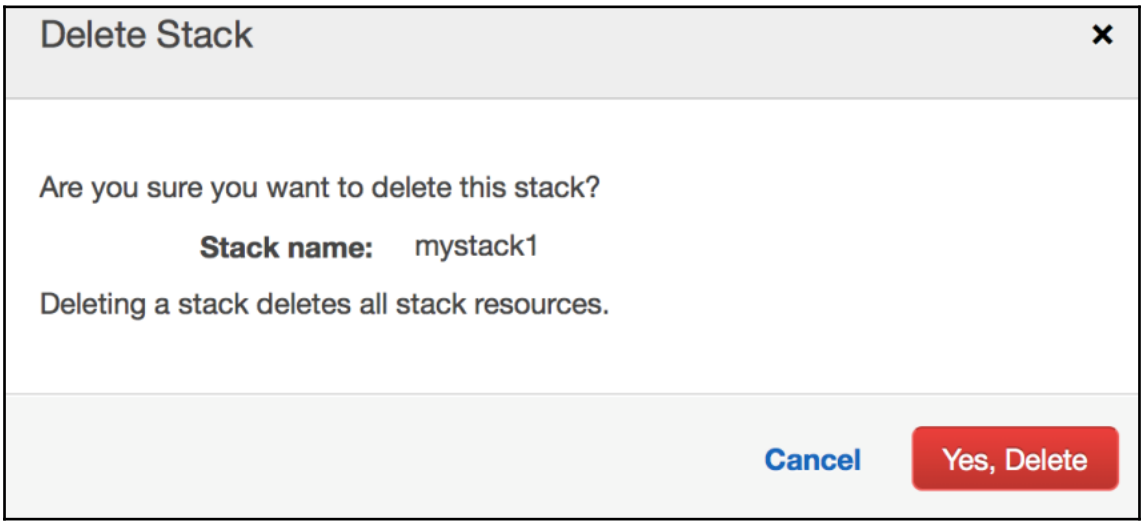


Figure 2.32: Confirmation for deleting a stack

Once done, your CloudFormation stack is deleted, including all the AWS resources that were used in your stack.

Next, we will perform this operation through the AWS CLI.

Deleting CloudFormation Stack using the AWS CLI

If you prefer to delete a CloudFormation stack through the AWS CLI, you can use `delete-stack` from CloudFormation. You can find this command at `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/delete-stack.html`.

To delete a CloudFormation stack, you can use the `delete-stack` command by passing in your stack name. For instance, to delete the CloudFormation stack named `mystackcli1`, type the following command:

```
$ aws cloudformation delete-stack --stack-name mystackcli1
```

If you want to get a verbose message, you can add the `--debug` parameter to the command. My program output can be seen in the following screenshot:

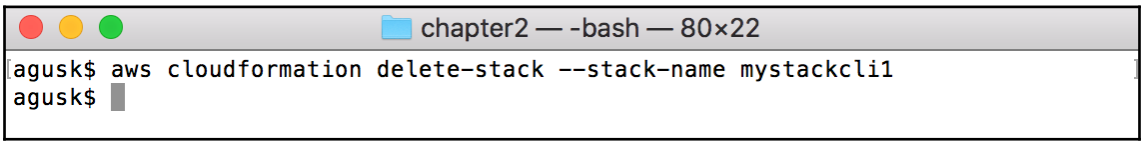


Figure 2.33: Deleting a stack from the CLI

To verify your deleted stack, you can use the `describe-stacks` command by passing in your stack name. For instance, for the stack name `mystackcli1`, type the following command:

```
$ aws cloudformation describe-stacks --stack-name mystackcli1
```

A sample of the program output can be seen in the following screenshot:

Figure 2.34: Checking a stack and the deletion process status from the CLI

We have now reached the end of this chapter, having learned how to build and deploy a CloudFormation project on Amazon AWS.

Summary

We learned how to build a simple CloudFormation project utilizing Amazon S3. We built a CloudFormation project using the CloudFormation management console and the AWS CLI. The demo operations included creating, editing, and deleting.

In the next chapter, we will learn how to build AWS CloudFormation templates.

Questions

1. What is a CloudFormation stack?
2. What are the benefits of building CloudFormation using the management console?
3. What are the advantages of building CloudFormation using the AWS CLI?

3
Developing AWS CloudFormation Templates

Building AWS CloudFormation needs knowledge on how to write IaC for AWS. In this chapter, we'll explore how to develop IaC scripts using JSON and YAML. We'll also review how to access AWS resources from IaC scripts.

The following topics will be covered in this chapter:

- Reviewing the AWS CloudFormation template format
- Reviewing JSON and YAML for AWS CloudFormation
- The programming model for AWS CloudFormation templates
- Writing JSON and YAML for creating AWS CloudFormation templates
- Getting input from the CloudFormation template
- Introduction to AWS CloudFormation Designer
- Giving template description
- Selecting input from options
- Mapping parameters
- Working with intrinsic functions

Reviewing the AWS CloudFormation template format

IaC technology is designed to enable developers to build infrastructure by writing scripts. AWS CloudFormation uses the IaC approach to build infrastructure-based AWS technology stacks. Each AWS resource can be declared in a scripting form. Developers can make their own infrastructures by declaring AWS resources in the AWS CloudFormation template.

To develop AWS CloudFormation, we should understand what the AWS CloudFormation template is and how to build that template. The AWS resource is defined in the AWS service. For instance, Amazon S3 can be defined in the AWS CloudFormation template in JSON format, which is shown as follows:

```
{
 "Type" : "AWS::S3::Bucket",
 "Properties" : {
 "AccessControl" : String,
 "AccelerateConfiguration" : AccelerateConfiguration,
 "AnalyticsConfigurations" : [ AnalyticsConfiguration, ... ],
 "BucketEncryption" : BucketEncryption,
 "BucketName" : String,
 "CorsConfiguration" : CorsConfiguration,
 "InventoryConfigurations" : [ InventoryConfiguration, ... ],
 "LifecycleConfiguration" : LifecycleConfiguration,
 "LoggingConfiguration" : LoggingConfiguration,
 "MetricsConfigurations" : [ MetricsConfiguration, ... ]
 "NotificationConfiguration" : NotificationConfiguration,
 "ReplicationConfiguration" : ReplicationConfiguration,
 "Tags" : [ Resource Tag, ... ],
 "VersioningConfiguration" : VersioningConfiguration,
 "WebsiteConfiguration" : WebsiteConfiguration
 }
}
```

This template can also be declared in the YAML format:

```
Type: "AWS::S3::Bucket"
Properties:
 AccessControl: String
 AccelerateConfiguration:
 AccelerateConfiguration
 AnalyticsConfigurations:
 - AnalyticsConfiguration
 BucketEncryption:
 BucketEncryption
 BucketName: String
 CorsConfiguration:
 CorsConfiguration
 InventoryConfigurations:
 - InventoryConfiguration
 LifecycleConfiguration:
 LifecycleConfiguration
 LoggingConfiguration:
 LoggingConfiguration
 MetricsConfigurations:
 - MetricsConfiguration
```

```
NotificationConfiguration:
NotificationConfiguration
ReplicationConfiguration:
ReplicationConfiguration
Tags:
- Resource Tag
VersioningConfiguration:
VersioningConfiguration
WebsiteConfiguration:
WebsiteConfiguration
```

We can see that each AWS resource can be declared as a resource type. In general, the AWS resource type can be defined as follows:

```
AWS::aws-product-name::data-type-name
```

We should about know AWS resource types if we want to use those resources. A sample of the AWS resource type list can be seen in the following table:

AWS resource name	**AWS resource type**
AWS EC2 instance	`AWS::EC2::Instance`
AWS EC2 VPC	`AWS::EC2::VPC`
AWS IAM user	`AWS::IAM::User`
AWS IoT device thing	`AWS::IoT::Thing`
AWS Lambda function	`AWS::Lambda::Function`
AWS RDS database instance	`AWS::RDS::DBInstance`
AWS S3 bucket	`AWS::S3::Bucket`

Table 3.1: A sample list of AWS resource types

Further information about various AWS resource types can be found at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html`.

An AWS CloudFormation template consists of various AWS resource types, depending on your needs in the infrastructure. In general, the AWS CloudFormation template format can be described in the following diagram. The template consists of various attributes, such as **AWSTemplateFormatVersion**, **Description**, **Metadata**, and **Parameters**:

Figure 3.1: AWS CloudFormation template format

Each AWS CloudFormation template has one or more of the following attributes:

- `AWSTemplateFormatVersion`: This is the AWS CloudFormation template version that is not the same as the API or WSDL version. This attribute is optional.
- `Description`: This defines a text string that describes the template. This attribute is optional.
- `Metadata`: This defines objects that provide additional information about the template. This attribute is optional.

- `Parameters`: This defines values to pass to your template at runtime. You can refer to parameters from the `Resources` and `Outputs` sections of the template. This attribute is optional.
- `Mappings`: This is a mapping of keys and associated values that you can use to specify conditional parameter values. This attribute is optional.
- `Conditions`: This defines conditions that control whether certain resources are created or whether certain resource properties are assigned a value during stack-creation or stack-update. This attribute is optional.
- `Transform`: This helps simplify template authoring by condensing the expression of AWS infrastructure as code and enabling the reuse of template components.
- `Resources`: This specifies the stack resources and their properties, such as EC2 and Amazon S3. This attribute is required.
- `Output`: This describes the values that are returned whenever you view your stack's properties. This attribute is optional.

Since AWS CloudFormation templates are written in JSON and YAML, we will learn how to develop both JSON and YAML. We'll explore these programming methods in the next section.

Reviewing JSON and YAML programming

In this section, we will learn a bit about programming for JSON and YAML. We should know about JSON and YAML in order to build AWS CloudFormation templates. Technically, JSON and YAML are part of a scripting language. We will explore each topic in the next section.

JSON programming

JSON is a data-interchange format. This format is based on ECMA-404. For more information, refer to `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`. JSON can be defined as a key-value model. We can write the following script:

```
{
 key: value
}
```

For instance, we can define a JSON object for the `Employee` data. We can write the following scripts:

```
{
 "id": 1,
 "name": "michael",
 "country": "us"
}
```

You can see that the `Employee` object has three attributes—`id`, `name`, and `country`. Technically, we can declare an attribute in numeric and string types. We can also define a collection of the `Employee` object using `[]`. Each object is defined in `{}`. We can write the scripts as follows:

```
[
 {
  "id": 1,
  "name": "michael",
  "country": "us"
 },
 {
  "id": 2,
  "name": "jason",
  "country": "uk"
 },
 {
  "id": 3,
  "name": "ryu",
  "country": "jp"
 }
]
```

JSON supports unlimited parent-child attributes. For instance, each `Employee` object probably has two or more addresses. We can declare them as follows:

```
{
 "id": 1,
 "name": "michael",
 "address":[
      {
        "street": "abc street",
        "city": "new york"
      },
      {
        "street": "vue street",
        "city": "hoston"
      }
```

```
 ],
 "country": "us"
}
```

This is the end of the JSON programming section. Next, we'll learn how to develop YAML.

YAML programming

YAML is a data-serialization language designed to be directly writable and readable by humans. A simple YAML object can be defined as follows:

```
key:
 value
```

In the JSON format, we use `{}` and `:` for key-value data. In YAML, we only use `:`.

In the previous section, we declared the `Employee` object in JSON. We can convert it to YAML with the following scripts:

```
id:
 1
name:
 "michael"
country:
 "us"
```

If we want to define a collection of YAML objects, we can use the `-` syntax to represent an item of a collection. Let's rewrite the collection of `Employee` from JSON into YAML. We can write the following scripts:

```
- id: 1
  name: "michael"
  country: "us"
- id: 2
  name: "jason"
  country: "uk"
- id: 3
  name: "ryu"
  country: "jp"
```

To declare a parent-child of a YAML object, we can use `:` by passing the YAML object. For instance, we add an `address` object from the `Employee` object. We can write the following scripts to implement it:

```
id:
 1
```

```
name:
 "michael"
address:
    - street: "abc street"
      city: "new york"
    - street: "vue street"
      city: "hoston"
country:
 "us"
```

That's a simple implementation for writing a YAML object. Next, we'll build AWS CloudFormation templates in JSON and YAML.

The programming model for AWS CloudFormation templates

Developing AWS CloudFormation templates means we write scripts for infrastructure-based AWS technology. In this model, we can apply the **Software Development Life Cycle** (**SDLC**). In general, we represent the development model for AWS CloudFormation templates in the following diagram:

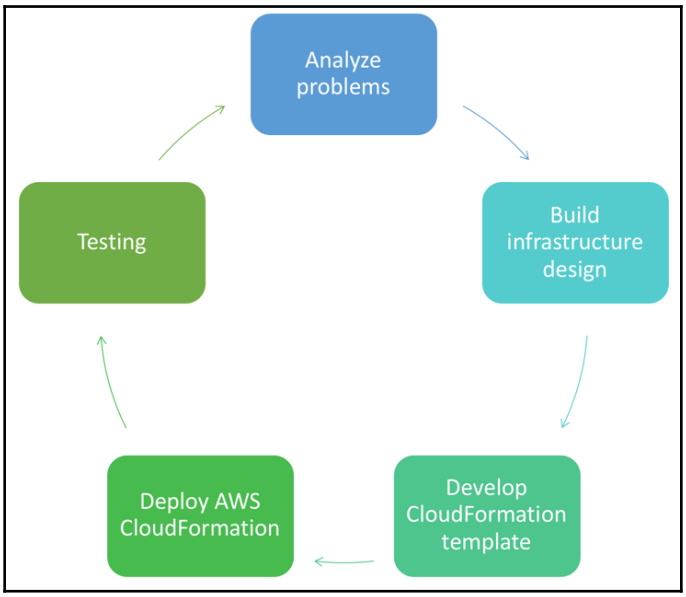


Figure 3.2: CloudFormation template development cycle

From the preceding diagram, we can perform the following tasks:

- **Analyze problems**: We identity what the problems are and decide which problems to fix.
- **Build an infrastructure design**: Based on your problem and solution, we define an infrastructure design, including some AWS resources to implement.
- **Develop a CloudFormation template**: After you have built the infrastructure design, you write the CloudFormation template. You define all AWS resources on your scripts. You can write the template using JSON and YAML files or with CloudFormation Designer.
- **Deploy AWS CloudFormation**: After you complete the CloudFormation template, you can upload and provision those scripts.
- **Testing**: You can test your infrastructure. If you find a problem in testing, you can modify your scripts and then perform testing again.

In term of SDLC, you can use an agile methodology while developing CloudFormation templates. There are a lot of agile methodology models, such as scrum and **Extreme Programming** (**XP**). You can also use source control to manage CloudFormation template files. Next, we'll write scripts for CloudFormation.

Writing JSON and YAML to create AWS CloudFormation templates

We have learned the CloudFormation template format that is described in *Figure 3.1*. CloudFormation templates can be implemented in JSON and YAML. From *Figure 3.1*, we can implement the CloudFormation template in JSON format as follows:

```
{
 "AWSTemplateFormatVersion" : "version date",
 "Description" : "JSON string",
 "Metadata" : {
   template metadata
 },
 "Parameters" : {
   set of parameters
 },
 "Mappings" : {
   set of mappings
 },
 "Conditions" : {
   set of conditions
```

```
 },
 "Transform" : {
   set of transforms
 },
 "Resources" : {
   set of resources
 },
 "Outputs" : {
   set of outputs
 }
}
```

All attributes in CloudFormation are not required. Only the `Resources` attribute should be defined on your CloudFormation template. You can also define the CloudFormation template in YAML. We can declare the CloudFormation template from *Figure 3.1* in YAML as follows:

```
AWSTemplateFormatVersion: "version date"
Description:
 String
Metadata:
 template metadata
Parameters:
 set of parameters
Mappings:
 set of mappings
Conditions:
 set of conditions
Transform:
 set of transforms
Resources:
 set of resources
Outputs:
 set of outputs
```

Some attributes from CloudFormation templates will be explored in this book with various use cases and scenarios. In the next section, we'll build a CloudFormation template to get input from users.

Getting input from the CloudFormation template

In `Chapter 2`, *Building Your First AWS CloudFormation Project*, we built CloudFormation while applying Amazon S3 resources. Since we didn't specify the S3 bucket name, Amazon S3 will generate a random name for our bucket. In this section, we'll build CloudFormation, which gets input from users to set the Amazon S3 bucket name.

We can use the `Parameters` attribute from the CloudFormation template. The value of the `Parameters` attribute will be passed to our Amazon AWS resources. To get `Parameters` from resources, you can use `Ref` by passing the parameter name. For instance, I have a parameter `YourBucketName`. This parameter passes to my resource `MySimpleBucket`, on the `BucketName` attribute, shown as follows:

```
{
  "Parameters": {
    "YourBucketName":{
      "Description": "Amazon S3 Bucket name",
      "Type": "String"
    }
 },
 "Resources" : {
    "MySimpleBucket" : {
      "Type" : "AWS::S3::Bucket",
      "Properties" : {
        "AccessControl" : "PublicRead",
        "BucketName" : {
           "Ref": "YourBucketName"
        }
      }
    }
  }
}
```

For demonstration purposes, we will create Amazon S3 with a custom bucket name that is filled in by the user. You can perform the following steps to implement the demo:

1. Prepare the CloudFormation template on your local computer.
2. Create a JSON or YAML file to build the CloudFormation template.
3. Give a template file for JSON format as `simple-s3.json` and the `simple-s3.yaml` file in YAML format. You can write the following scripts.

The script for the `simple-s3.json` file is as follows:

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Amazon S3 with custom Bucket name",
    "Parameters": {
      "YourBucketName":{
          "Description": "Amazon S3 Bucket name",
          "Type": "String"
      }
    },
    "Resources" : {
        "MySimpleBucket" : {
            "Type" : "AWS::S3::Bucket",
            "Properties" : {
                "AccessControl" : "PublicRead",
                "BucketName" : {
                    "Ref": "YourBucketName"
                  }
            }
        }
    }
}
```

The script for the `simple-s3.yaml` file is as follows:

```
AWSTemplateFormatVersion:
 "2010-09-09"
Description:
 "Amazon S3 with custom Bucket name"
Parameters:
 YourBucketName:
   Description:
     "Amazon S3 Bucket name"
   Type:
     "String"
Resources:
 MySimpleBucket:
   Type:
     "AWS::S3::Bucket"
   Properties:
     AccessControl:
        "PublicRead"
   BucketName:
     Ref:
        "YourBucketName"
```

4. Once done, save these scripts into a file.
5. Upload the template file to AWS CloudFormation.
6. Navigate to `https://console.aws.amazon.com/cloudformation` and click on the **Create New Stack** button. You should see a screen, which is shown in the following screenshot.
7. Select the **Upload a template to Amazon S3** option.
8. Click on the **Choose File** button to upload the CloudFormation template file and then click on the **Next** button:

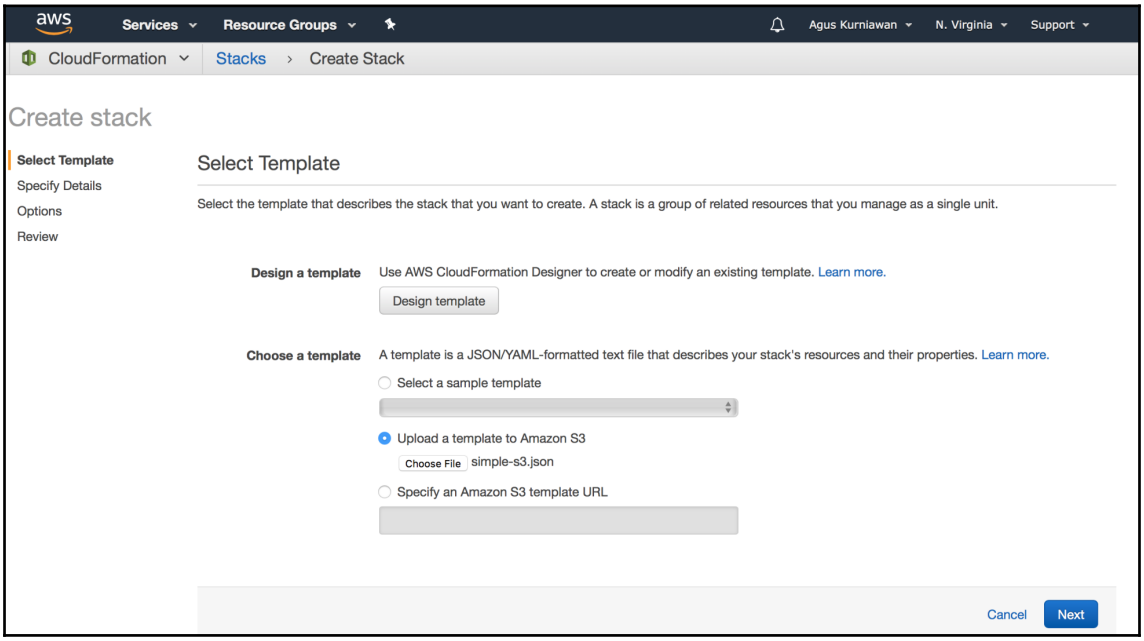


Figure 3.3: Selecting the CloudFormation template

9. You should see a screen, which is shown in *Figure 3.4*. You should use our defined parameter called `YourBucketName`. This is our S3 bucket name.
10. Fill in the **Stack name** and **YourBucketName** fields for the bucket name:

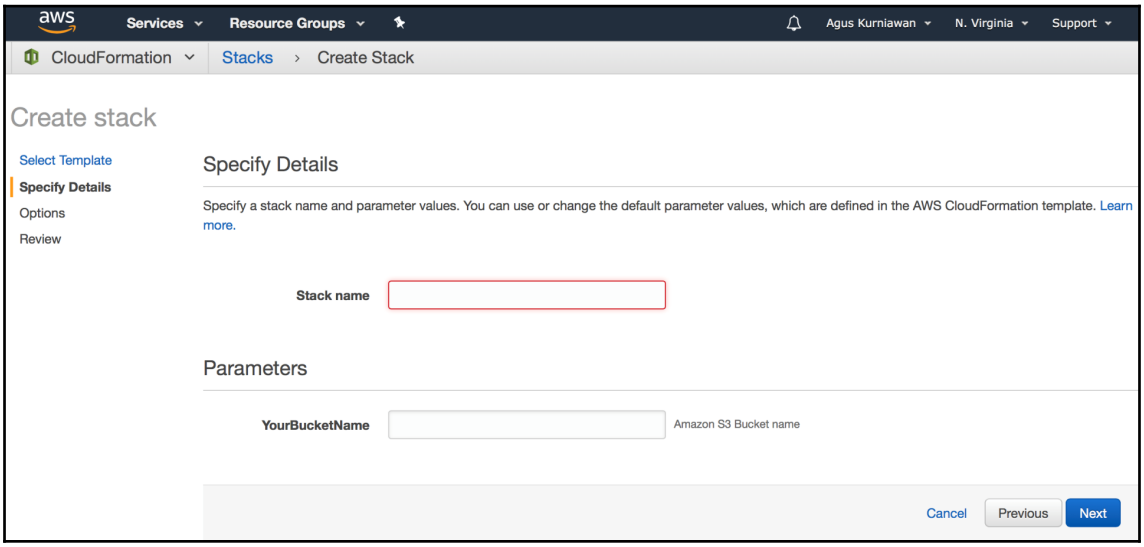


Figure 3.4: Displaying parameters for CloudFormation templates

11. For instance, I filled in my stack name as `my-simple-stack` and my bucket name as `my-simple-s3`. You can see my input in the following screenshot.
12. Once done, you can click on the **Next** button, as shown in the following screenshot:

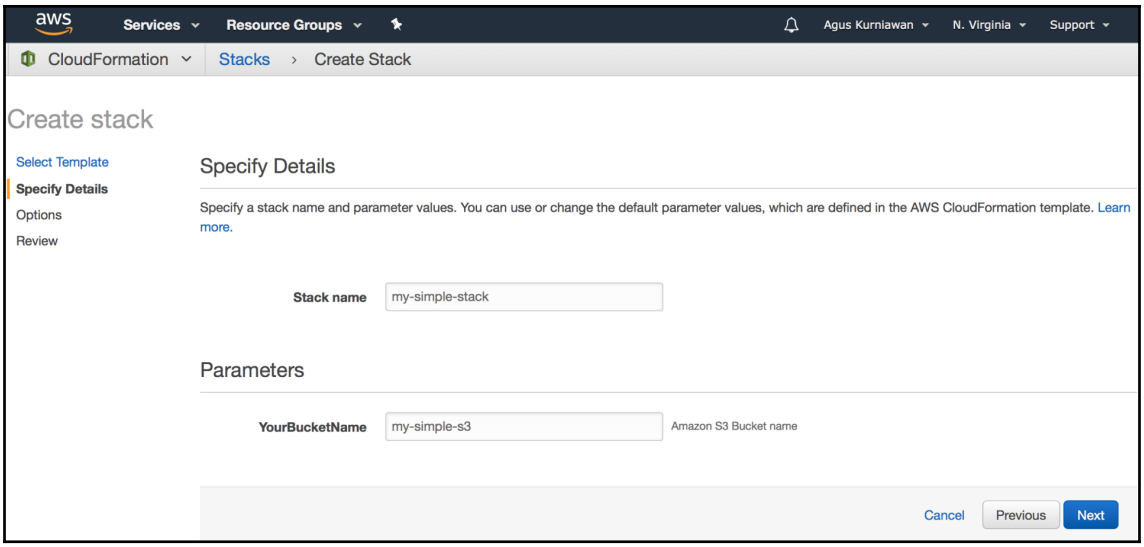


Figure 3.5: Filling in the CloudFormation parameters

13. After you have clicked the **Next** button, you should be asked to fill in the options. In this case, we don't fill in any items. Just click on the **Next** button:

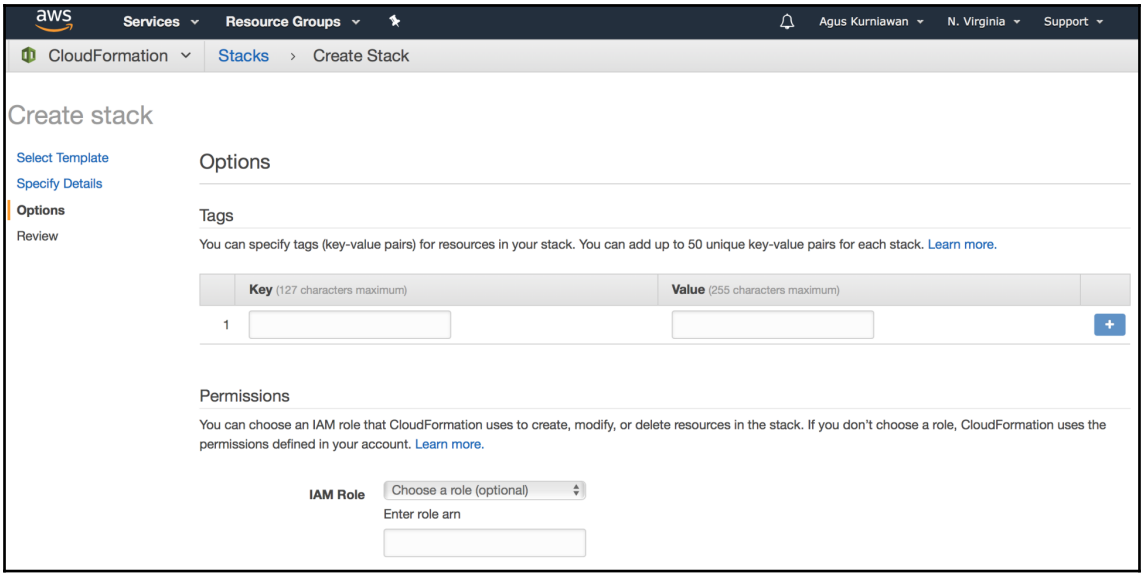


Figure 3.6: Setting the CloudFormation options

14. You should have confirmation, as shown in the following screenshot, before provisioning your CloudFormation template.
15. If you are done, you can click on the **Create** button to deploy CloudFormation:

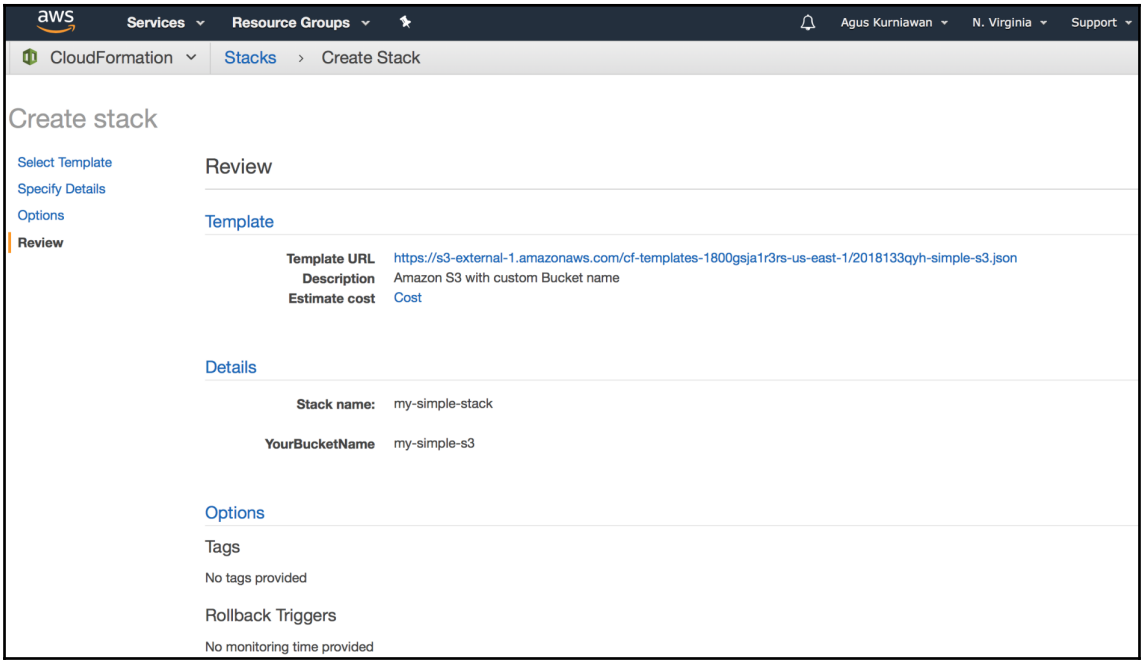


Figure 3.7: Confirmation for CloudFormation

16. AWS CloudFormation will create all the resources that are defined in the template.
17. You should see the provision status on the dashboard, which is shown in the following screenshot:

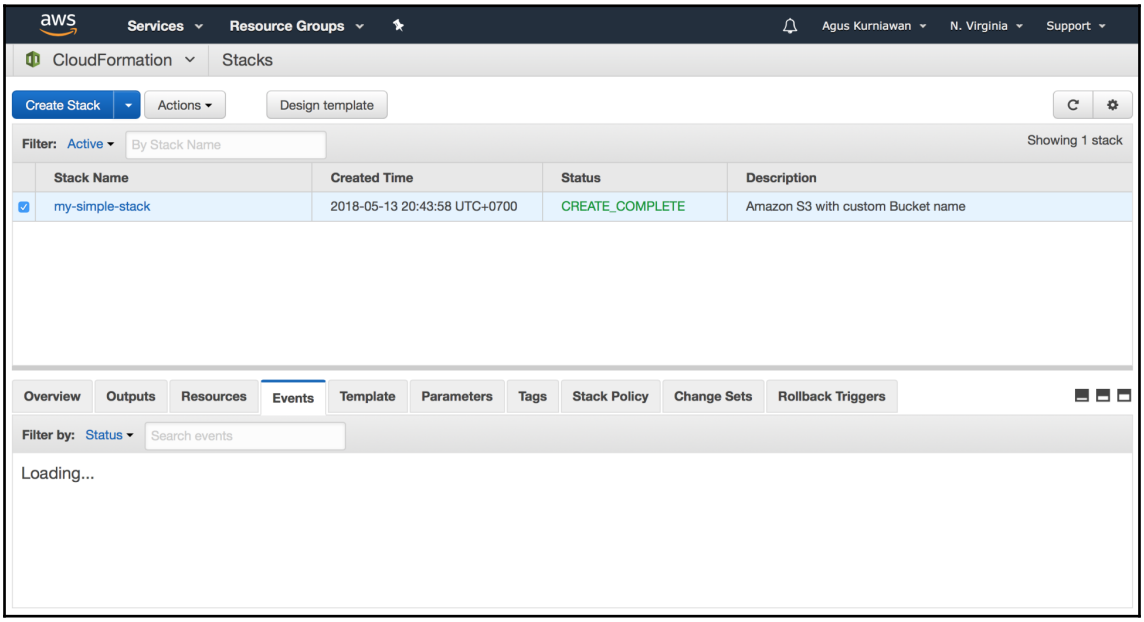


Figure 3.8: Deploying CloudFormation

18. You can verify the Amazon S3 console to ensure your resource has been created by CloudFormation.
19. Check out the following screenshot, my Amazon S3 with my bucket name, `my-simple-s3`. This bucket name is filled in from our input from *Figure 3.5*:

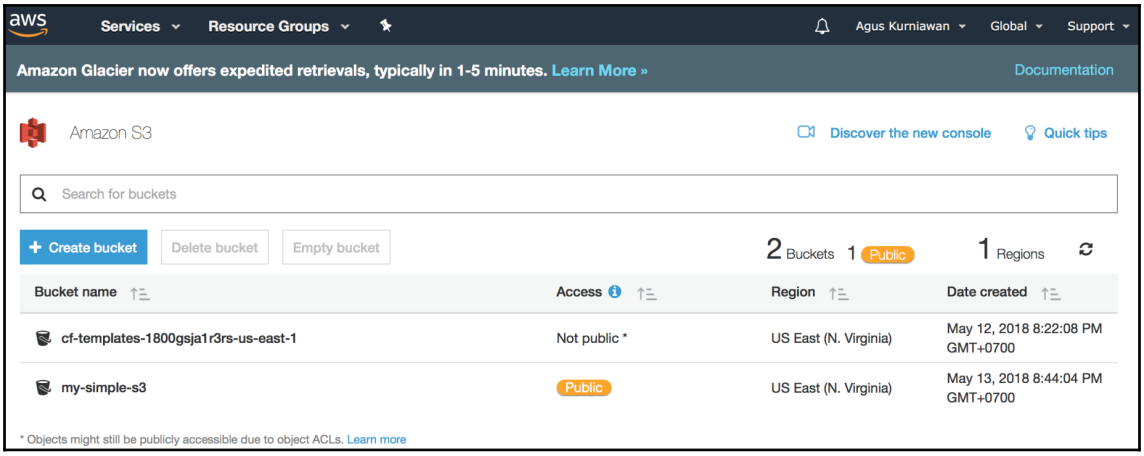


Figure 3.9: Amazon S3 is created from the CloudFormation template

If you prefer to run the CloudFormation template using the AWS CLI, you can do that easily. For instance, the template file is located on `home/user/templates/simple-s3.json`. We also set the stack name as `my-simple-stack` and pass parameters for `YourBucketName` with `my-simple-s3`. You can type the following command:

```
$ aws cloudformation create-stack --stack-name my-simple-stack
 --template-body file://home/user/templates/simple-s3.json
 --parameters YourBucketName=my-simple-s3
```

The AWS CLI will deploy your CloudFormation based on its template file.

This is the end of our demo. We have created a resource, Amazon S3, and passed a parameter to define a bucket name. You can practice more with various parameters from the Amazon AWS resources.

Next, we will build CloudFormation Designer.

Introducing AWS CloudFormation Designer

We have learned how to create a CloudFormation template manually by writing scripts, in JSON or YAML, with a text editor. In this section, we'll build a CloudFormation template graphically using CloudFormation Designer. You can click and drag AWS resources into the template. This tool will generate JSON or YAML files. Then, you can upload the template to AWS CloudFormation.

For a demo, we'll build a CloudFormation template using CloudFormation Designer. We'll use Amazon S3 to show how CloudFormation Designer works. You can perform the following steps for the demo:

1. Open your browser and navigate to the AWS CloudFormation console at `https://console.aws.amazon.com/cloudformation`. You should see the AWS CloudFormation console dashboard.
2. To work with CloudFormation Designer, click on the **Design template** button at the top of the dashboard. You should have the following CloudFormation Designer form:

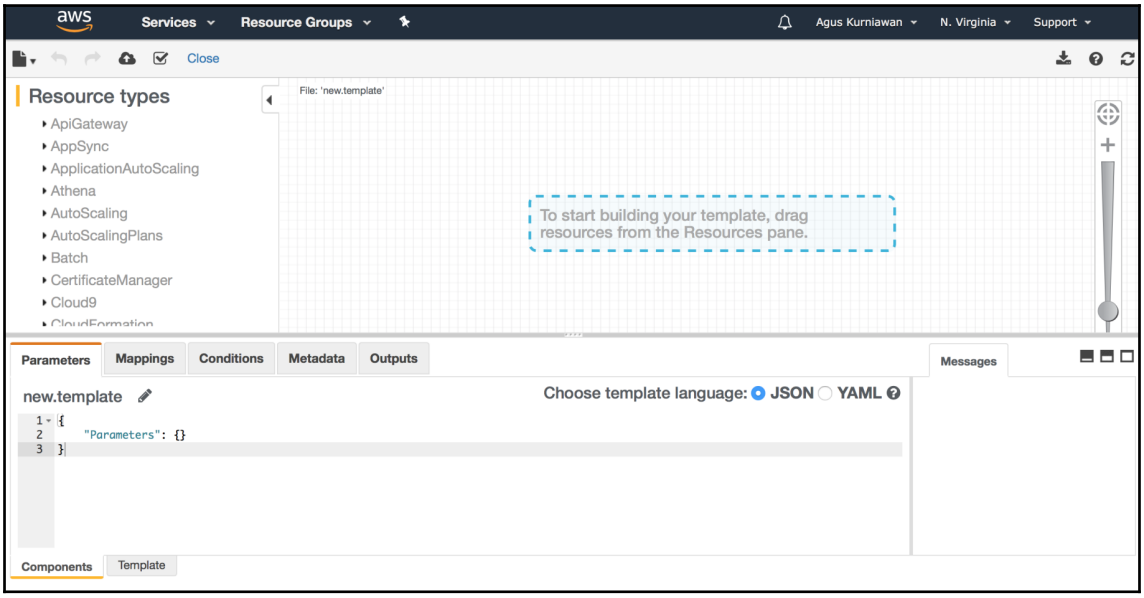


Figure 3.10: A form for CloudFormation designer

3. You should see a list of AWS resource types on the left (see *Figure 3.10*). Find the S3 bucket on the list and drag it onto the right panel, as shown in the following screenshot:

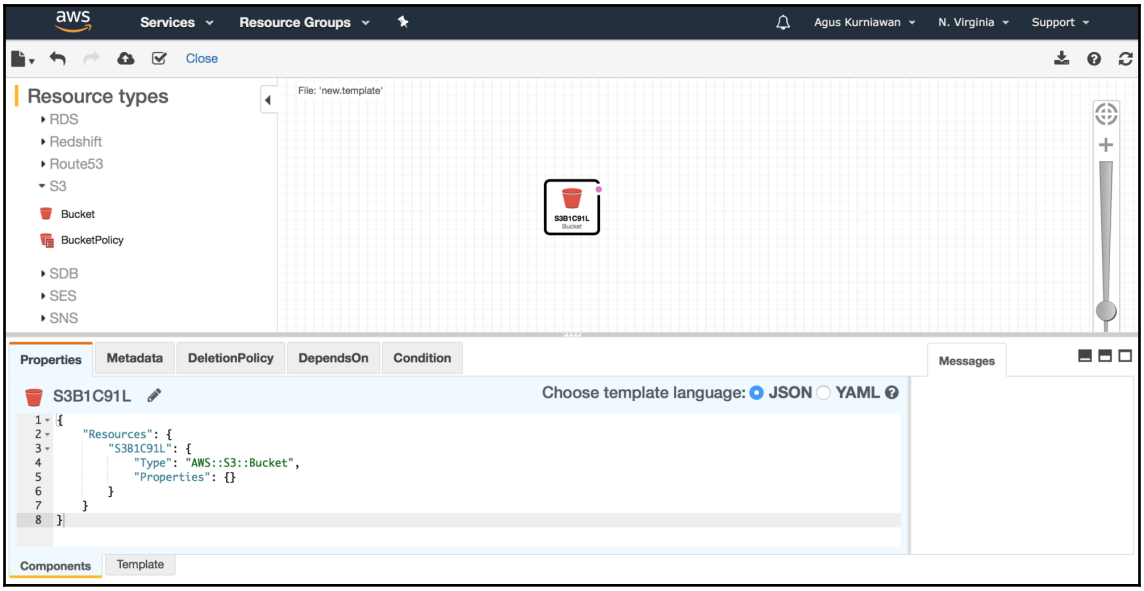


Figure 3.11: Adding Amazon S3 into designer

4. After dropping the AWS resource type into the right panel, there will be skeleton scripts on bottom.

5. Clicking on the **Properties** tab, you should see JSON or YAML scripts which are generated from the CloudFormation Designer.
6. Upload the template to Amazon S3 by clicking the cloud with the up-arrow icon. You should get a dialog box, which is shown in *Figure 3.12*.
7. Click the Amazon S3 bucket tab, fill in the template name, such as `mysimples3.template`, and click on the **Save** button:

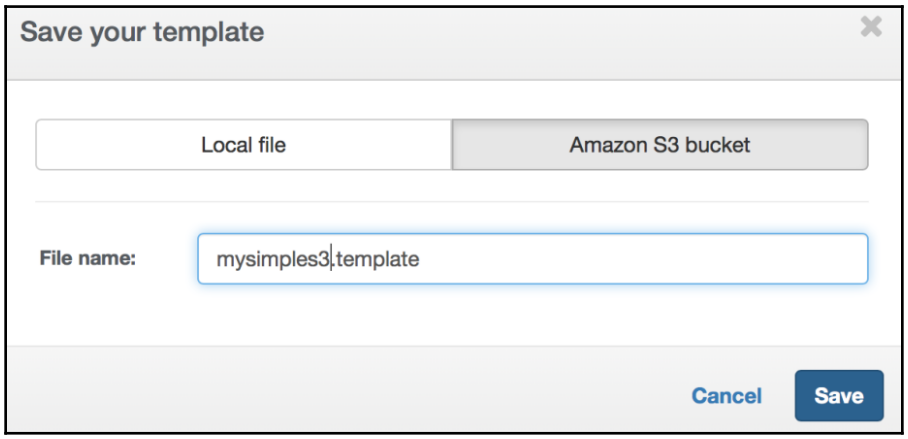


Figure 3.12: Saving the template to the Amazon S3 bucket

8. You should get a dashboard for creating a stack. Since we have uploaded the template to Amazon S3, the dashboard has selected the URL on Amazon S3, as shown in the following screenshot. Click on the **Next** button:

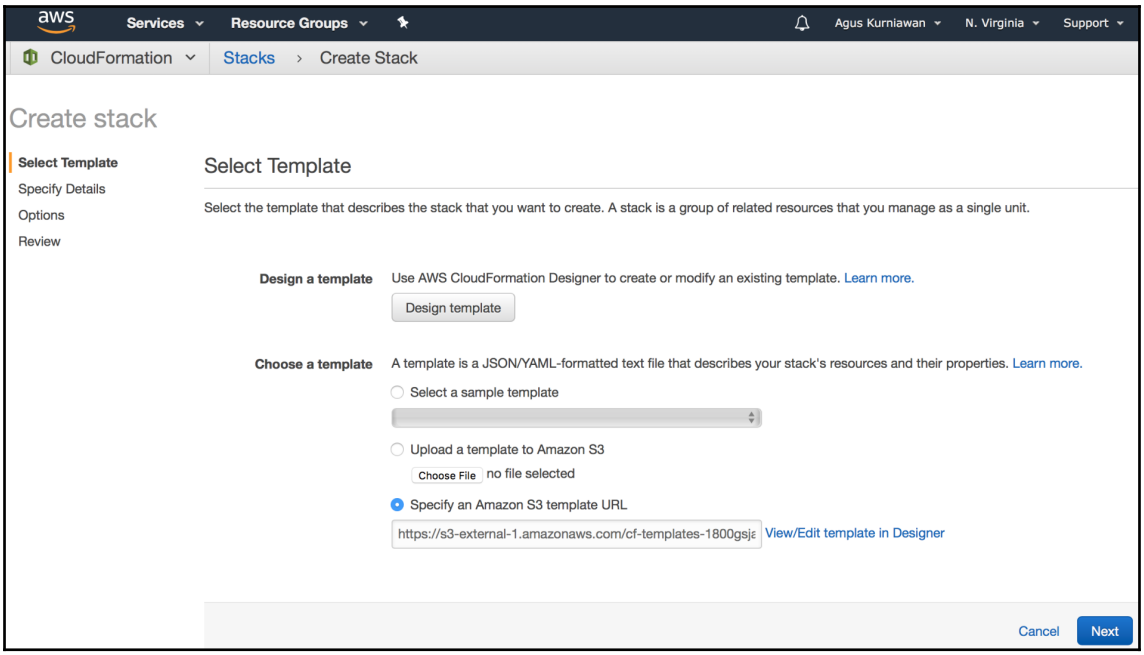


Figure 3.13: Uploading a designer to create a stack

9. Follow the instructions until you review configurations, as shown in the following screenshot:

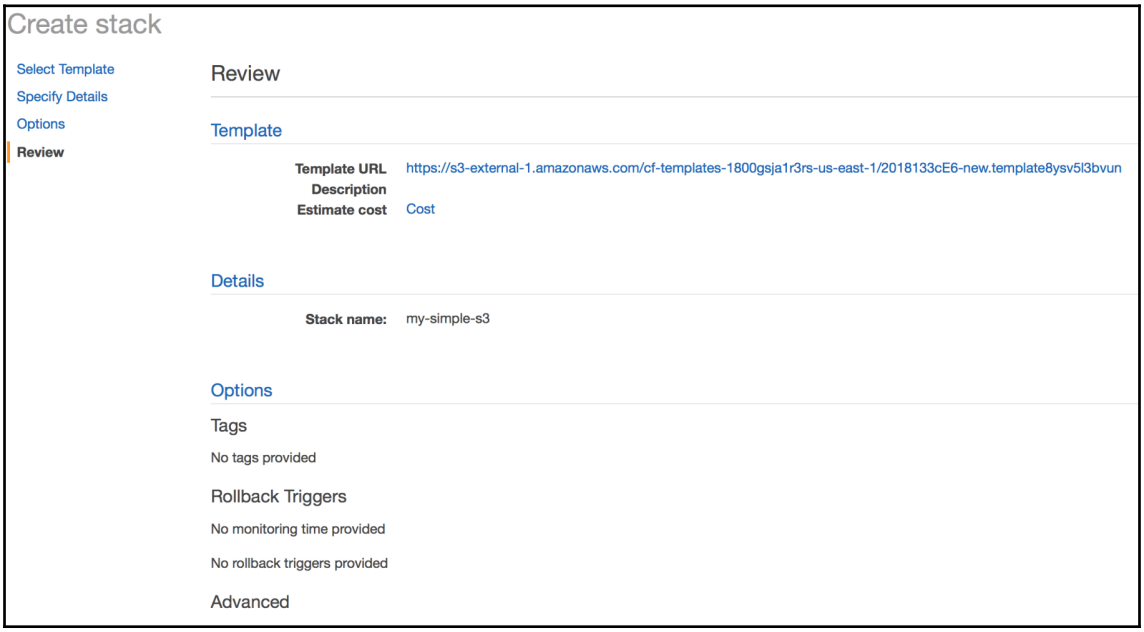


Figure 3.14: Confirmation for creating CloudFormation

10. AWS CloudFormation generates Amazon S3 based on your template. You can verify it by checking it on the Amazon S3 console:

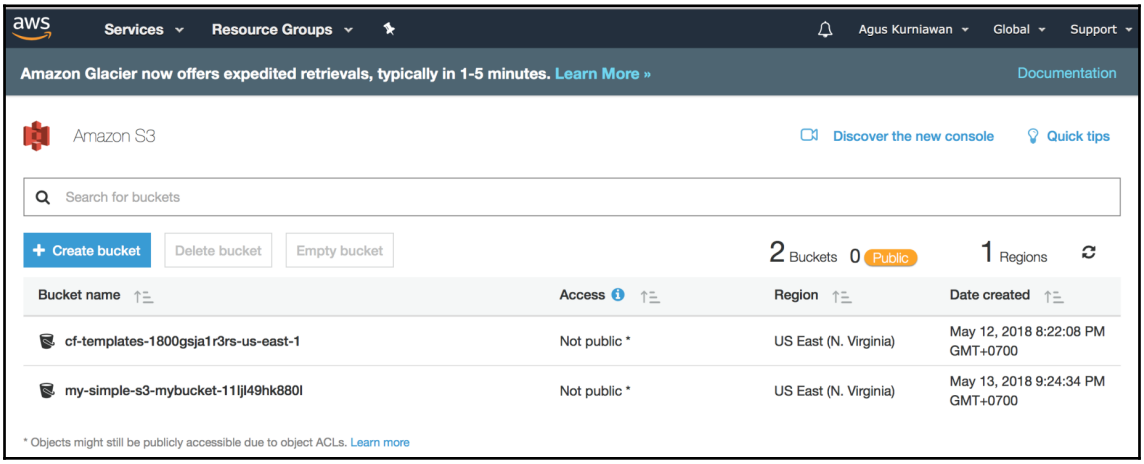


Figure 3.15: Amazon S3 is created from CloudFormation designer

We have finished creating the CloudFormation template and deploying it to AWS CloudFormation. You can now perform other experiments with various resource types.

Next, we'll work with template descriptions.

Giving a template description

The CloudFormation template provides the `Description` attribute to display information about your template. You must set a description as a literal string that is between 0 and 1,024 bytes in length. You can see our sample implementation for the `Description` attribute. A sample program for JSON is as follows:

```
"AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Amazon S3 with custom Bucket name",
    "Parameters": {
      "YourBucketName":{
          "Description": "Amazon S3 Bucket name",
          "Type": "String"
      }
    }
```

For YAML, you can implement it with the following scripts:

```
AWSTemplateFormatVersion:
  "2010-09-09"
Description:
  "Amazon S3 with custom Bucket name"
Parameters:
  YourBucketName:
    Description:
      "Amazon S3 Bucket name"
    Type:
```

This attribute is optional. When you apply words on the `Description` attribute on the template, you can see it when you review the template. See *Figure 3.7* for more information.

Next, we'll learn how to select a value from various options in CloudFormation.

Selecting the input from options

Sometimes, when you build an infrastructure in CloudFormation, you need input from users. In the previous section, we learned how to get input from users by filling in the Amazon S3 bucket name. Now, we want to learn how to select input from various options.

The scenario can be illustrated to get an EC2 instance type size. Users can select the instance types that we defined on the `Parameters` section. You can see the following scripts for implementation in JSON:

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "Amazon EC2 instance with Amazon Linux AMI.",
 "Parameters": {
    "InstanceType": {
      "Description": "EC2 instance type",
      "Type": "String",
      "Default": "t1.micro",
      "AllowedValues": [
        "t1.micro",
        "t2.nano",
        "t2.micro",
        "t2.small"
      ],
      "ConstraintDescription": "must be a valid EC2 instance type."
    }
 }
```

For YAML, you can use these scripts:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Amazon EC2 instance with Amazon Linux AMI.
Parameters:
  InstanceType:
  Description: EC2 instance type
  Type: String
  Default: t1.micro
  AllowedValues:
    - t1.micro
    - t2.nano
    - t2.micro
    - t2.small
  ConstraintDescription: must be a valid EC2 instance type.
```

You can see that we defined four instance types, which are defined in the `AllowedValues` attribute. Users can select one of the instance types. The following is a list of options:

- `t1.micro`
- `t2.nano`
- `t2.micro`
- `t2.small`

There are a lot of EC2 instance types. In this demo, I set only four types. You can find a list of types supported for EC2 instance here at `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html`. You can put all instance types if you need them.

We can also set a default value for our options. We can use the `Default` attribute and set its value as the default value. For instance, I set `t1.micro` as the default value.

Now, we can set this selection from the user on the `InstanceType` attribute from `EC2Instance`. The following is a sample program in JSON:

```
"Resources": {
   "EC2Instance": {
     "Type": "AWS::EC2::Instance",
     "Properties": {
        "InstanceType": {
           "Ref": "InstanceType"
        },
```

In the YAML format, do the following:

```
Resources:
 EC2Instance:
   Type: AWS::EC2::Instance
   Properties:
     InstanceType:
       Ref: InstanceType
```

Now, if you deploy this template file (`simple-ec2.json` or `simple-ec2.yaml`) to CloudFormation, you should see a selection option on the `Parameters` section, as shown in the following screenshot:

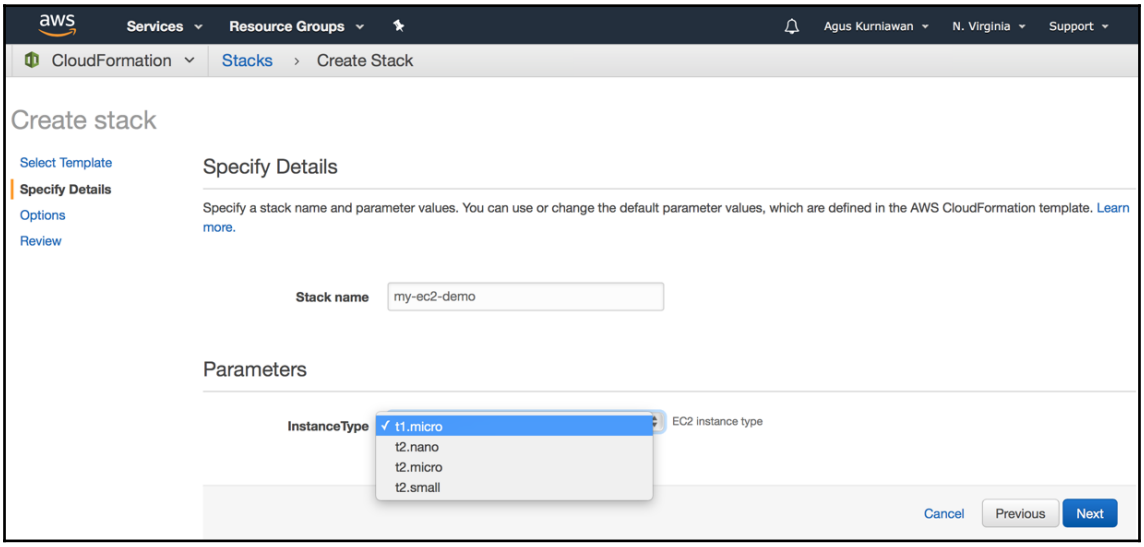


Figure 3.16: Selecting instance type for EC2

This is a simple sample to get input from some options. You can practice more by exploring some AWS resources.

Next, we'll learn how to use the `Mappings` attribute on CloudFormation.

Mapping parameters

Some AWS resources need specific attribute types that probably do not become accommodated from a user's input. In this case, we can utilize the `Mappings` attribute on CloudFormation. For instance, we can map the EC2 instance type based on region. We can declare this on the `Mapping` attribute as follows:

```
"Mappings" : {
   "RegionMap" : {
     "us-east-1" : { "32" : "ami-6411e20d"},
     "us-west-1" : { "32" : "ami-c9c7978c"},
     "eu-west-1" : { "32" : "ami-37c2f643"},
     "ap-southeast-1" : { "32" : "ami-66f28c34"},
     "ap-northeast-1" : { "32" : "ami-9c03a89d"}
   }
}
```

Here is the preceding code in YAML format:

```
Mappings
 RegionMap
  us-east-1
    32: ami-6411e20d
  us-west-1
    32 : ami-c9c7978c
  eu-west-1
    32 : ami-37c2f643
  ap-southeast-1
    32 : ami-66f28c34
  ap-northeast-1
    32 : ami-9c03a89d
```

As you can see, `RegionMap` will map various values based on region input, such as `us-east-1`, `us-west-1`, and `eu-west-1`. If we choose `eu-west-1`, we will obtain `ami-37c2f643` for the `32` architecture.

In a real program, we usually use the `Mappings` attribute with the `Fn::FindInMap` intrinsic function that's available at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-findinmap.html`. This function returns the value corresponding to keys in a two-level map that is declared in the `Mappings` section. `Fn::FindInMap` can be defined as follows:

Here it is in JSON:

```
{ "Fn::FindInMap" : [ "MapName", "TopLevelKey", "SecondLevelKey"] }
```

And here it is in YAML:

```
Fn::FindInMap: [ MapName, TopLevelKey, SecondLevelKey ]
```

For instance, we want to set the `ImageId` attribute from `EC2Instance`. We can look up `AWSRegionArch2AMI` and `AWSInstanceType2Arch` from the `Mappings` attribute. You can see a sample implementation of `Fn::FindInMap` in the following JSON scripts:

```
    "Mappings": {
     "AWSInstanceType2Arch": {
       "t1.micro": {
         "Arch": "PV64"
       },
       "t2.nano": {
         "Arch": "HVM64"
       },
       "t2.micro": {
         "Arch": "HVM64"
       },
       "t2.small": {
         "Arch": "HVM64"
       }
     },
      "AWSRegionArch2AMI": {
       "us-east-1": {
         "PV64": "ami-2a69aa47",
         "HVM64": "ami-6869aa05",
         "HVMG2": "ami-61e27177"
       },
       "us-west-2": {
         "PV64": "ami-7f77b31f",
         "HVM64": "ami-7172b611",
         "HVMG2": "ami-60aa3700"
       },
       "us-west-1": {
         "PV64": "ami-a2490dc2",
         "HVM64": "ami-31490d51",
         "HVMG2": "ami-4b694d2b"
       }
     }
    },
    "Resources": {
      "EC2Instance": {
        "Type": "AWS::EC2::Instance",
```

```
        "Properties": {
          "InstanceType": {
            "Ref": "InstanceType"
          },
          "ImageId": {
            "Fn::FindInMap": [
              "AWSRegionArch2AMI",
              {
                "Ref": "AWS::Region"
              },
              {
                "Fn::FindInMap": [
                  "AWSInstanceType2Arch",
                  {
                    "Ref": "InstanceType"
                  },
                  "Arch"
                ]
              }
            ]
          }
        }
      }
```

The following script is for YAML:

```
Mappings:
  AWSInstanceType2Arch:
    t1.micro:
      Arch: PV64
    t2.nano:
      Arch: HVM64
    t2.micro:
      Arch: HVM64
    t2.small:
      Arch: HVM64
  AWSInstanceType2NATArch:
    t1.micro:
      Arch: NATPV64
    t2.nano:
      Arch: NATHVM64
    t2.micro:
      Arch: NATHVM64
    t2.small:
      Arch: NATHVM64
  AWSRegionArch2AMI:
    us-east-1:
```

```
        PV64: ami-2a69aa47
        HVM64: ami-6869aa05
        HVMG2: ami-61e27177
      us-west-2:
        PV64: ami-7f77b31f
        HVM64: ami-7172b611
        HVMG2: ami-60aa3700
      us-west-1:
        PV64: ami-a2490dc2
        HVM64: ami-31490d51
        HVMG2: ami-4b694d2b
  Resources:
    EC2Instance:
      Type: AWS::EC2::Instance
      Properties:
        InstanceType:
          Ref: InstanceType
        ImageId:
          Fn::FindInMap:
          - AWSRegionArch2AMI
          - Ref: AWS::Region
          - Fn::FindInMap:
            - AWSInstanceType2Arch
            - Ref: InstanceType
            - Arch
```

Keep practicing with the `Mappings` attribute in CloudFormation. Next, we'll learn how to use intrinsic functions on the CloudFormation template.

Working with intrinsic functions

Intrinsic functions enable us to put logic functions inside CloudFormation templates. Technically, we have used intrinsic functions, for example `Ref` and `Fn::FindInMap`, in previous CloudFormation templates.

In general, Amazon AWS provides some intrinsic functions that we can apply to CloudFormation templates. The following is a list of intrinsic functions:

- `Fn::Base64`
- `Fn::Cidr`
- `Condition Functions`
- `Fn::FindInMap`
- `Fn::GetAtt`

- `Fn::GetAZs`
- `Fn::ImportValue`
- `Fn::Join`
- `Fn::Select`
- `Fn::Split`
- `Fn::Sub`
- `Ref`

The various intrinsic functions are explained as follows:

- The `Fn::Base64` intrinsic function returns the Base64 representation of the input string. `Fn::Base64` is defined as follows.

  Here it is in JSON:

  ```
  { "Fn::Base64" : valueToEncode }
  ```

  And here it is in YAML:

  ```
  Fn::Base64: valueToEncode
  ```

- The `Fn::Cidr` intrinsic function returns the specified CIDR address block. You can declare this function as follows.

  Here it is in JSON:

  ```
  { "Fn::Cidr" : [ipBlock, count, sizeMask]}
  ```

  And here it is in YAML:

  ```
  Fn::Cidr: [ ipBlock, count, sizeMask ]
  ```

- Conditional functions apply logical conditional on a CloudFormation template. We can use the following conditional functions:
  - `Fn::And`
  - `Fn::Equals`
  - `Fn::If`
  - `Fn::Not`
  - `Fn::Or`

For example, we want to decide the volume size of EC2. We check `CreateLargeSize` if this is true, and we set the volume size as `100`. Otherwise, we set the volume size as `10`. The following is a sample program for JSON:

```
"NewVolume" : {
  "Type" : "AWS::EC2::Volume",
  "Properties" : {
    "Size" : {
      "Fn::If" : [
        "CreateLargeSize",
        "100",
        "10"
      ]},
  }
}
```

Here it is in YAML:

```
NewVolume:
  Type: "AWS::EC2::Volume"
  Properties:
    Size:
      !If [CreateLargeSize, 100, 10]
```

- `Fn::FindInMap` returns the value corresponding to keys in a two-level map that is declared in the `Mappings` section. We have also learned how to use this function.
- The `Fn::GetAtt` intrinsic function returns the value of an attribute from a resource in the template. We define `Fn::GetAtt` in JSON as follows:

```
{ "Fn::GetAtt" : [ "logicalNameOfResource", "attributeName" ] }
```

Here it is in the YAML format:

```
Fn::GetAtt: [ logicalNameOfResource, attributeName ]
```

- `Fn::GetAZs` returns an array that lists the **Availability Zones** (**AZs**) for a specified region. This function is useful when you are working with multiple regions.
- The `Fn::Join` intrinsic function appends a set of values into a single value, separated by the specified delimiter.

- The `Fn::Select` function returns a single object from a list of objects by index. The `Fn::Select` function is used to split a string into a list of string values.
- `Fn::Sub` substitutes variables in an input string with values that you specify.

For more information about intrinsic functions, check out `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html`.

Working with Metadata on the CloudFormation template

The CloudFormation template offers an optional attribute, `Metadata`, that provides details about the template. For example, you can include template implementation details about specific resources, such as the `AWS::ElasticLoadBalancing::LoadBalancer` resource. You can write the scripts as follows:

```
  "Resources" : {
    "ElasticLoadBalancer" : {
      "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
      "Metadata" : {
        "Comment1" : "Configure the Load Balancer with a simple health
check and cookie-based stickiness",
        "Comment2" : "Use install path for healthcheck to avoid redirects -
ELB healthcheck does not handle 302 return codes"
      },
```

You cannot change the `Metadata` attribute once you update the CloudFormation template.

CloudFormation resources

The `Resources` attribute is one of the required attributes in the CloudFormation template. You can declare one or more AWS resources in your template. For instance, you can use the Amazon EC2 instance and an Amazon S3 bucket in one template. In general, the `Resources` attribute in CloudFormation can be defined as follows:

```
"Resources" : {
    "Logical ID" : {
        "Type" : "Resource type",
        "Properties" : {
```

```
            Set of properties
        }
    }
}
```

Here it is in YAML:

```
Resources:
  Logical ID:
    Type: Resource type
    Properties:
      Set of properties
```

`Logical ID` is a unique ID for AWS resources. You can declare it in alphanumeric form (A-Z a-z 0-9). The `Type` attribute is defined for AWS resource types. You can check a list of resource types that you can use in the template. Check them out at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html`.

Each resource type has properties. You probably want to set the resource properties on the `Properties` attribute. You should match between resource type and resource type properties.

CloudFormation output

If we want to get responses from AWS resources after they're deployed, we can import those responses into your CloudFormation. We can use the `Outputs` attribute. It will return in response (to describe stack calls), or view on the AWS CloudFormation console.

The `Outputs` section consists of the key name `Outputs`, followed by a space and a single colon. You can declare a maximum of 60 output in a template.

We can declare the `Outputs` attribute in JSON as follows:

```
"Outputs" : {
  "Logical ID" : {
    "Description" : "Information about the value",
    "Value" : "Value to return",
    "Export" : {
      "Name" : "Value to export"
    }
  }
}
```

Here it is in the YAML format:

```
Outputs:
  Logical ID:
    Description: Information about the value
    Value: Value to return
    Export:
      Name: Value to export
```

For a demo, we want to retrieve the bucket name when we create an Amazon S3 bucket. We pass our bucket resource into the `Outputs` section. We can implement it with the following scripts in JSON:

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Amazon S3 with output",
    "Resources" : {
        "MySimpleBucket" : {
            "Type" : "AWS::S3::Bucket"
        }
    },
    "Outputs" : {
        "BucketName" : {
            "Value" : { "Ref" : "MySimpleBucket" }
        }
    }
}
```

Here it is in YAML:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Amazon S3 with custom Bucket name
Resources:
  MySimpleBucket:
    Type: AWS::S3::Bucket
Outputs:
  BucketName:
    Value:
      Ref: MySimpleBucket
```

Save this file as `simple-s3-outputs.json` or `simple-s3-outputs.yaml`.

Then, you can upload this template to CloudFormation. If done, you should see the Amazon S3 bucket name on the **Outputs** tab in your stack:

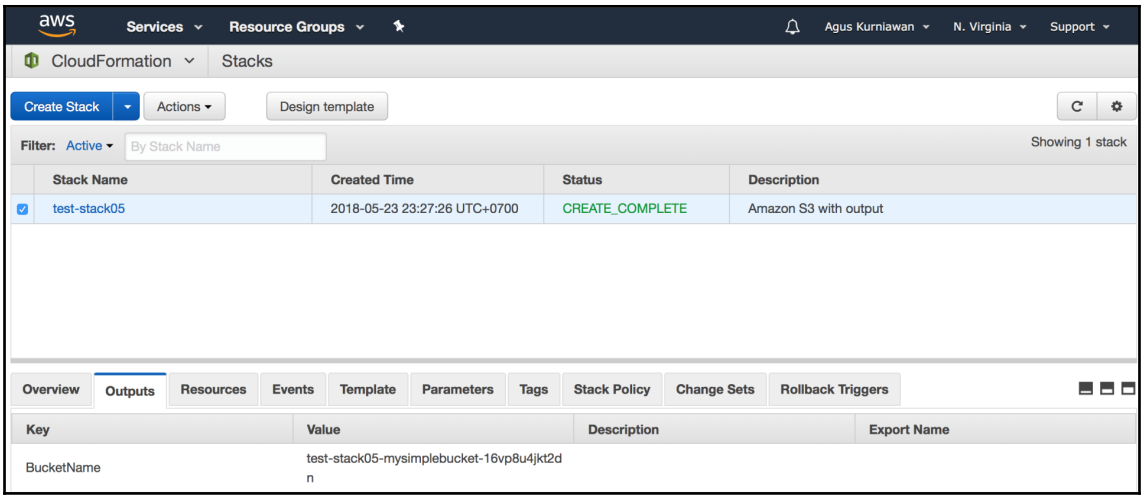


Figure 3.17: Output messages from CloudFormation

You have learned about CloudFormation output. Keep practicing by applying some output from various AWS resources.

Next, we'll try to build EC2 using CloudFormation.

Demo – building Amazon EC2 using AWS CloudFormation

In this section, we will build Amazon EC2 using CloudFormation. We explored some of the CloudFormation attributes for EC2 in previous sections. Just so you know, Amazon EC2 may not be free, so you may be charged.

Next, we'll prepare before we build EC2 on CloudFormation.

Preparing

To work with the EC2 instance, we should have a key-pair to access EC2. If you don't have a key-pair yet, you can create it through the EC2 management console.

Open your browser and navigate to `https://console.aws.amazon.com/ec2`. You should see the EC2 management console dashboard. You can click on the **Key Pairs** option on the left menu so that you get the following screen:

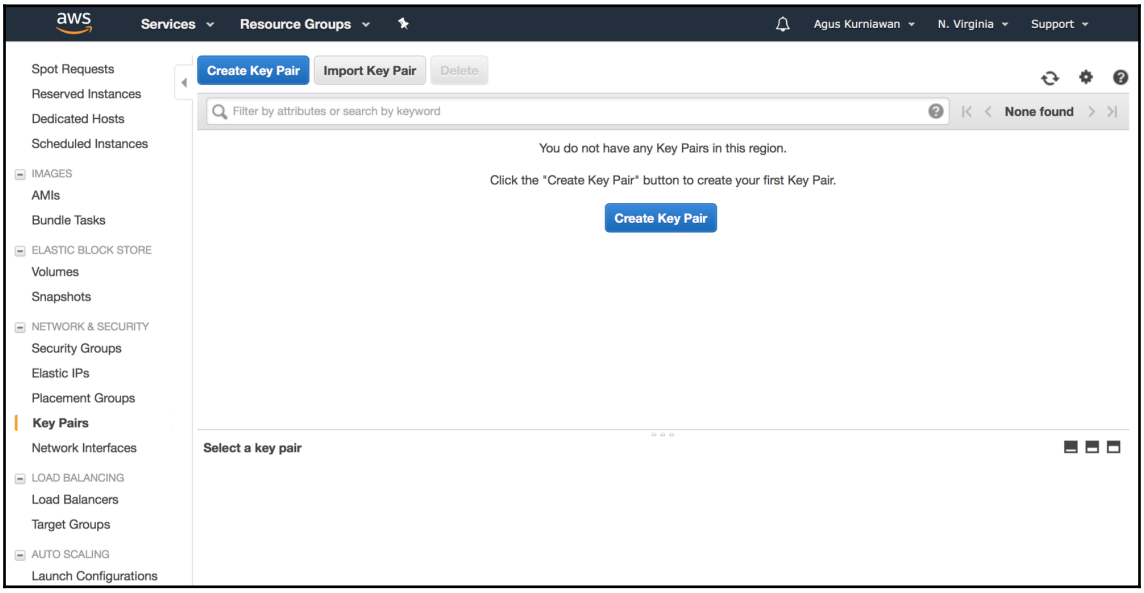


Figure 3.18: AWS EC2 console management

You can click on the **Create Key Pair** button to create a new key-pair. You should get a dialog box, as shown in the following screenshot. Fill in the **Key pair name** field and click on the **Create** button in the dialog box:

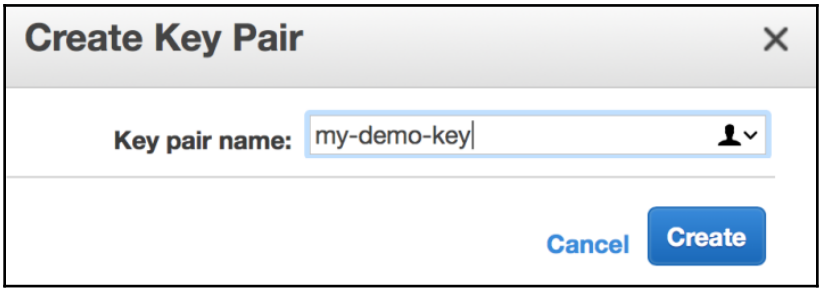


Figure 3.19: Creating a key-pair on AWS EC2 management console

Now you have a key-pair that we can use in our EC2 instance. Next, we'll develop a CloudFormation template for EC2.

Developing a CloudFormation template

In this section, we'll develop a CloudFormation template for EC2. Some attributes have already been explained in the previous sections in this chapter. Since we need a key-pair on EC2, we set it on the `Parameters` attribute so that users can select their key-pairs. The following is a sample program in JSON:

```
"KeyName": {
        "Description": "Name of an existing EC2 KeyPair to enable SSH
        access to the instance",
        "Type": "AWS::EC2::KeyPair::KeyName",
        "ConstraintDescription": "must be the name of an existing EC2
        KeyPair."
},
```

Here it is in the YAML format:

```
KeyName:
  Description: Name of an existing EC2 KeyPair to enable SSH access
  to the instance
  Type: AWS::EC2::KeyPair::KeyName
  ConstraintDescription: must be the name of an existing EC2 KeyPair.
```

A key-pair from the `KeyName` parameter will be used on the EC2 instance. We put it on the resource as follows:

```
"Resources": {
   "EC2Instance": {
     "Type": "AWS::EC2::Instance",
     "Properties": {
       ...
       "KeyName": {
         "Ref": "KeyName"
       },
```

Here is it in YAML:

```
Resources:
   EC2Instance:
     Type: AWS::EC2::Instance
     Properties:
       ...
       KeyName:
         Ref: KeyName
```

We also need a public IP address for the EC2 instance, so we access it from SSH. We define it on the `SSHLocation` section from the `Parameters` attribute. The JSON format is as follows:

```
      "SSHLocation": {
        "Description": "The IP address range that can be used to SSH to the
EC2 instances",
        "Type": "String",
        "MinLength": "9",
        "MaxLength": "18",
        "Default": "0.0.0.0/0",
        "AllowedPattern":
"(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
        "ConstraintDescription": "must be a valid IP CIDR range of the form
x.x.x.x/x."
      }
```

Here it is in YAML:

```
SSHLocation:
    Description: The IP address range that can be used to SSH to the EC2
instances
    Type: String
    MinLength: '9'
    MaxLength: '18'
    Default: 0.0.0.0/0
    AllowedPattern:
"(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})"
    ConstraintDescription: must be a valid IP CIDR range of the form
x.x.x.x/x.
```

For details of the EC2 instance template, check out `ec2demo.json` and `ec2demo.yaml`. You can start deploying this template to AWS CloudFormation. We'll perform this task in the next section.

Deploying the template

After completing our CloudFormation template for EC2, you can upload it on AWS CloudFormation. You should select the **KeyName** and other options on the template:

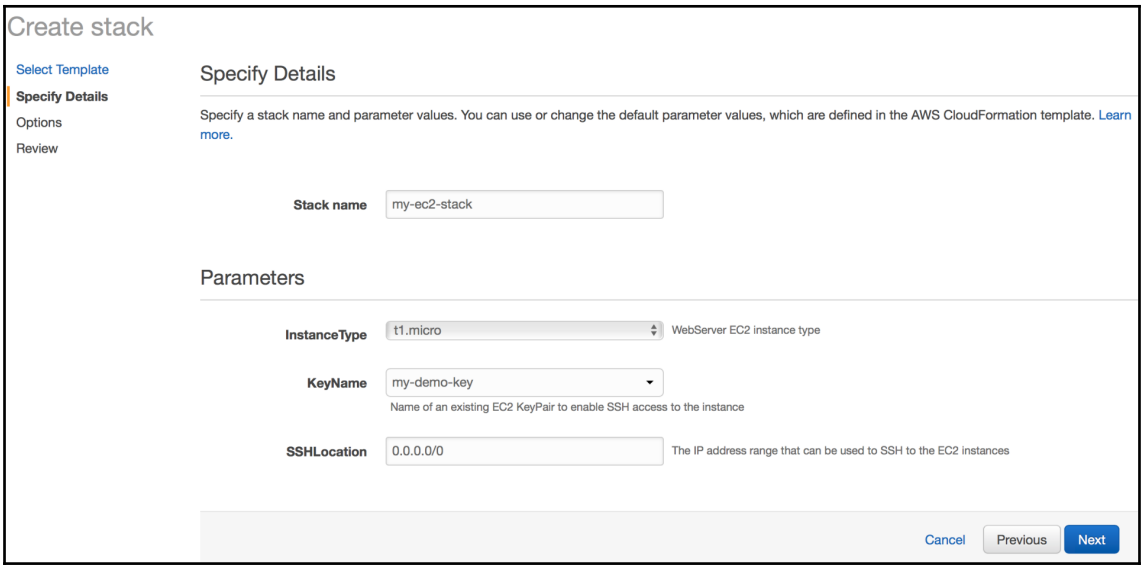


Figure 3.20: Deploying EC2 through AWS CloudFormation

You probably set the IP address for SSH access on the **SSHLocation** parameter. Then, follow the instructions to deploy the template.

After this has been created, you can check the progress of CloudFormation provision on the CloudFormation dashboard:

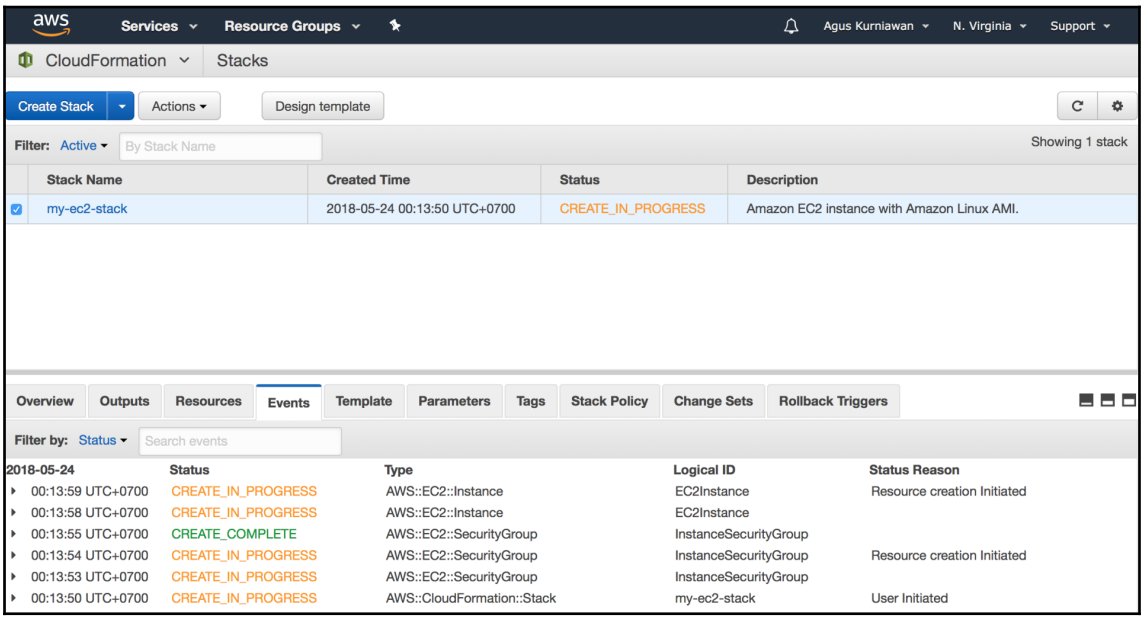


Figure 3.21: AWS EC2 instance has created

After creating the EC2 instance, you can verify this on the EC2 management console. You should see the EC2 instance as shown in the following screenshot:

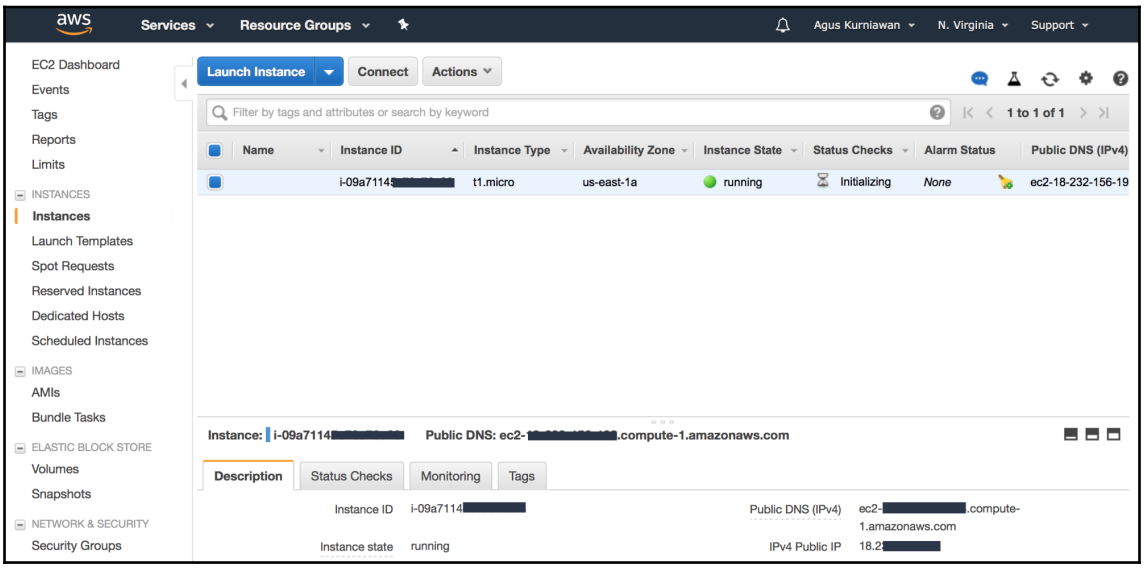


Figure 3.22: Verifying EC2 instance on EC2 management console

Now, you can manage your EC2 instance on the management console . If you want to modify your EC2 instance, you can update the CloudFormation template. Then, you can provision that template.

Summary

We have learned how to build AWS CloudFormation templates using JSON and YAML. We reviewed the CloudFormation template format. Then, we looked at some attributes from the CloudFormation template. Finally, we deployed Amazon EC2 using the CloudFormation template.

In the next chapter, we will learn about CloudFormation StackSets.

Questions

1. What is the CloudFormation template?
2. How do we develop the CloudFormation template?
3. How do you implement AWS resources in the CloudFormation template?

4
AWS CloudFormation StackSets

Serving worldwide customers from various regions needs more attention in designing applications and infrastructure. Sometimes, deploying infrastructure to some regions is not easy. Moreover, we should perform infrastructure changes on various regions. This chapter will explore how to design and build infrastructure in various regions by applying CloudFormation StackSets.

The following topics will be covered in this chapter:

- Introduction to AWS CloudFormation StackSets
- Preparing CloudFormation StackSets
- Implementing StackSets using the management console
- Creating StackSets using the AWS CLI
- Editing CloudFormation StackSets
- Deleting CloudFormation StackSets

Introduction to AWS CloudFormation StackSets

A company with worldwide customers probably encounters problems when serving their customers. We should be able to provide services that are close to a customer's location. In an IT problem context, we should deploy our applications and services in various regions to address network latency.

We can describe our problem to serve worldwide customers as shown in *Figure 4.1*. We have customers from the US, Asia, and Europe. We have applications and services that should be deployed in those regions. Sometimes, each region has unique features on applications and services. Problems will become more complex if we change our applications and services and deploy them on various servers:

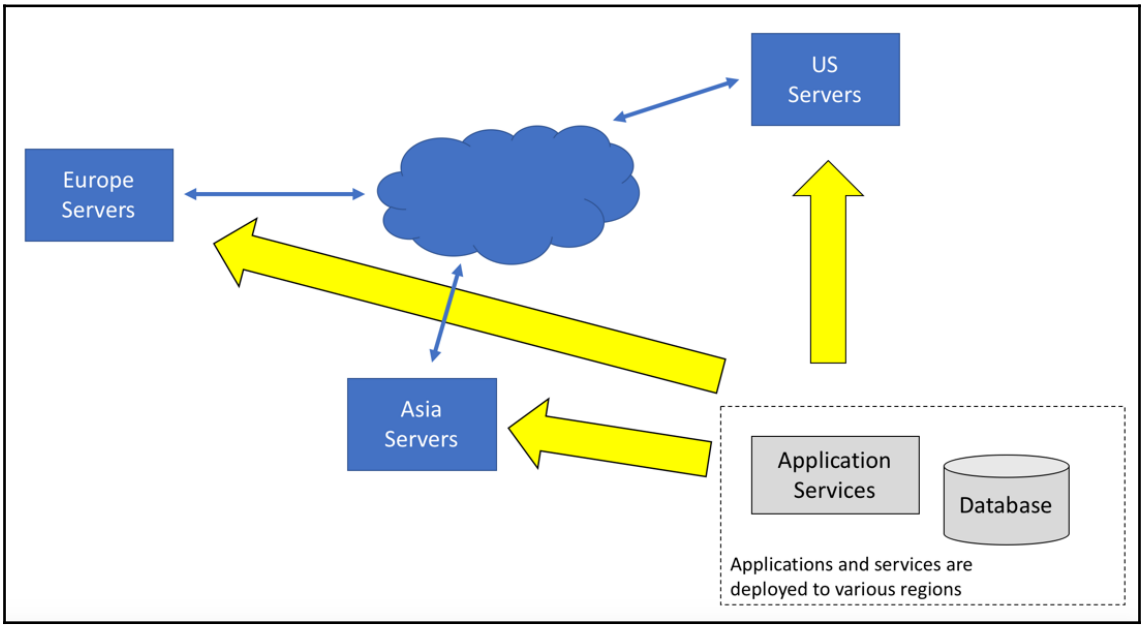


Figure 4.1: Infrastructure design with regional features

To address this issue, we can apply the AWS CloudFormation template to build a dynamic infrastructure. We could also deploy the AWS CloudFormation template to various regions. Some CloudFormation templates can be deployed in certain regions.

In this chapter, we'll explore how to use AWS CloudFormation StackSets to deploy CloudFormation stacks in certain regions.

Preparing CloudFormation StackSets

We should perform some tasks in order to work with CloudFormation StackSets. These tasks are part of the security rules from AWS CloudFormation. To get started with CloudFormation StackSets, let's first perform the following tasks:

1. Getting the user ID from the IAM user as the AWS CloudFormation administrator
2. Creating an IAM role, `AWSCloudFormationStackSetAdministrationRole`
3. Creating a service role, `AWSCloudFormationStackSetExecutionRole`

We'll go through these tasks in the next sections.

Getting the user ID from the IAM user

First, you should have an administrator account on AWS IAM. Copy your ARN user from your IAM account on the AWS IAM Management Console at `https://console.aws.amazon.com/iam/`. For instance, you could have an ARN user account as follows:

```
arn:aws:iam::123456789012:user/myusername
```

Pick up your user ID from the account number of the ARN user. From the preceding ARN user, we have `123456789012` as the user ID. We will use this user ID to attach to a CloudFormation policy.

Next, we will create the IAM role, called `AWSCloudFormationStackSetAdministrationRole`, on the IAM management console.

Creating the AWSCloudFormationStackSetAdministrationRole IAM role

To build CloudFormation StackSets, we should create an IAM role, `AWSCloudFormationStackSetAdministrationRole`. This is required by AWS. If we don't, we will obtain an error when creating a CloudFormation StackSet.

We can create an IAM role, named `AWSCloudFormationStackSetAdministrationRole`, using CloudFormation Stack. First, we create the CloudFormation template. In this case, we use the following YAML format:

```
AWSTemplateFormatVersion: 2010-09-09
Description: Configure the AWSCloudFormationStackSetAdministrationRole to
enable use of AWS CloudFormation StackSets.

Resources:
  AdministrationRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: AWSCloudFormationStackSetAdministrationRole
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service: cloudformation.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: /
      Policies:
        - PolicyName: AssumeRole-AWSCloudFormationStackSetExecutionRole
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - sts:AssumeRole
                Resource:
                  -
"arn:aws:iam::*:role/AWSCloudFormationStackSetExecutionRole"
```

You can save these scripts into a file called `AWSCloudFormationStackSetAdministrationRole.yaml`.

Now, you can navigate to the AWS CloudFormation dashboard at `https://console.aws.amazon.com/cloudformation/` and create a new CloudFormation stack. Upload the `AWSCloudFormationStackSetAdministrationRole.yaml` file. Give it a stack name, for instance `AWSCloudFormationStackSetAdministrationRole`, as shown in the following screenshot:

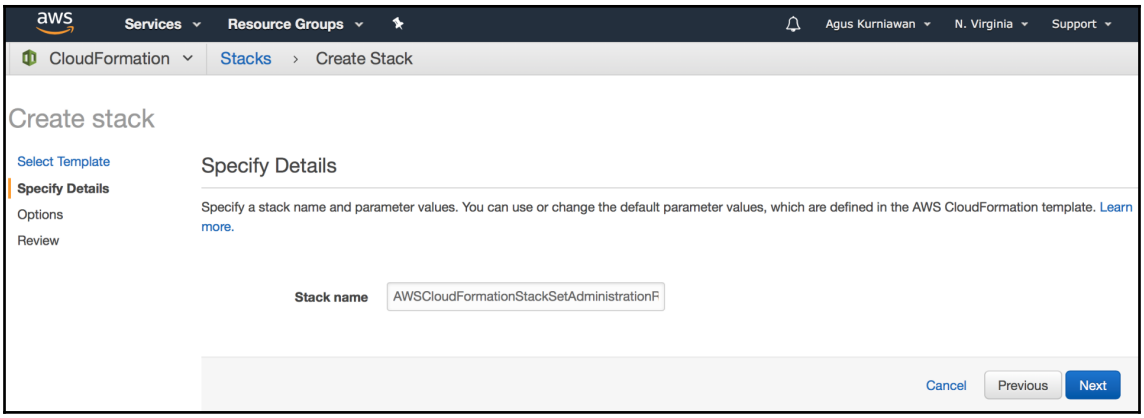


Figure 4.2: Adding CloudFormation stack to create an IAM role, AWSCloudFormationStackSetAdministrationRole

When you're done, click on the **Next** button until you complete this task. Read `Chapter 2`, *Building Your First AWS CloudFormation Project* and `Chapter 3`, *Developing AWS CloudFormation Templates* for details on how to work with CloudFormation stacks. After you've created a CloudFormation stack, you should see your CloudFormation stack on the CloudFormation stacks dashboard, as shown in the following screenshot:

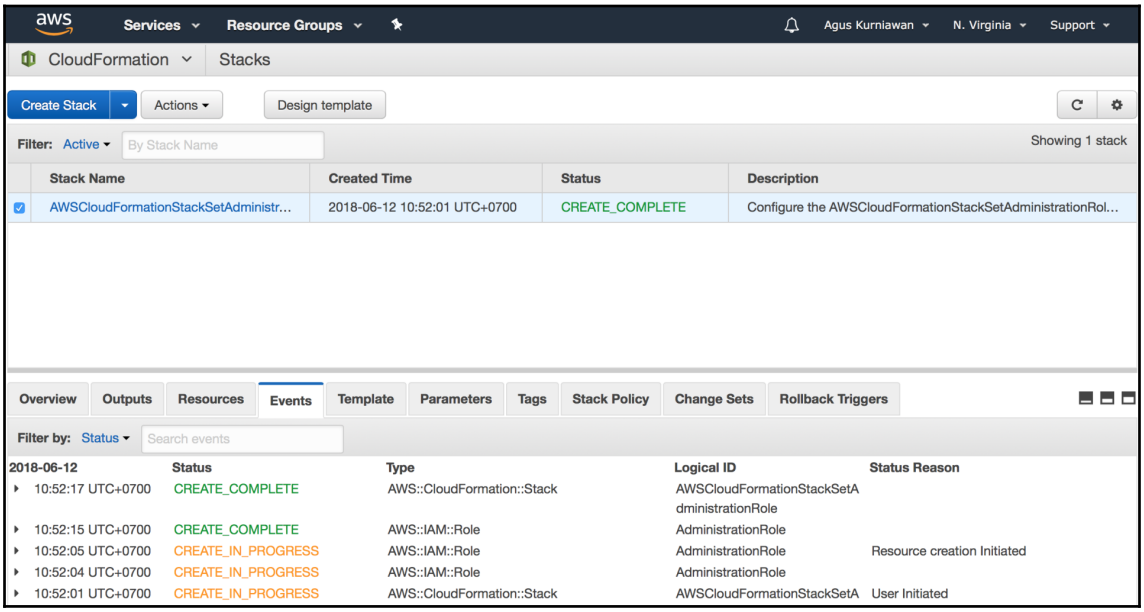


Figure 4.3: Creation of the AWSCloudFormationStackSetAdministrationRole role was completed

If you see the stack status as **CREATE_COMPLETE**, it means your CloudFormation stack has already been created. Now, you can verify it by opening the IAM AWS Management Console. Open a browser and navigate to `https://console.aws.amazon.com/iam/`.

On the management console, click on the **Roles** in the left menu. You should see `AWSCloudFormationStackSetAdministrationRole` on the role list:

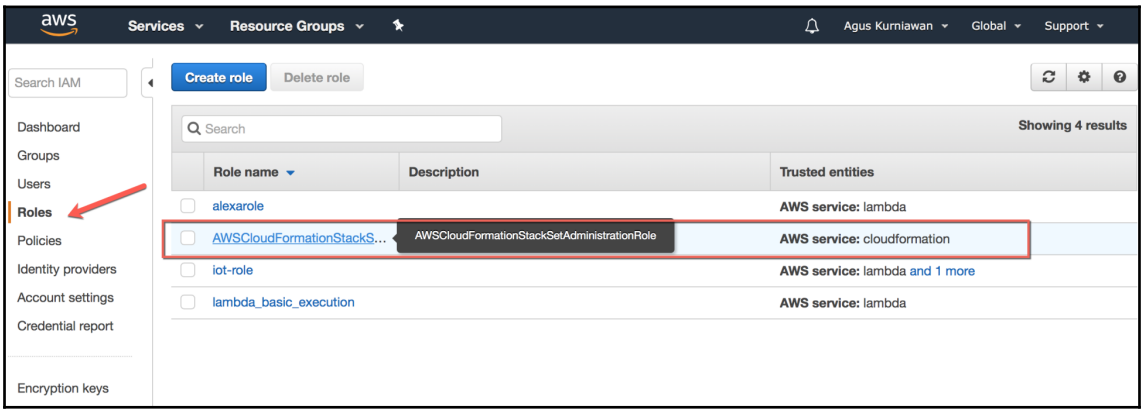


Figure 4.4: Checking the AWSCloudFormationStackSetAdministrationRole IAM roles dashboard

You have created the `AWSCloudFormationStackSetAdministrationRole` role on AWS IAM. Next, we should create a service role named `AWSCloudFormationStackSetExecutionRole`.

Creating a service role – AWSCloudFormationStackSetExecutionRole

Now that we've created the `AWSCloudFormationStackSetAdministrationRole` role on IAM Roles, we'll create a service role named `AWSCloudFormationStackSetExecutionRole`.

We can create a service role using CloudFormation. You can write these scripts for the CloudFormation template:

```
AWSTemplateFormatVersion: 2010-09-09
Description: Configure the AWSCloudFormationStackSetExecutionRole to enable
use of your account as a target account in AWS CloudFormation StackSets.

Parameters:
  AdministratorAccountId:
    Type: String
    Description: AWS Account Id of the administrator account (the account
in which StackSets will be created).
    MaxLength: 12
    MinLength: 12

Resources:
  ExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: AWSCloudFormationStackSetExecutionRole
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              AWS:
                - !Ref AdministratorAccountId
            Action:
              - sts:AssumeRole
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AdministratorAccess
```

Save these scripts into a file called `AWSCloudFormationStackSetExecutionRole.yaml`.

This service needs an administrator account to enable it to run a service on CloudFormation. We have prepared the IAM user and copied the ARN of the user. Now, you can open a browser and navigate to the CloudFormation dashboard. Upload the `AWSCloudFormationStackSetExecutionRole.yaml` file to AWS CloudFormation stacks. Fill in your stack name and the ARN user from your IAM user, for instance, `123456789012`.

You can see the stack screen in the following screenshot:

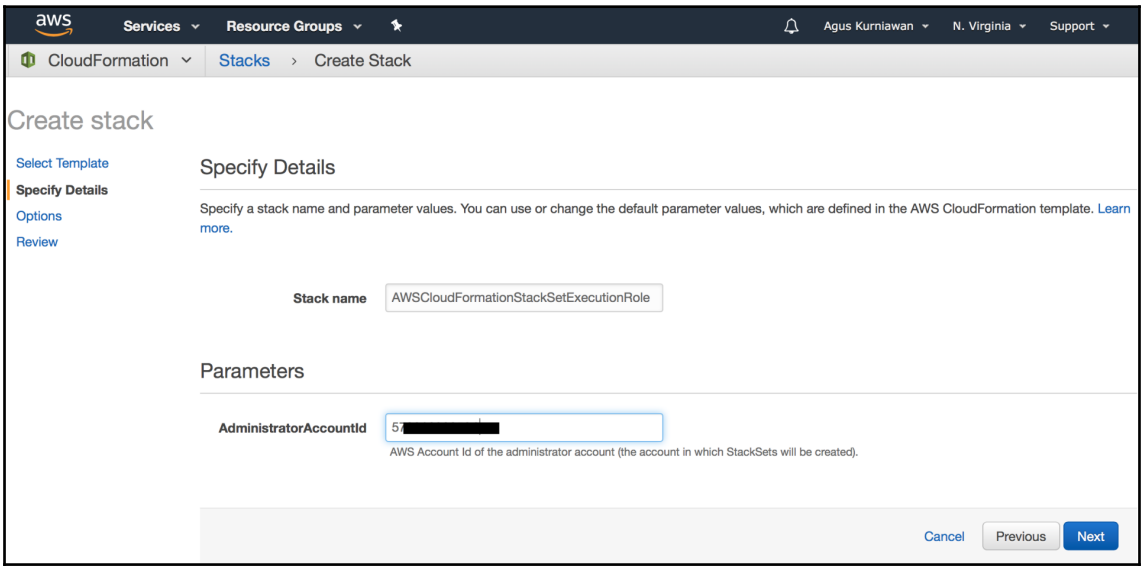


Figure 4.5: Creating a service role, AWSCloudFormationStackSetExecutionRole

You can follow created stacks instructions. Once done, you can check your stack creation status. Make sure your stack completes:

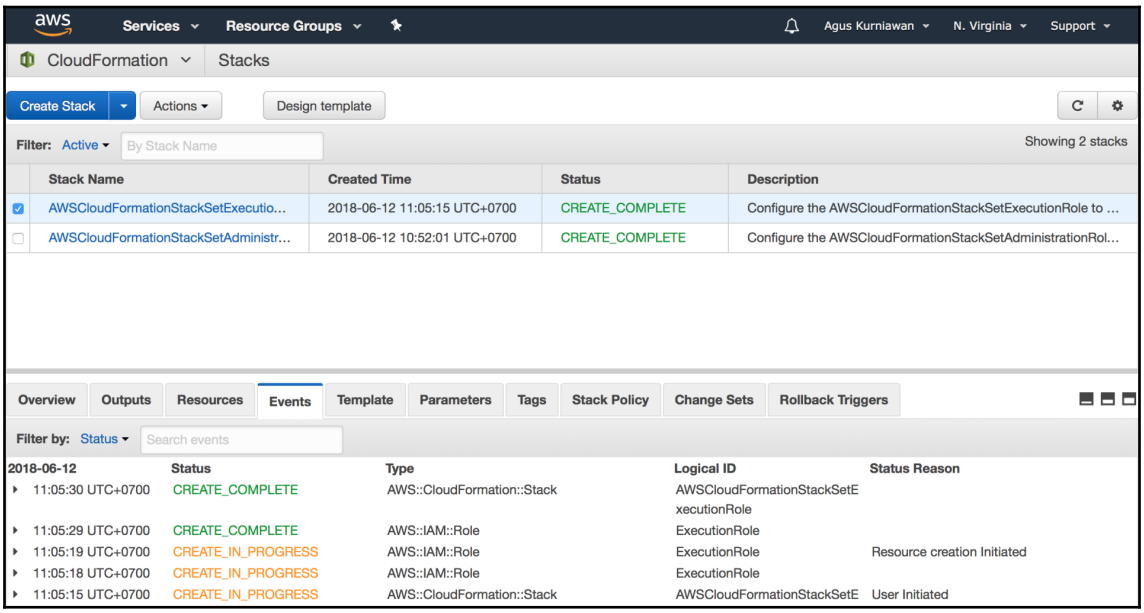


Figure 4.6: AWSCloudFormationStackSetExecutionRole service role was completed

We have created a service role, `AWSCloudFormationStackSetExecutionRole`. Now, we are ready to create a StackSet. Next, we will create a StackSet using the management console and the AWS CLI.

Implementing StackSets using management console

In this section, we will create a StackSet using the **Web Management Console** (**WMC**). You should have created the IAM role, `AWSCloudFormationStackSetAdministrationRole`, and a service role, `AWSCloudFormationStackSetExecutionRole`.

Now, you can open a browser and navigate to the CloudFormation StackSet dashboard at `https://console.aws.amazon.com/cloudformation/stacksets/home`. You should see the dashboard shown in *Figure 4.7*:

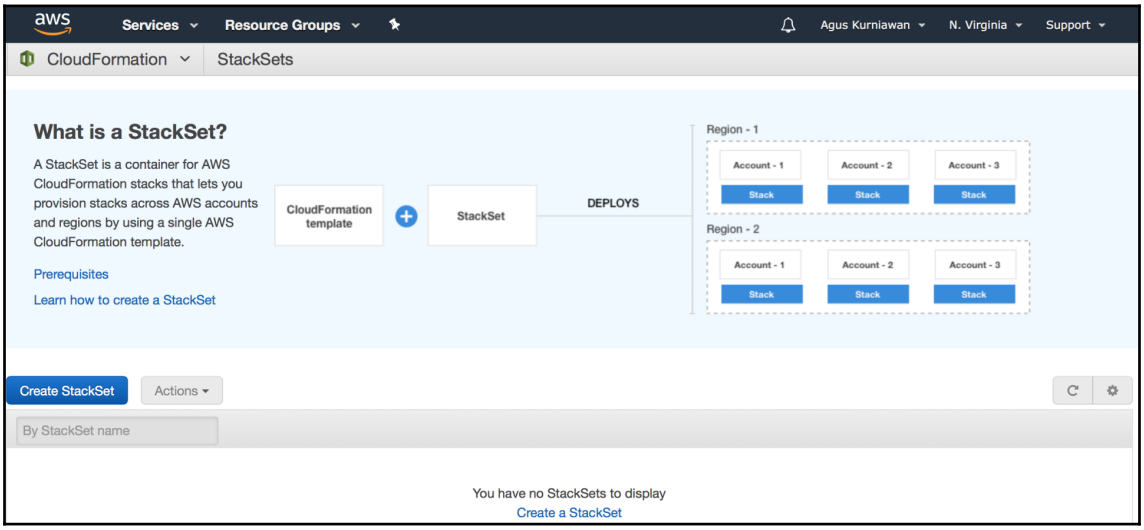


Figure 4.7: CloudFormation StackSets dashboard

By default, the CloudFormation management console shows the **Stacks** dashboard. You can select the StackSets dashboard by clicking on the **StackSets** menu from the main menu, as shown in *Figure 4.8*:

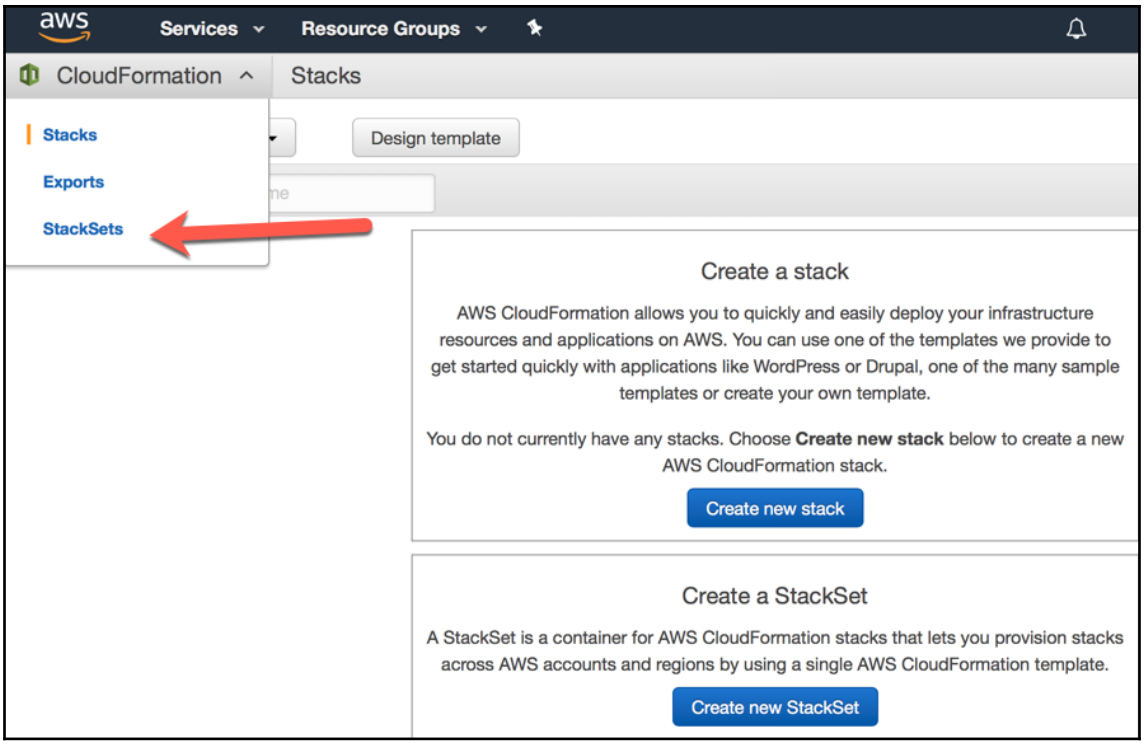


Figure 4.8: Opening the CloudFormation StackSets dashboard from the main menu

In this section, we will perform the following tasks:

1. Creating a new StackSet
2. Adding a CloudFormation stack into a StackSet
3. Deleting a CloudFormation stack from a StackSet

We'll perform these tasks in the next sections.

Creating a new StackSet

In this section, we'll create a StackSet using the management console. A StackSet consists of stacks that are deployed in one or more regions.

Some steps should be followed to create a StackSet. Perform these steps:

1. Open your browser and navigate to the CloudFormation StackSets dashboard at `https://console.aws.amazon.com/cloudformation/stacksets/home`.
2. You should see the CloudFormation StackSets dashboard, shown in *Figure 4.7*.
3. To create a new StackSet, click on the **Create StackSet** button.
4. Once you've clicked it, you should see the screen shown in *Figure 4.9*:

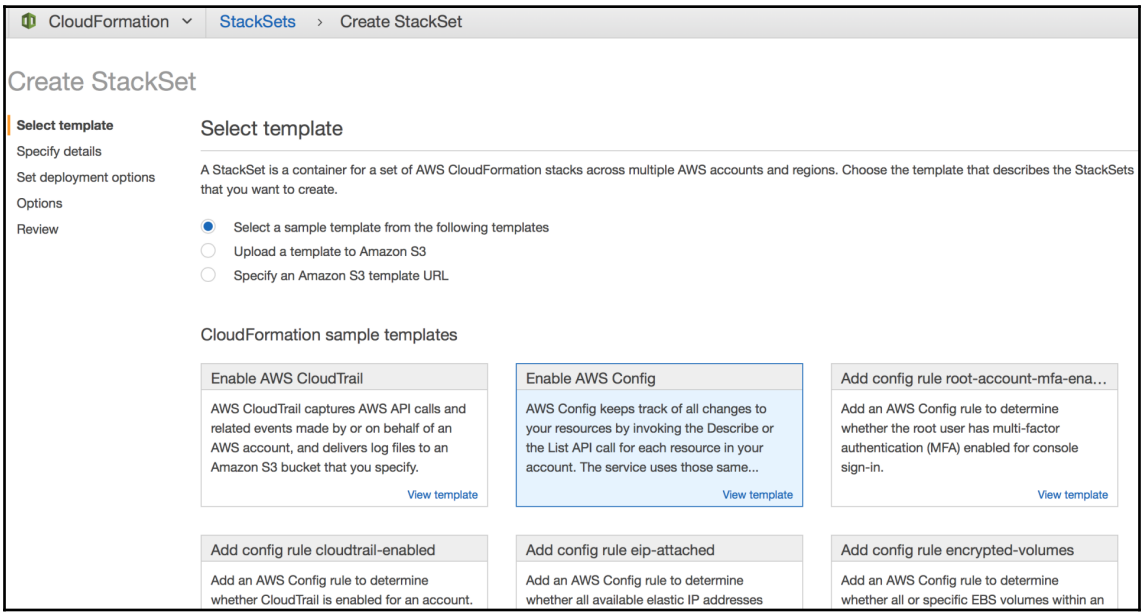


Figure 4.9: Creating StackSet by selecting template

5. Choose the **Select a sample template from the following templates** option.
6. Click on the **Enable AWS Config** template. Once done, click on the **Next** button.
7. You should see the screen, shown in *Figure 4.10*:

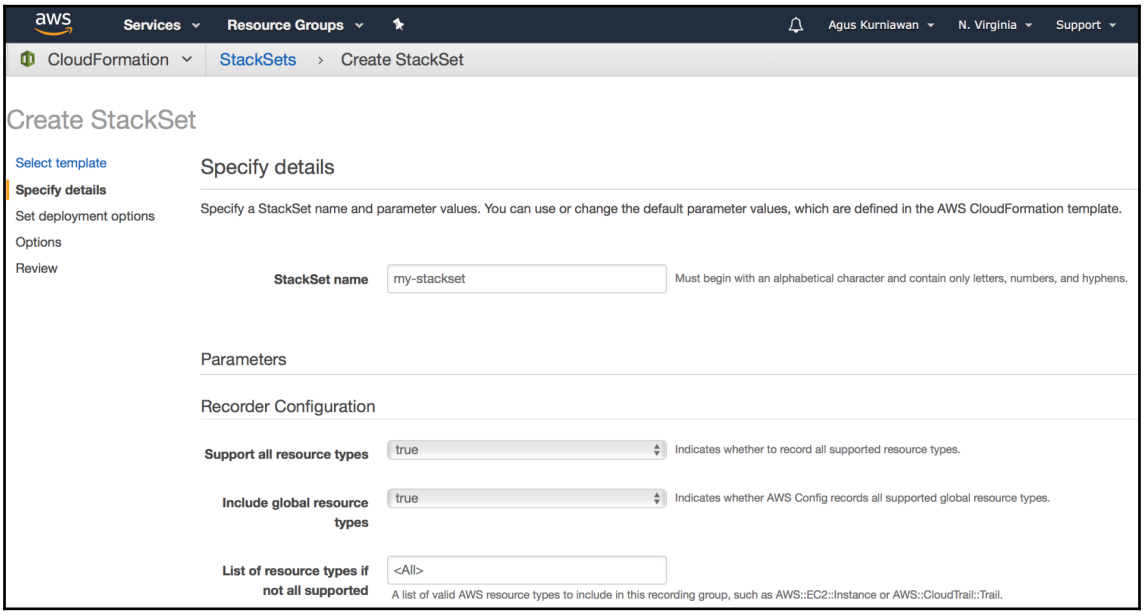


Figure 4.10: Filling StackSet name and attributes

8. Fill in your StackSet name, for instance, `my-stackset`. Once done, click on the **Next** button.
9. You should see the screen shown in *Figure 4.11*. Select the **Deploy stacks in accounts** option.
10. Enter your ARN user in the textbox. You can fill in some accounts on this StackSet:

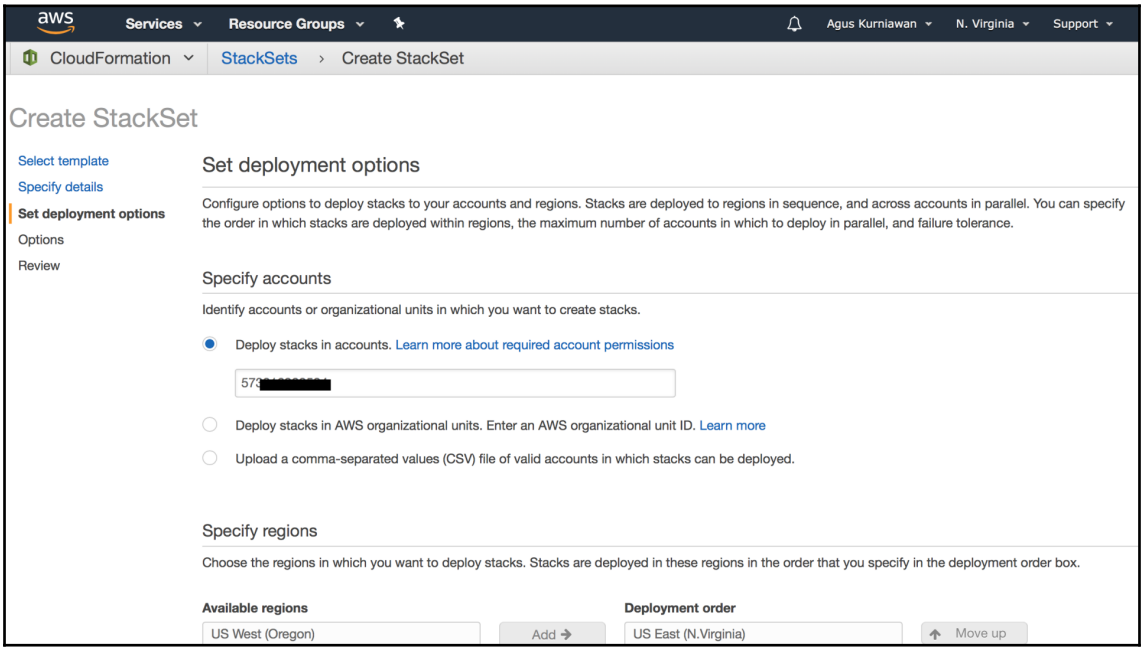


Figure 4.11: Setting up accounts for StackSet

11. Scroll down so you see the screen shown in *Figure 4.12*. Select one or a few regions for your accounts.
12. For our demo, I've selected the **US East (N.Virginia)** and **Asia Pacific (Singapore)** regions.
13. Once done, click on the **Next** button:

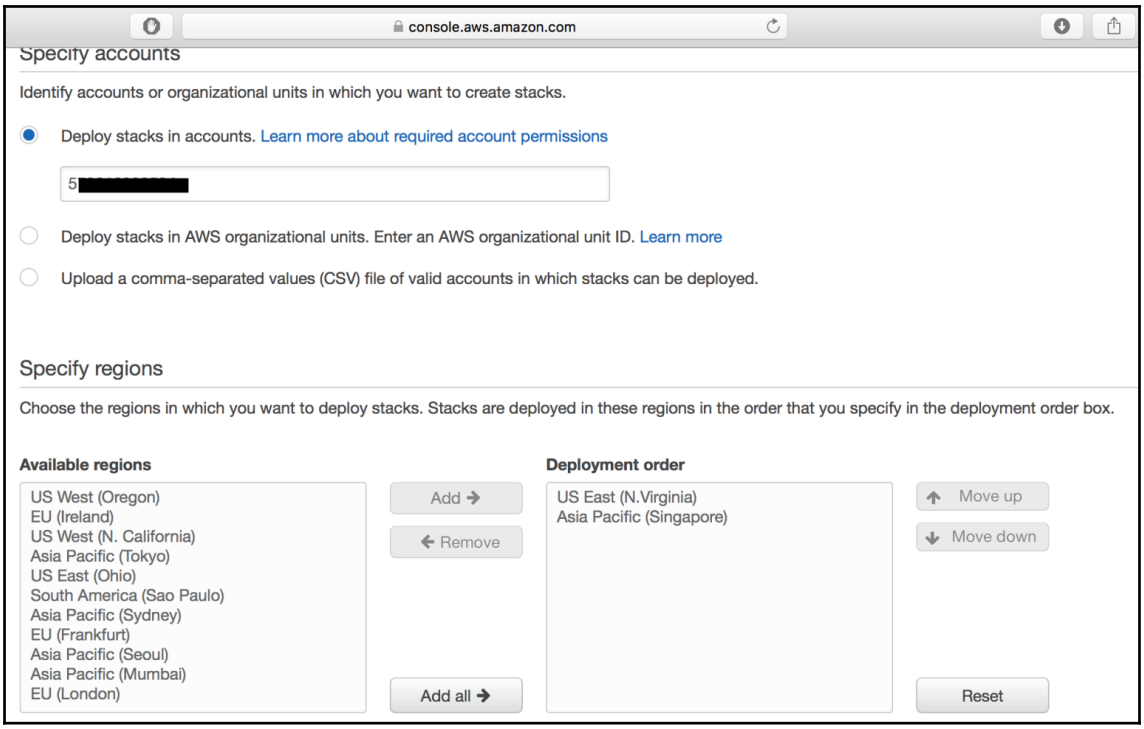


Figure 4.12: Setting targeted regions

14. You should get the screen shown in *Figure 4.13*. In this form, we'll do nothing. Click on the **Next** button to continue:

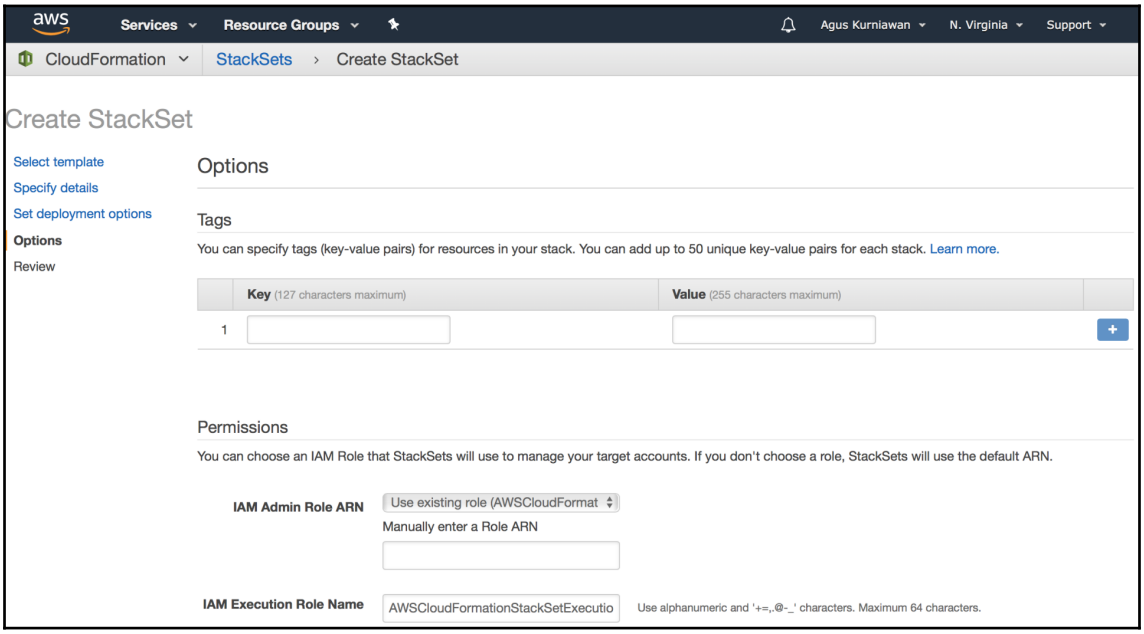


Figure 4.13: Setting up options for StackSet

15. You will get the review screen, shown in *Figure 4.14*. Review your input. Click on the **Create** button to start to create a StackSet:

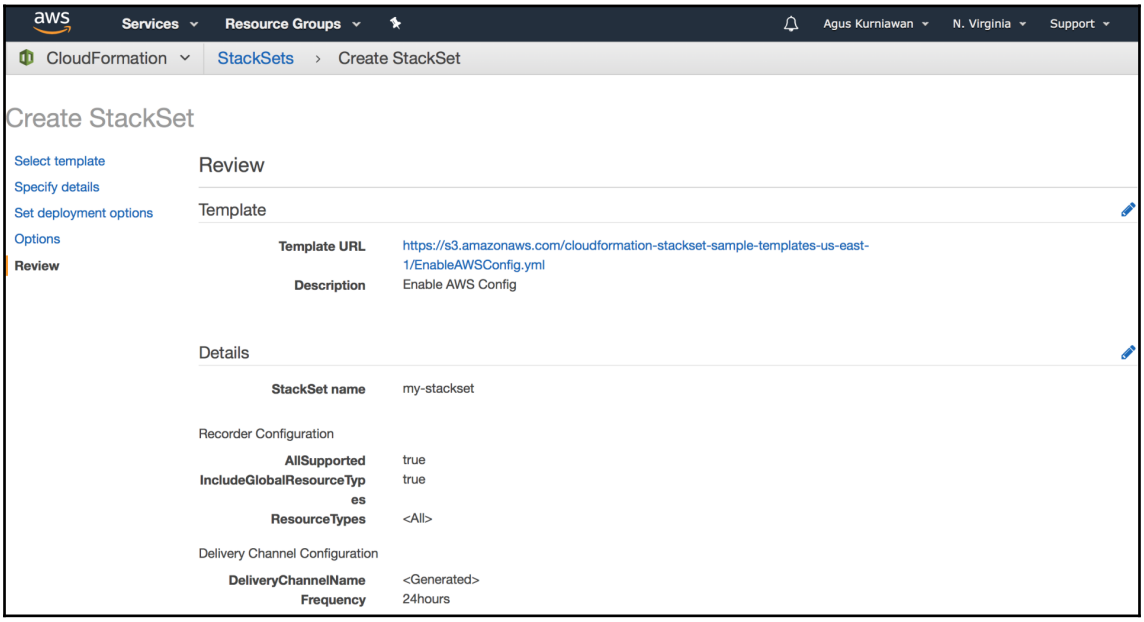


Figure 4.14: Confirmation for creating StackSet

16. You should get the screen shown in *Figure 4.15*. You should see your **StackSet with Enable AWS Config** template:

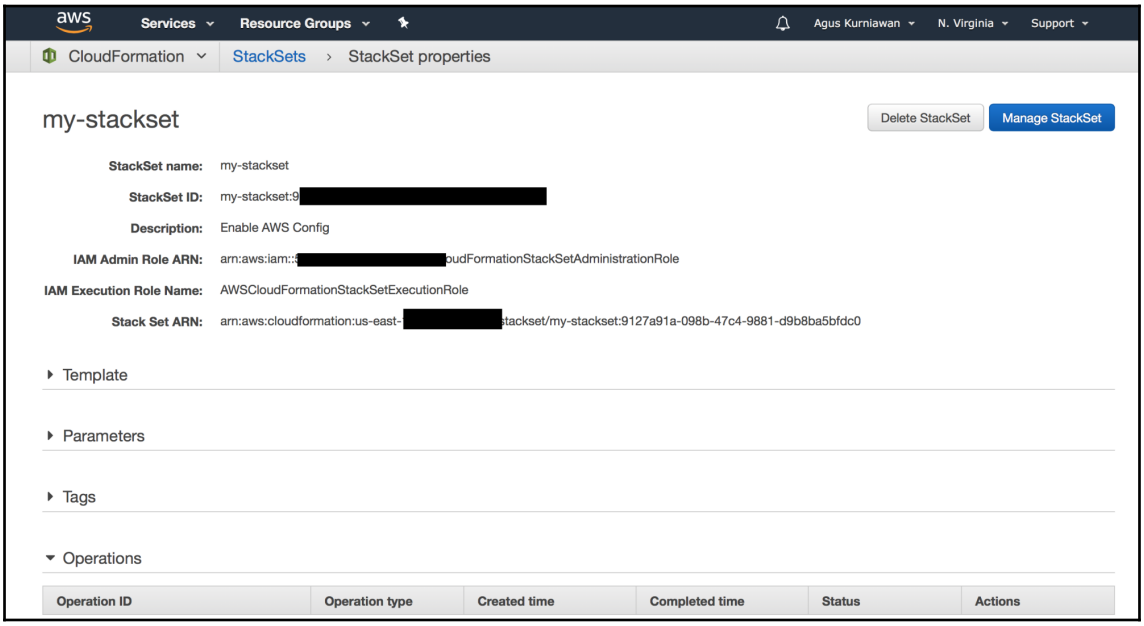


Figure 4-15. StackSet was created

17. To see whether your StackSet was created successfully or not, scroll down to view the **Operations** and **Stacks** statuses. You should see the screen shown in *Figure 4.16*. You can see your operation and region stack status. If the StackSet operation status is **SUCCEEDED**, your StackSet was created successfully. You also should see all Stacks statuses are **CURRENT**, as shown in the following screenshot:

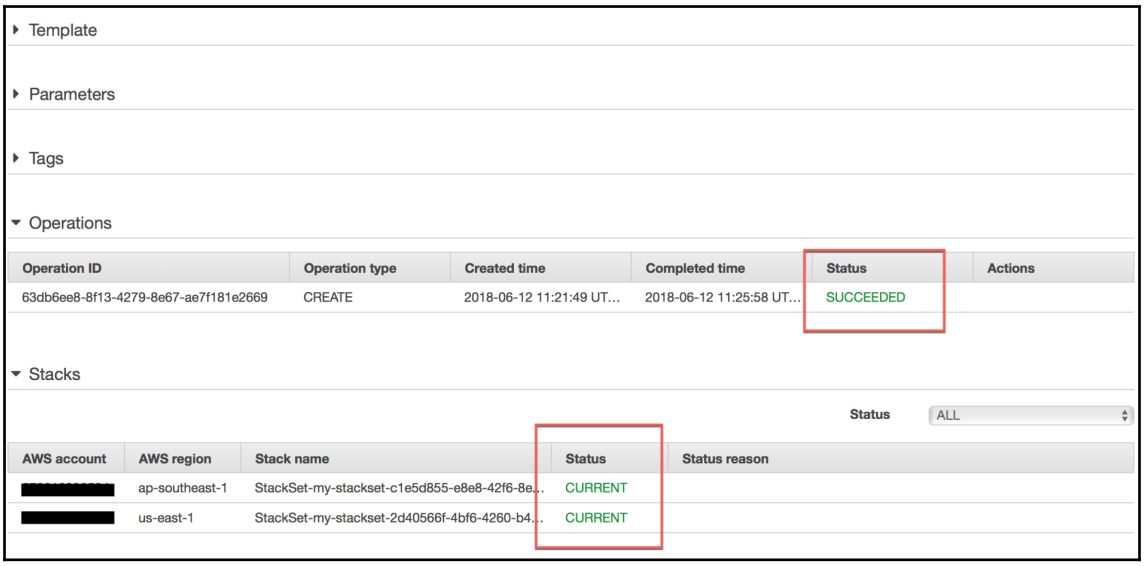


Figure 4.16: Checking the status on Operations and Stacks

Some status information for the **Operations** and **Stacks** sections can be found in *Table 4.1* and *Table 4.2*:

StackSets operation status	Operation
RUNNING	The operation is currently in progress
SUCCEEDED	The operation finished without exceeding the failure tolerance for the operation
FAILED	The number of stacks on which the operation could not be completed exceeded the user-defined failure tolerance
STOPPING	The operation is in the process of stopping, at the user's request
STOPPED	The operation has been stopped, at the user's request

Table 4.1: A list of status information for StackSets operations

Stack instance status	Operation
CURRENT	The stack is currently up-to-date with the stack set
OUTDATED	The stack is not currently up-to-date
INOPERABLE	A **DeleteStackInstances** operation has failed and left the stack in an unstable state

Table 4.2: A list of status information for the Stack instance

Now that you have created StackSet on CloudFormation through management console. Next, we'll create one or more stacks on StackSet.

Adding a new CloudFormation Stack

You can add additional stacks on an existing StackSet. In this case, you can't use a new CloudFormation stack template. You should use an existing stack template from StackSet that has been already deployed. We can use cross accounts and regions.

In this demo, I'll add a new region for my existing StackSet. Perform the following tasks to add a CloudFormation stack into a StackSet:

1. Open a browser and navigate to the CloudFormation StackSets dashboard at `https://console.aws.amazon.com/cloudformation/stacksets/home`.
2. You should see the screen shown in *Figure 4.17*:

Figure 4.17: A list of CloudFormation StackSets

3. Select a StackSet that will be added to stacks. Click on the **Actions** button so you get the menu shown in the following screenshot:

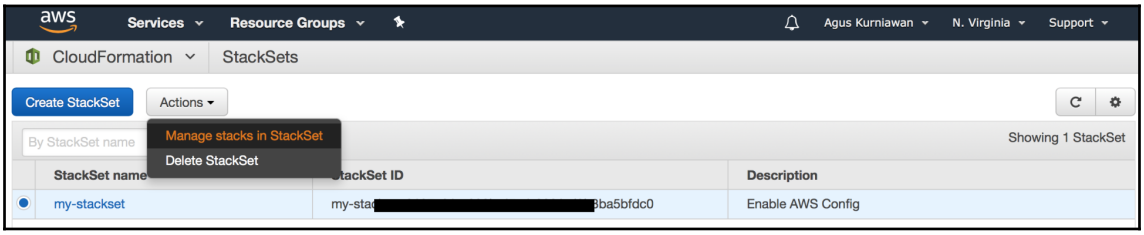


Figure 4.18: Actions menu on StackSets dashboard

4. Click on the **Manage stacks in StackSet** menu. You should get the screen shown in *Figure 4.19*. Select the **Create stacks** option:

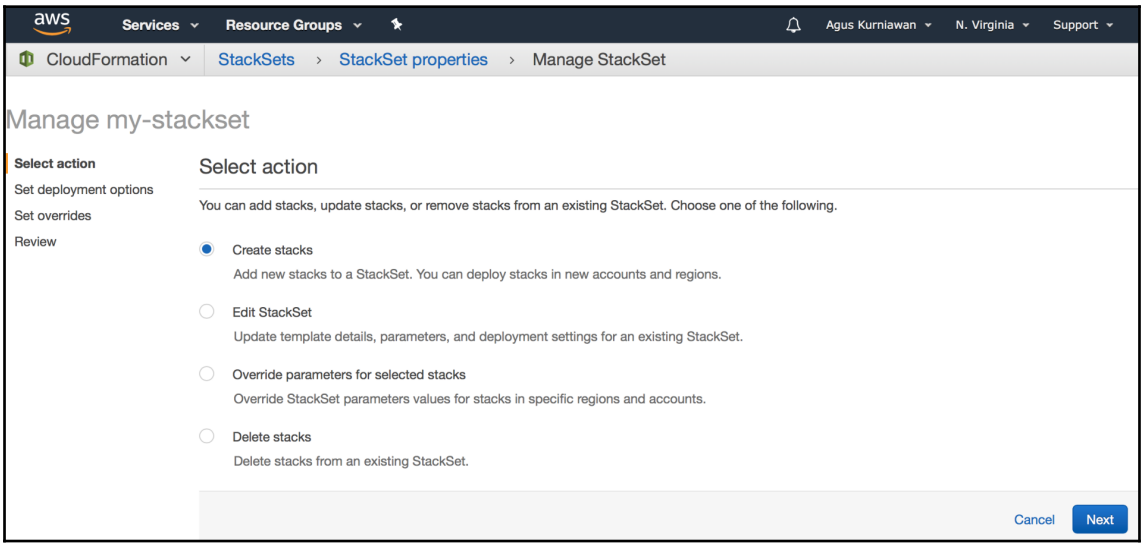


Figure 4.19: Select an option to add stacks

5. Once done, click on the **Next** button. You should get the screen shown in *Figure 4.20*. Select the **Create stacks in accounts** option. Fill in your AWS IAM accounts. You also should specify the regions that will be used for targeted deployment:

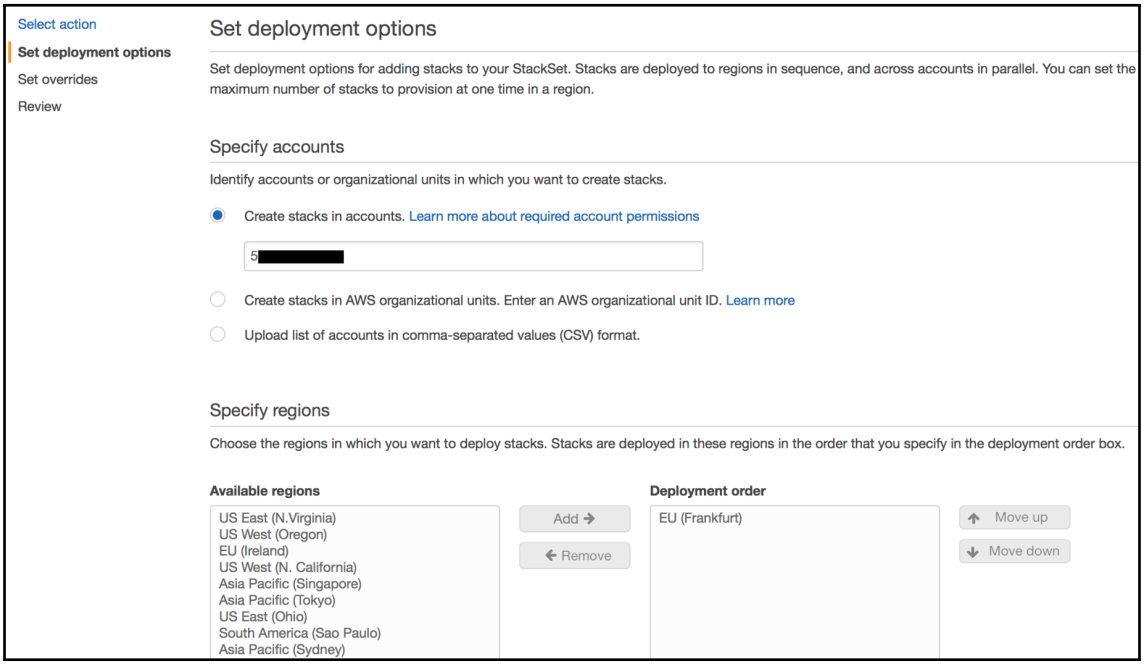


Figure 4.20: Set accounts and their regions

6. Once you've filled in the data, click on the **Next** button. You should get the screen shown in *Figure 4.21*. Set stack parameters. Once done, click on the **Next** button:

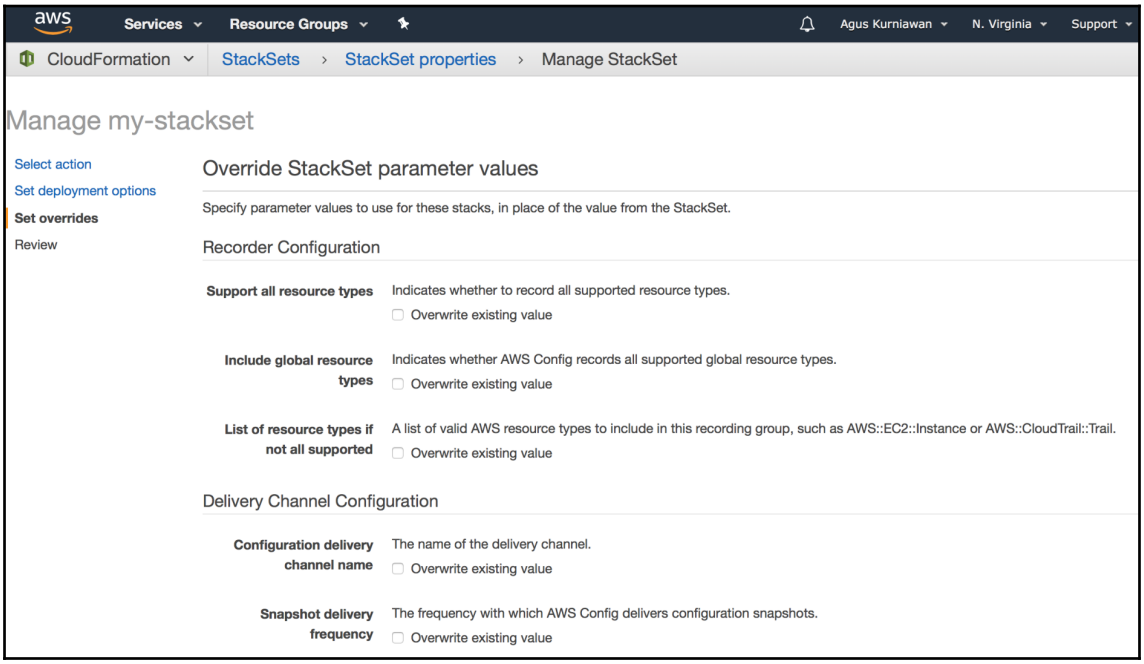


Figure 4.21: Set StackSet parameters

7. You will get a confirmation screen, which is shown in *Figure 4.22*. Check your input data. Once done, click on the **Create** button to provision your stack:

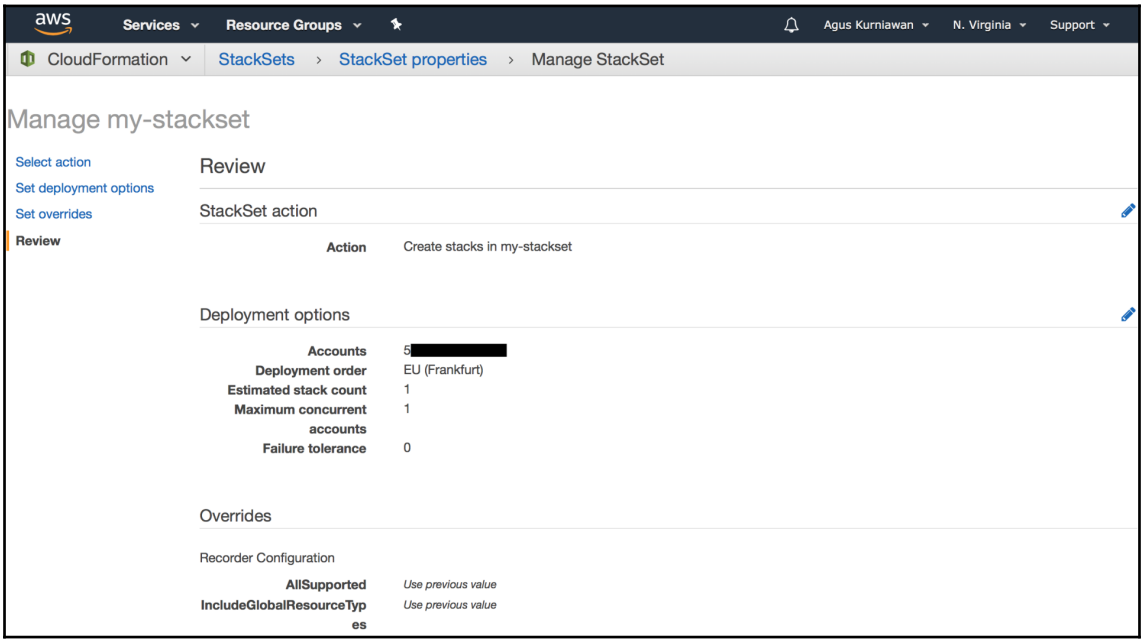


Figure 4.22: A confirmation for adding stacks

8. Once clicked, CloudFormation will perform this task. You should see your stack on the StackSet properties:

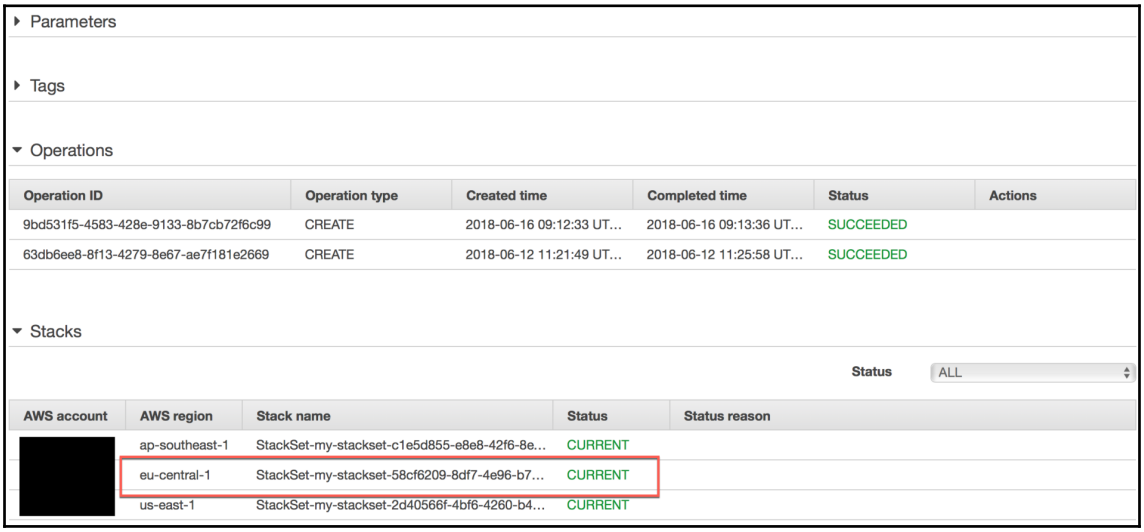


Figure 4.23: A new stack on an existing StackSet

This is the end of adding stacks. You can add more stacks as you need them. Next, you will learn how to delete stacks from StackSets.

Deleting CloudFormation stacks

You can delete stacks from StackSets if you think they're not going to be used. In this section, we'll learn how to remove stacks from StackSets.

Follow these steps to delete stacks:

1. Open a browser and navigate to CloudFormation StackSets (see *Figure 4.17*). Select and open your StackSet.
2. Click on the **Manage StackSet** button, and you will get the screen shown in the following screenshot:

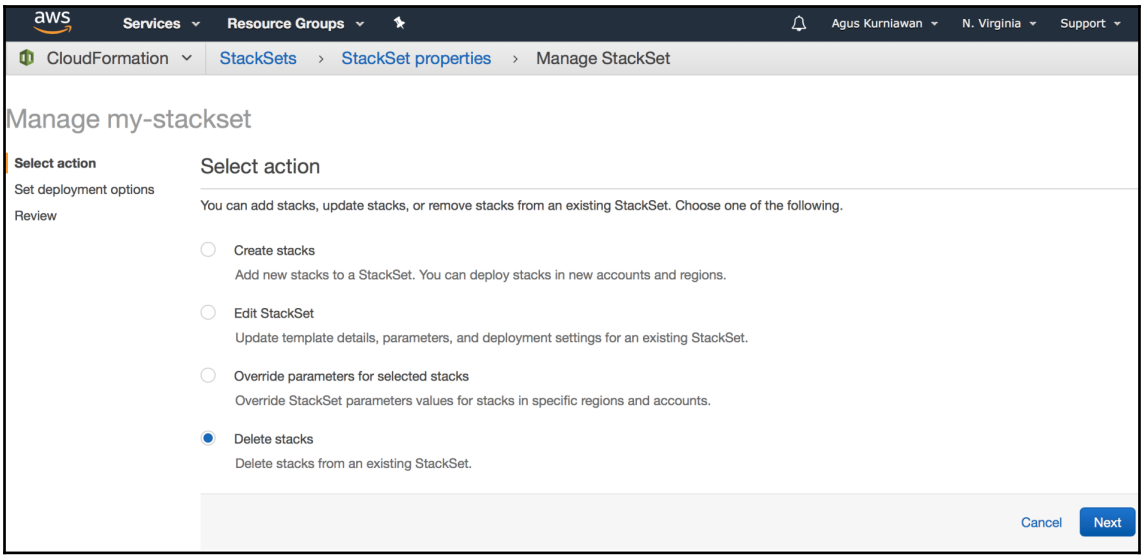


Figure 4.24: Select a deletion option for existing Stacks

3. Select the **Delete stacks** option. Click on the **Next** button. Once clicked, you should see the screen shown in the following screenshot:

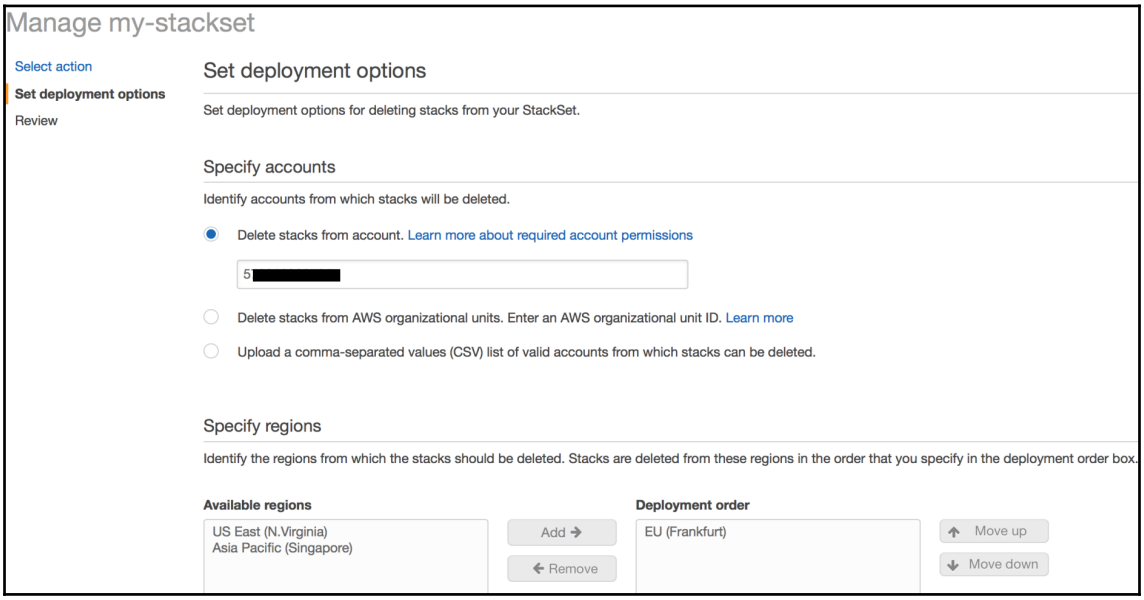


Figure 4.25: Fill in accounts and the regions that will be deleted

4. Fill in your accounts and their region details. Click on the **Next** button to continue.
5. You should see the confirmation screen shown in *Figure 4.26*. Click on the **Delete stacks** button to confirm deletion:

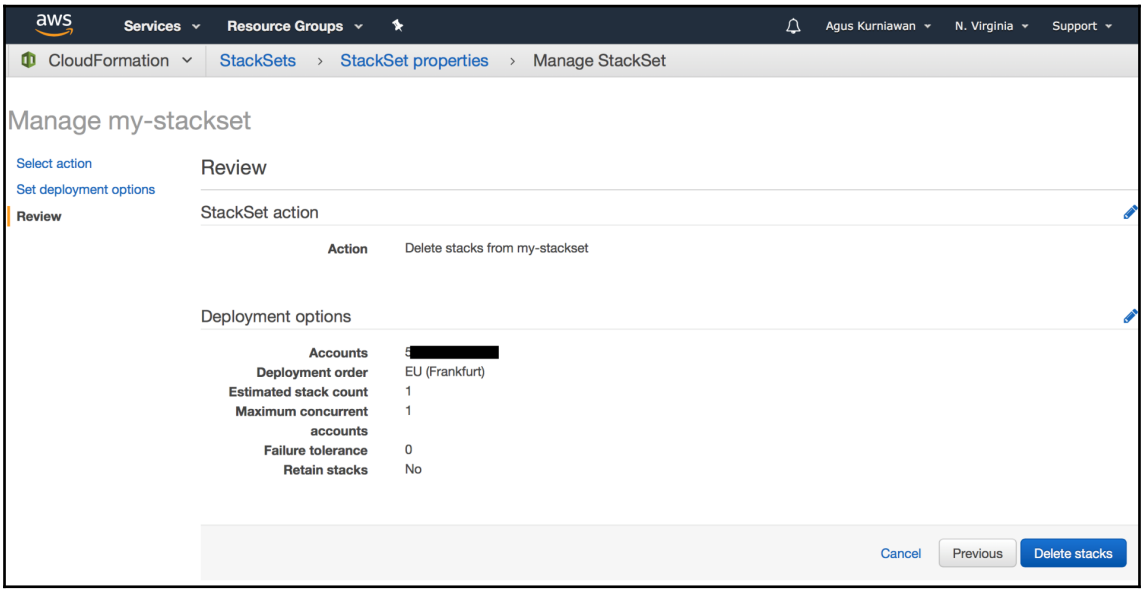


Figure 4.26: Confirming deletion of stack accounts and regions

You can continue to delete other stacks. This is the end of deleting stacks. Next, you can perform StackSets operations using the AWS CLI.

Creating StackSets using the AWS CLI

In the previous section, we built StackSets using the CloudFormation management console. In this section, we'll use the AWS CLI to manage CloudFormation StackSets. A list of AWS CLI commands for CloudFormation can be found here at `https://docs.aws.amazon.com/cli/latest/reference/cloudformation/index.html#cli-aws-cloudformation`.

We'll use the AWS CLI to create a StackSet. Follow these steps:

1. Configure your working region. A list of AWS regions can be found at `https://docs.aws.amazon.com/general/latest/gr/rande.html`.
2. For instance, I changed my region to the `us-east-1` region. You can type the following commands at the Terminal:

```
$ aws configure get region
$ aws configure set region us-east-1
```

3. Once executed, you can verify using the `aws configure get region` command. You can see my program output in the following screenshot:

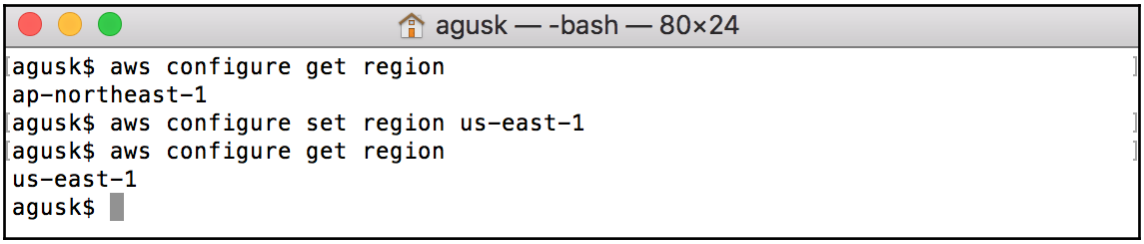


Figure 4.27: Configure the working region on the AWS CLI

4. Create a StackSet using the **Enable AWS Config** template. This template is available at `https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSConfig.yml`.
5. Create a StackSet named `my-stackset`. You can use the `create-stack-set` command. Type this command:

```
$ aws cloudformation create-stack-set --stack-set-name my-
stackset --template-url
https://s3.amazonaws.com/cloudformation-stackset-sample-templat
es-us-east-1/EnableAWSConfig.yml --capabilities CAPABILITY_IAM
```

6. Once executed, you should obtain **StackSetId**, which can be used to retrieve your StackSet information. You can see my **StackSetId** in the following screenshot when I created a StackSet:

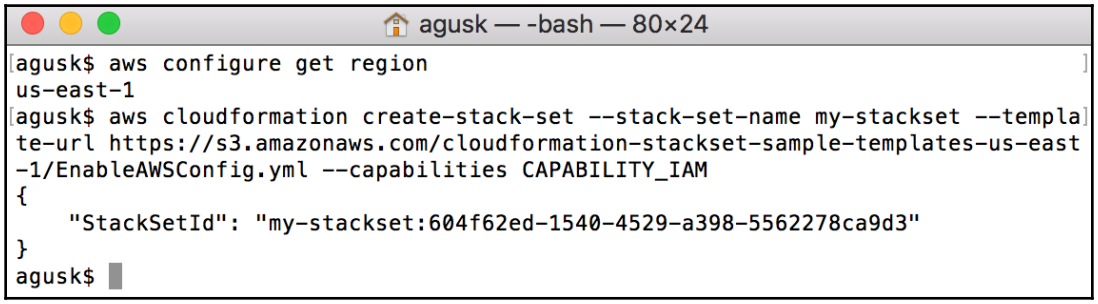


Figure 4.28: Creating a StackSet through the AWS CLI

7. To get a creation status from your StackSet, use the `list-stack-sets` command.
8. If your StackSet has been created successfully, you should see your StackSet status as **ACTIVE**:

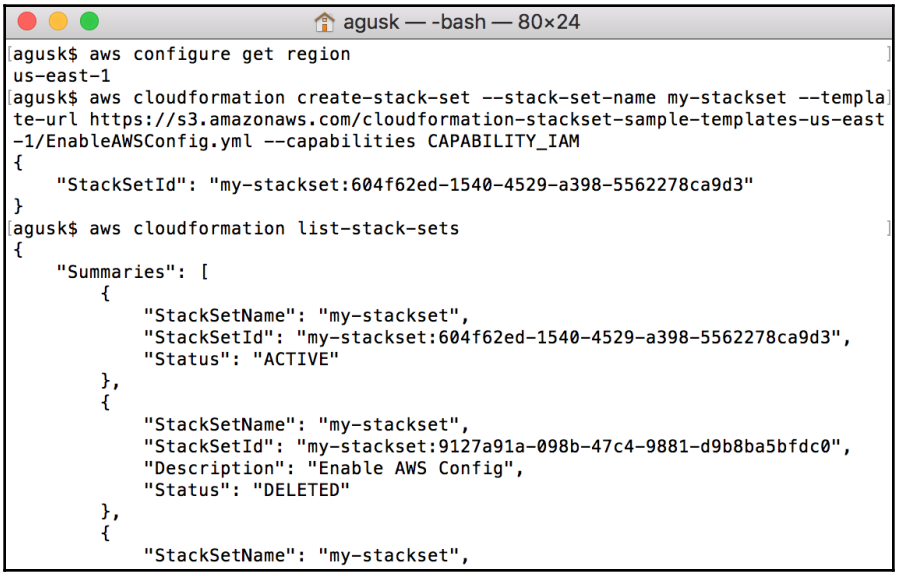


Figure 4.29: Getting a list of StackSets through the AWS CLI

9. We can continue to create stack instances. We define accounts and their regions.
10. Use the `create-stack-instances` command when passing your StackSet name, accounts, and regions.
11. I created a stack instance on my StackSet name—`my-stackset`. I set one account, `123456789012`, with target regions `us-east-1` and `us-east-2`. You can type this command:

```
$ aws cloudformation create-stack-instances --stack-set-name
my-stackset --accounts '["123456789012"]' --regions '["us-
east-1","us-east-2"]' --operation-preferences
FailureToleranceCount=0,MaxConcurrentCount=1
```

12. Once executed, you should get the **OperationId** field. It will be used to check your execution. You can see my program output in the following screenshot:

Figure 4.30: Creating stacks instances

13. To check and verify your operation status, use the `describe-stack-set-operation` command, passing the name of the StackSet and `operation-id`.
14. For instance, I want to check the StackSet named `my-stackset` with the `7c5d3b1e-844c-4e02-9cb2-ad42e3e4d53d` Operation ID. I type the following command:

```
$ aws cloudformation describe-stack-set-operation --stack-set-
name my-stackset --operation-id 7c5d3b1e-844c-4e02-9cb2-
ad42e3e4d53d
```

15. You can see my program output sample in *Figure 4.31*. If you see your Operation ID status as **SUCCEEDED**, it means your stack instance was created:

Figure 4.31: Checking operation status

This is the end of creating a CloudFormation StackSet using the AWS CLI. You can practice more with CloudFormation StackSet development.

Next, we will learn how to edit CloudFormation StackSets.

Editing CloudFormation StackSets

Sometimes, you want to edit your StackSet parameters. You can perform this task. In this section, we'll explore how to modify StackSets using the CloudFormation management console.

You can perform the following steps to edit a StackSet:

1. Open your browser and navigate to the StackSets dashboard, as shown in *Figure 4.17*. Select the StackSet you want to edit.
2. Click on the **Actions** button and select the **Manage stacks in StackSet** menu (see *Figure 4.18*).
3. You should get the screen shown in *Figure 4.32*:

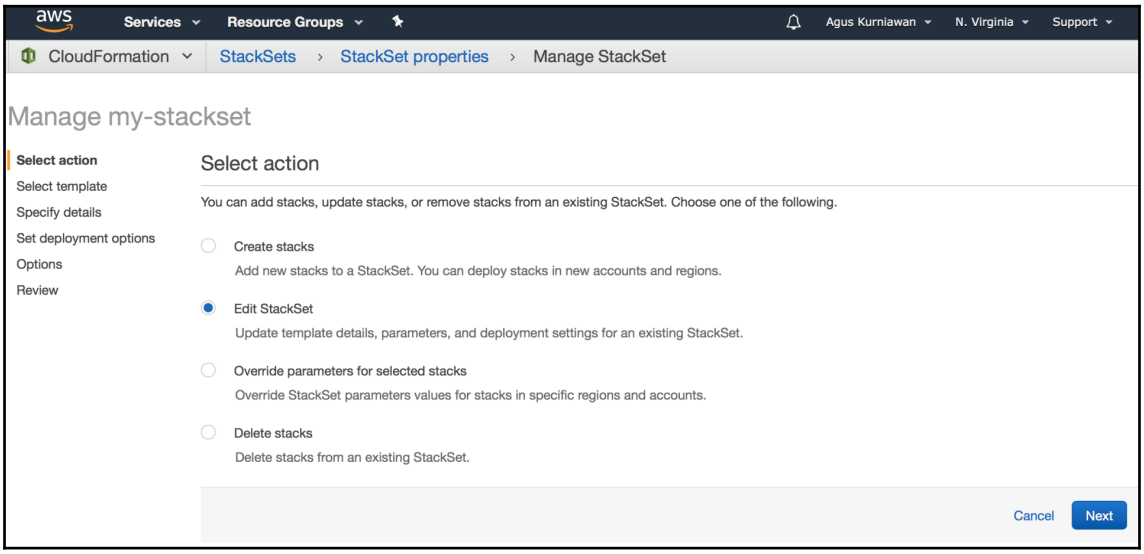


Figure 4.32: Opting to edit a StackSet

4. Select the **Edit StackSet** option. Once done, click on the **Next** button.
5. You should see the screen shown in the following screenshot:

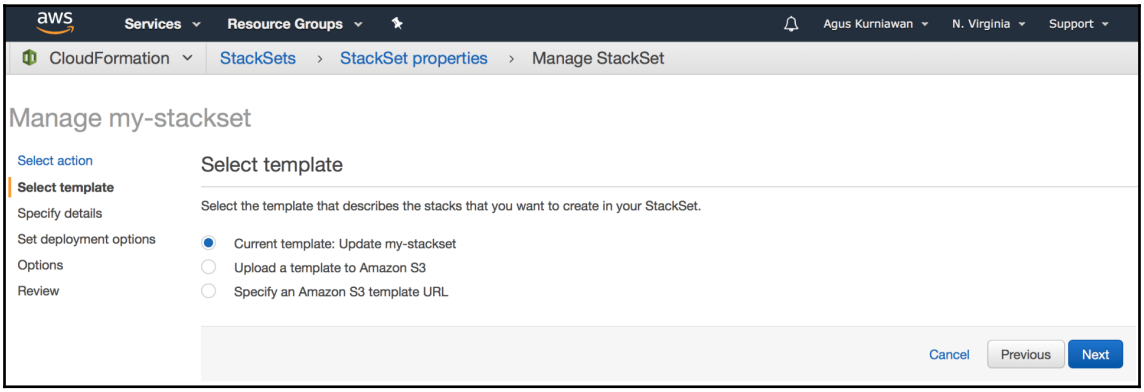


Figure 4.33: Selecting a StackSet template to be edited

6. In this case, we'll modify the current template, so select the **Current template: Update my-stackset** option. Once done, click on the **Next** button.
7. You should get a screen showing your template. For instance, the **Enable AWS Config** template is shown in the following screenshot:

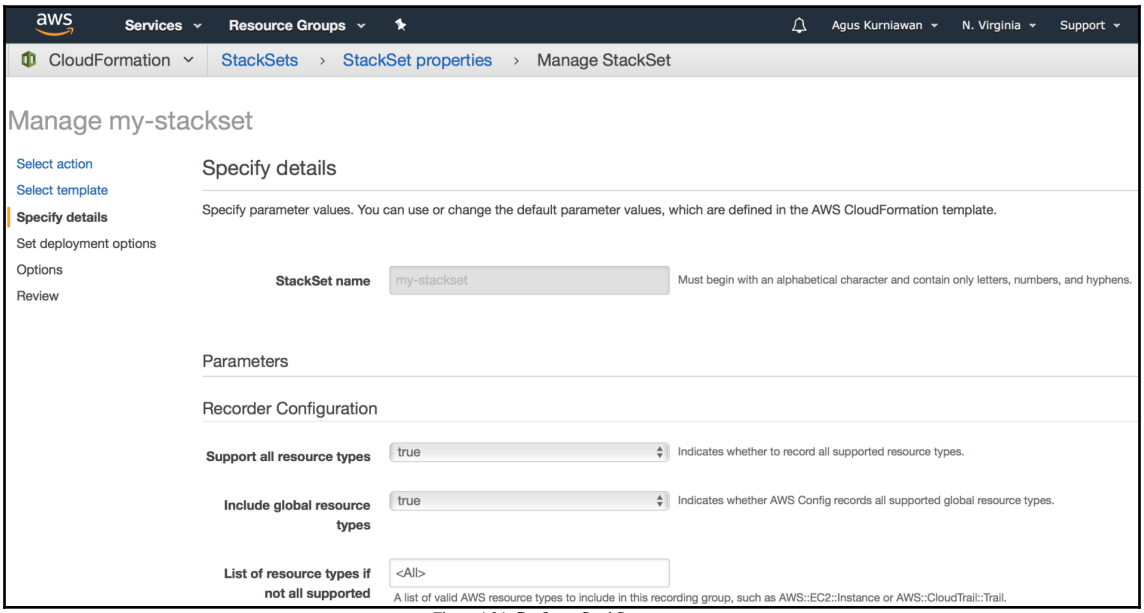


Figure 4.34: Configure StackSet parameters

8. Click on the **Next** button. Follow the next step to confirm and execute your StackSet changes. Once done, your StackSet is modified.

You have learned how to modify your StackSet template. Next, we'll learn how to delete StackSets.

Deleting CloudFormation StackSets

In this section, we will learn how to delete StackSets. To delete a StackSet, you should delete all stacks from the StackSet first. If you don't, you will get errors when deleting StackSets.

You can perform the following steps to delete a StackSet:

1. Open your browser and navigate to the CloudFormation StackSets dashboard at `https://console.aws.amazon.com/cloudformation/stacksets/home`. Select the StackSet you want to delete.
2. Click on the **Actions** button, and you should get the menu shown in the following screenshot:

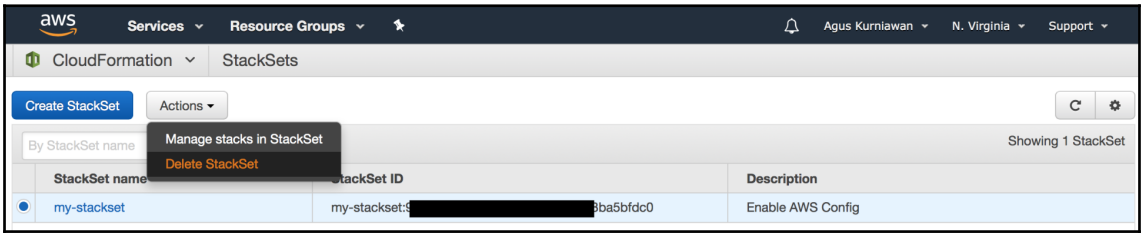


Figure 4.35: A menu for deleting StackSet

3. You should get the confirmation dialog shown in *Figure 4.36*. Click on the **Yes, Delete** button:

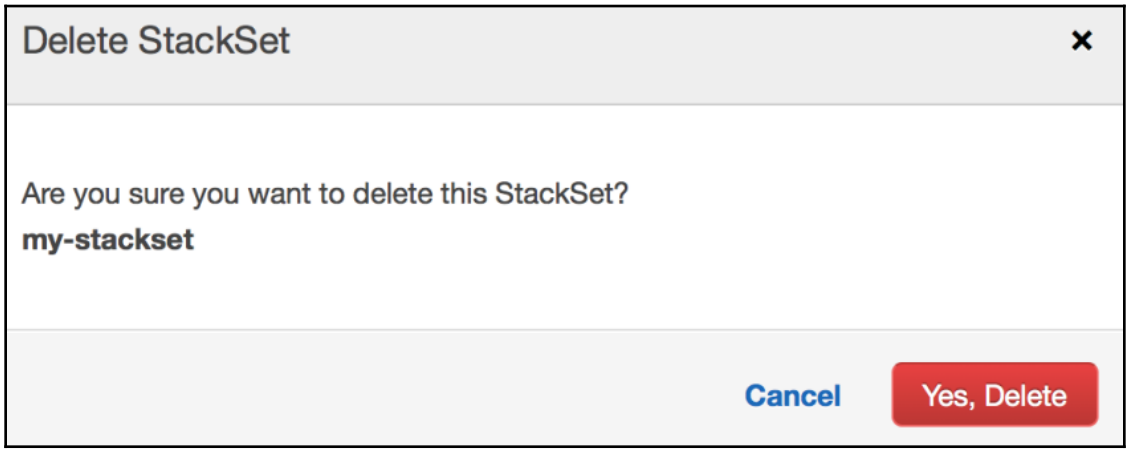


Figure 4.36: Confirmation for deleting a StackSet

4. If you have stacks within the StackSet, you should delete these stacks before deleting the StackSet. If you don't, you'll get errors, as shown in the following screenshot:

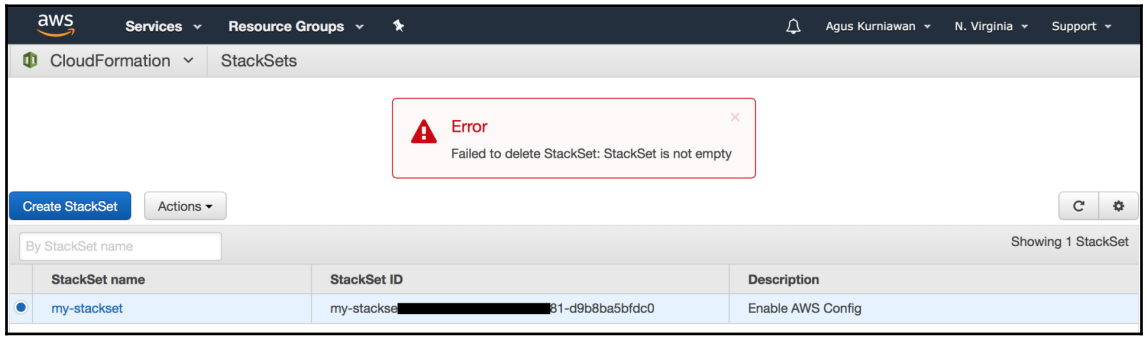


Figure 4.37: Getting errors on deleting a StackSet

5. You can set accounts and target regions to be removed. You can see the account and regions that are being deleted in *Figure 4.38*. Once done, you can back to perform actions on step 2:

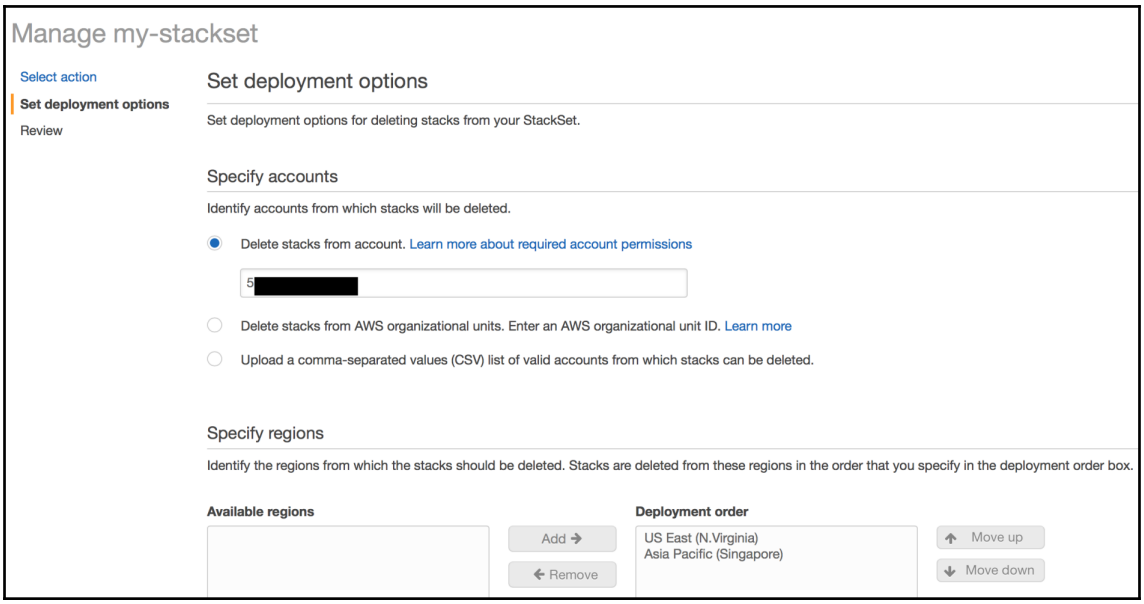


Figure 4.38: Removing all stacks from a StackSet

You have deleted a StackSet from CloudFormation. I recommend you delete StackSets that you no longer use. I also recommend you learn best practices related to CloudFormation StackSets, which you can find at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html`.

Summary

You learned how to create CloudFormation StackSets using the WMC and the AWS CLI. You also learned how to add stacks into a StackSet by attaching accounts and target regions. Then, you learned how to manage stacks from a StackSet, for instance, by editing and deleting them. Finally, you learned how to modify and delete the current StackSet.

In the next chapter, we'll learn how to build AWS Lambda functions using AWS CloudFormation.

Questions

Answer the following questions to evaluate your learning:

1. What is a CloudFormation StackSet?
2. What is the maximum number of stacks in a StackSet?

5
Building Lambda Functions Using AWS CloudFormation

A serverless solution is a problem-solving technique to address issues regarding dynamic infrastructure. In this chapter, we'll explore how to use AWS Lambda to apply a serverless solution. We will build an AWS Lambda through the use of AWS CloudFormation.

The following topics will be covered in this chapter:

- Introducing AWS Lambda
- Building AWS Lambda
- Creating a CloudFormation template for AWS Lambda functions
- Deploying AWS Lambda through CloudFormation
- Deploying AWS Lambda and DynamoDB with CloudFormation
- Deploying AWS Lambda to specific regions through CloudFormation

Introducing AWS Lambda

AWS Lambda is an AWS service that provides a serverless compute service. We can build functions with various programming languages, then AWS Lambda will take care of our infrastructure and its scaling.

We can access other AWS resources from Lambda function. In the following diagram, you can see how a Lambda function accesses AWS resources. Client apps, such as on browser, mobile, or CLI, can invoke our Lambda functions:

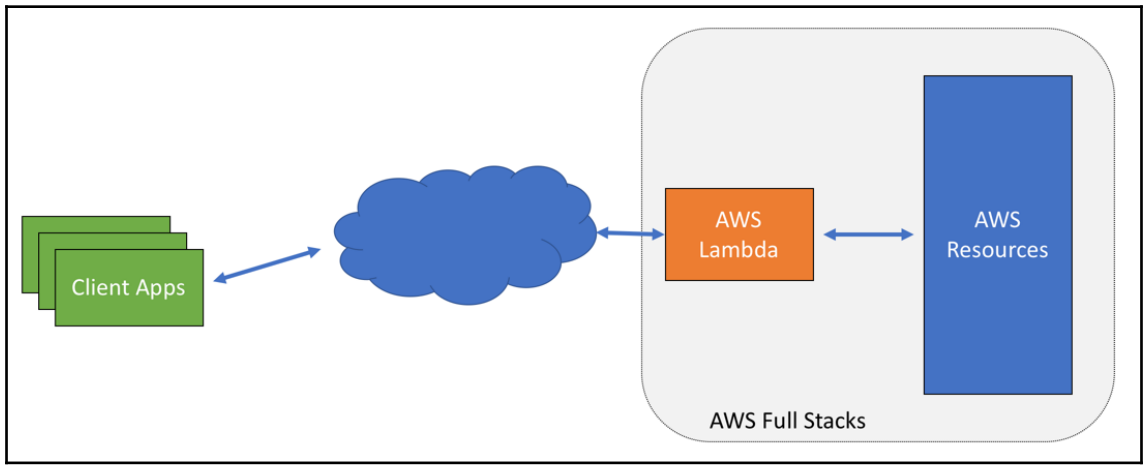


Figure 5.1: Lambda functions in an internet environment

In this chapter, we'll explore Lambda functions with CloudFormation. We'll also combine other AWS resources inside the Lambda function.

Next, we'll build a simple Lambda function in order to understand how the Lambda function works.

Building AWS Lambda

Before we build AWS Lambda through CloudFormation, we'll develop the AWS Lambda application. The objective is to understand AWS Lambda development, especially for readers with no experience of AWS Lambda.

AWS Lambda works in various AWS regions. Visit the AWS Lambda Management Console at `https://console.aws.amazon.com/lambda/`. A sample of the AWS Lambda Management Console window can be seen in *Figure 5.2*. Then, select your region for AWS Lambda development.

For testing, we build a simple AWS Lambda application. We use Node.js for the Lambda function. To implement the AWS Lambda function, we do the following three tasks:

1. Creating the IAM role
2. Developing the AWS Lambda function
3. Testing AWS Lambda

These steps will be followed in the next sections. The AWS Lambda Management Console is shown as follows:

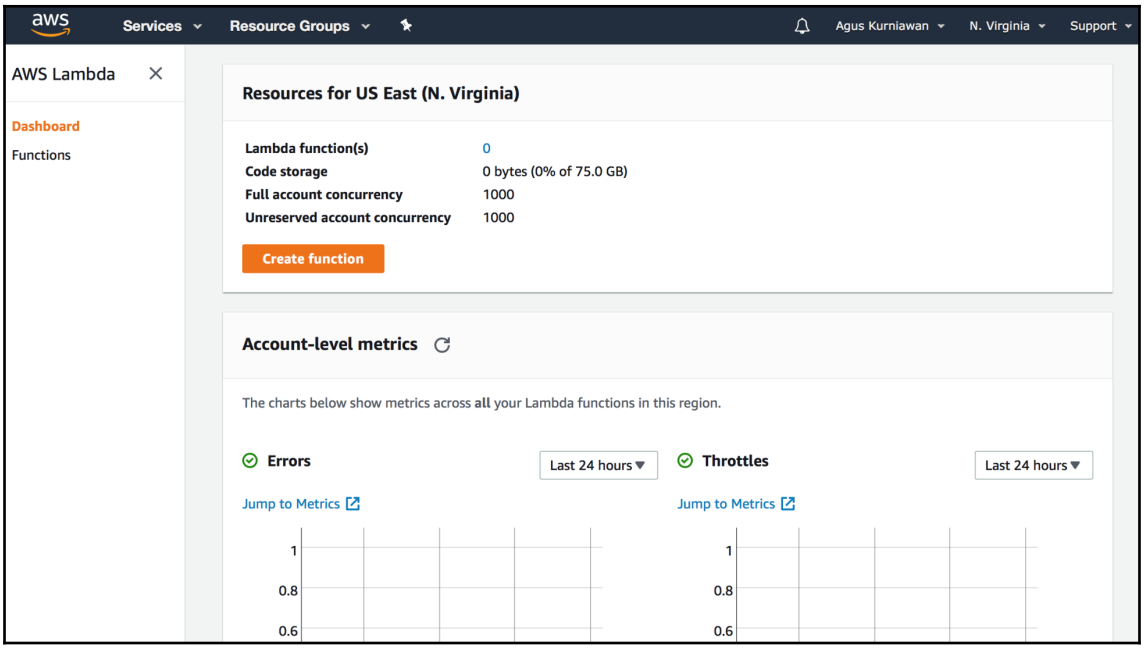


Figure 5.2: AWS Lambda Management Console

Creating the IAM role

Amazon AWS applies security policies on AWS resources. To access the AWS Lambda resources, we should apply a policy in order to create and execute AWS Lambda. In this section, we create a role on IAM roles section.

To create an IAM role, we can follow these steps:

1. Open a browser and navigate to `https://console.aws.amazon.com/iam/`.
2. Open a list of IAM roles by clicking on the **Roles** menu; you should see the following screenshot:

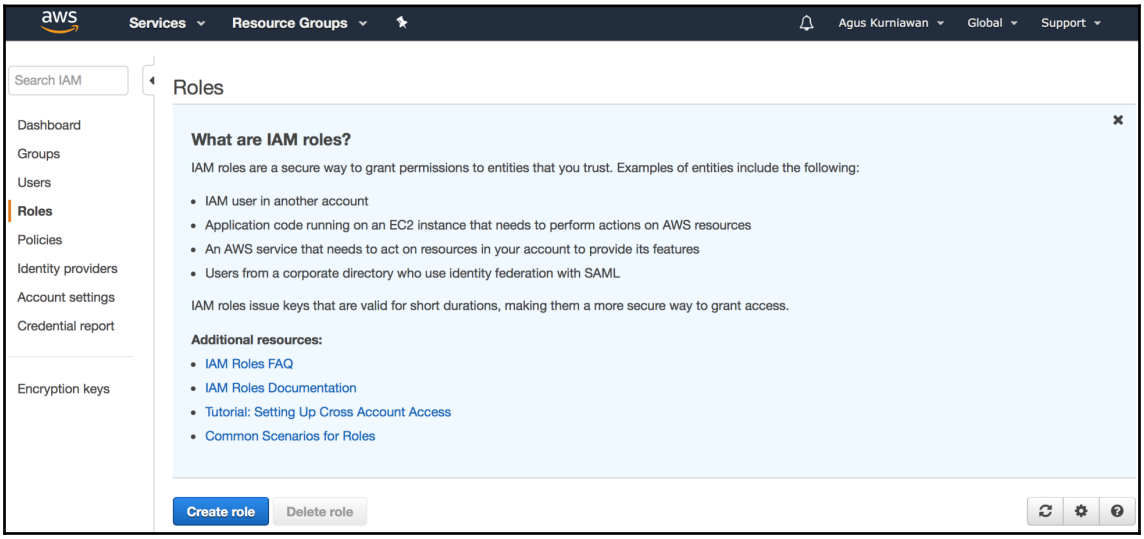


Figure 5.3: AWS IAM Management Console

3. To create a role, click on the **Create role** button to get the screen shown in the *Figure 5.4*.
4. Select the **Lambda** option for the AWS service and then click on the **Next: Permissions** button:

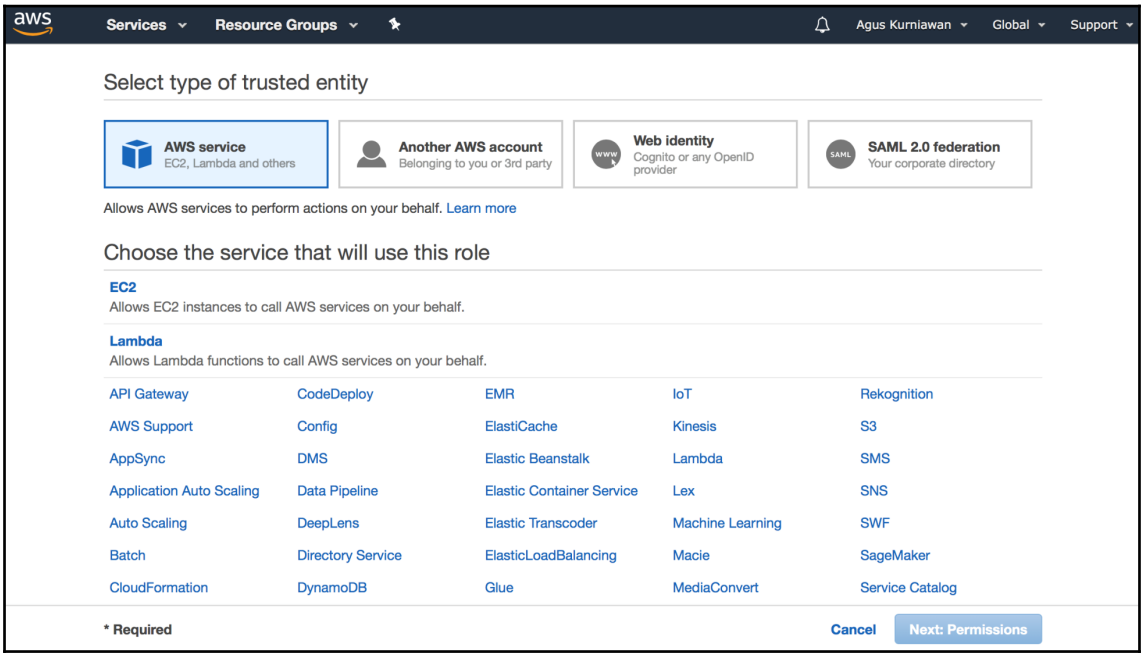


Figure 5.4: Selecting a trusted entity

5. After you click the button, you should get the screenshot shown in *Figure 5.5*. You will be asked to select permission policies.
6. Select the **AWSLambdaFullAccess** policy for full access to AWS Lambda resources and then click on the **Next: Review** button:

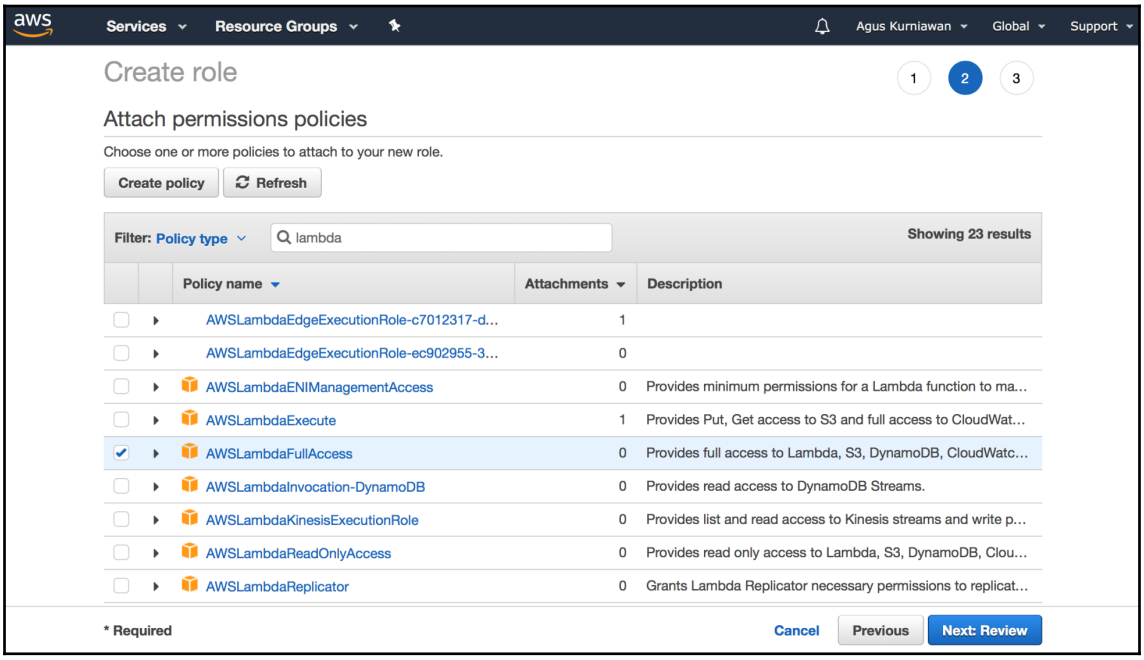


Figure 5.5: Attaching permission policies on the IAM role

7. You get a review screen, which is shown in *Figure 5.6*. Review your input.
8. Fill in your new role, for instance, `my-simple-lambda-role`. Once done, click on the **Create role** button to create the IAM role:

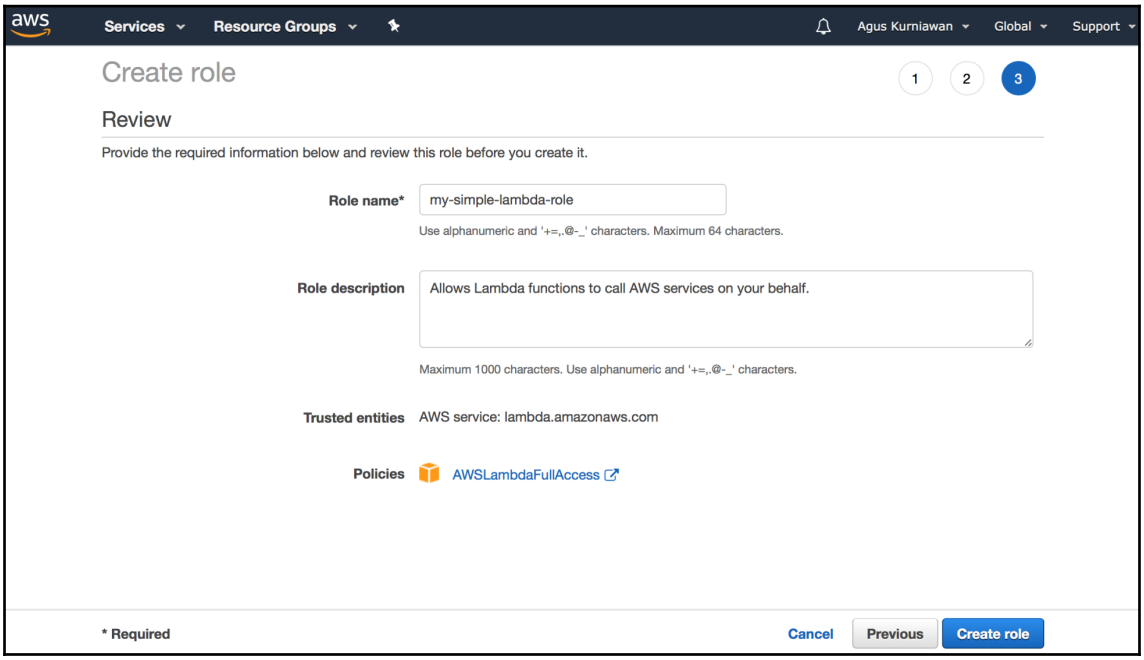


Figure 5.6: Reviewing the created role

9. Once done, you can check your new role on IAM Roles. For instance, here is my IAM role, `my-simple-lambda-role`:

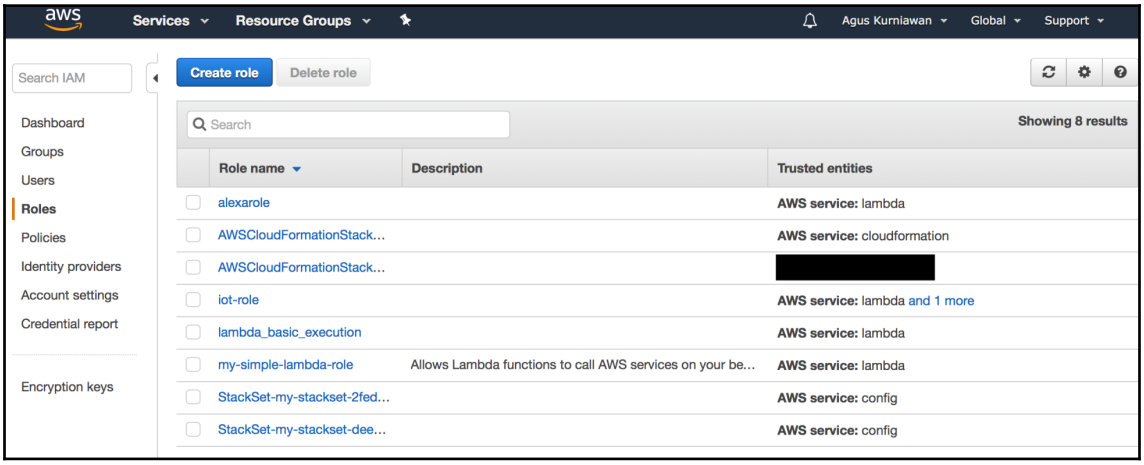


Figure 5.7: A list of IAM roles on IAM Management Console

You have successfully created an IAM role. In the next section, you will develop AWS Lambda.

Developing AWS Lambda using Web Management Console (WMC)

We have created an IAM role that will be applied on AWS Lambda. In this section, we'll develop a simple AWS Lambda function. We use the Node.js application. We receive a message input, `msg`, and then send it to the caller.

Execute the following steps to develop the AWS Lambda function:

1. Open your browser and navigate to the AWS Lambda Management Console at `https://console.aws.amazon.com/lambda/`.
2. Click on the **Create function** button to create a new Lambda function. Once done, you will get the following screen:

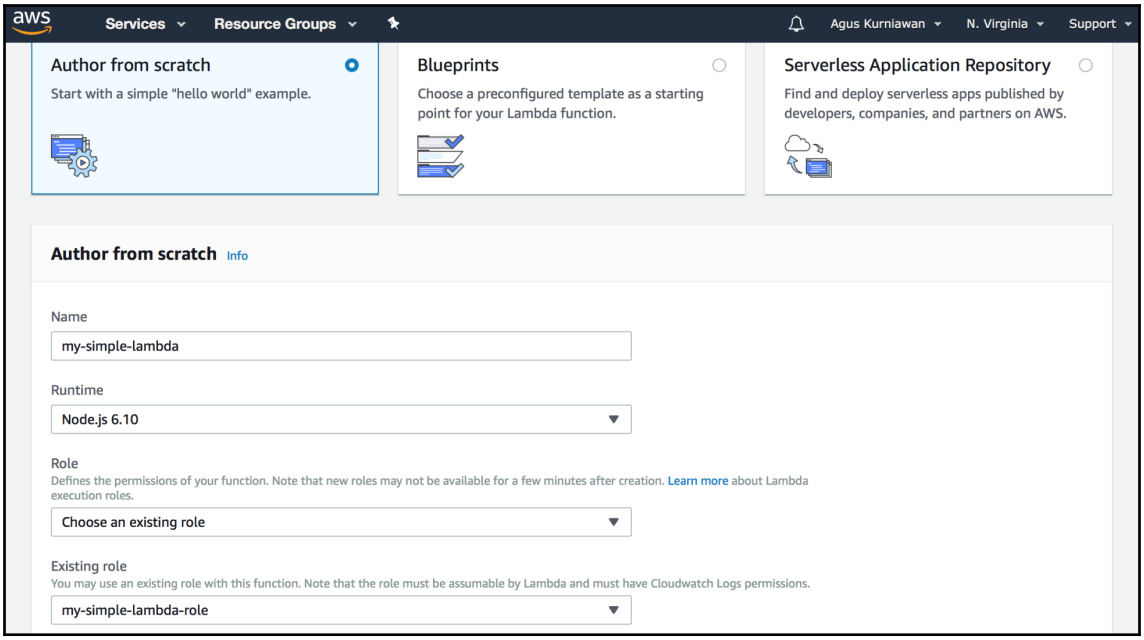


Figure 5.8: Creating AWS Lambda

3. Fill the the Lambda function name, for instance, `my-simple-lambda`.
4. Select Node.js from **Runtime** drop-down. In this demo, I use **Node.js 6.10**. You can use the latest Node.js version.
5. In the **Role** drop-down, select the **Choose an existing role** option, then select the IAM role you created.
6. Once done, click on the **Create function** button to create your Lambda function.
7. You should get a Lambda function with a web editor, as shown in the following screenshot:

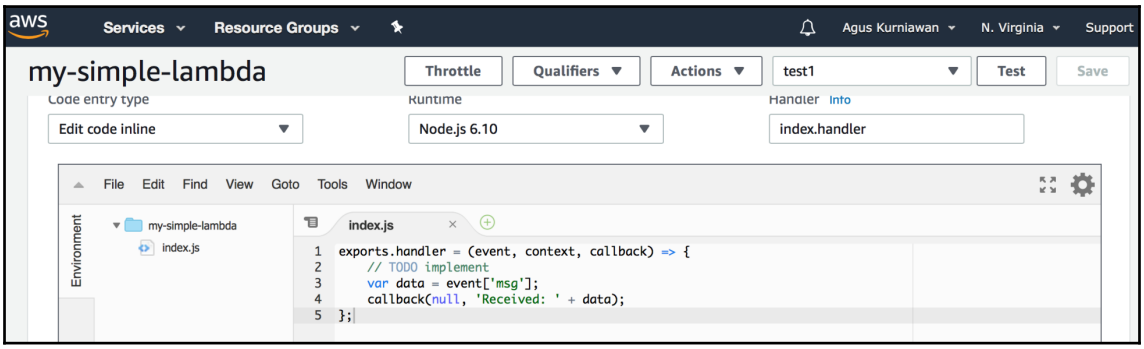


Figure 5.9: Writing code for Lambda function

8. We write our Node.js program inside the web editor. Write the following scripts:

```
exports.handler = (event, context, callback) => {
    var data = event['msg'];
    callback(null, 'Received: ' + data);
};
```

9. Save your function by clicking on the **Save** button and then publish your Lambda function.
10. Click on **Publish new version** on the **Actions** drop-down list. You will be asked to fill in a version description, as shown in the following screenshot:

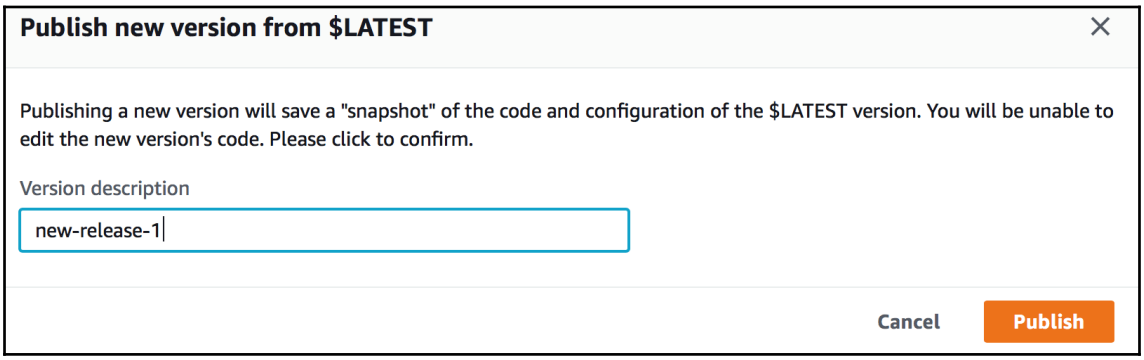


Figure 5.10: Providing a release note on publishing AWS Lambda

Now, your AWS Lambda has been published. Next, we'll test our AWS Lambda.

Testing AWS Lambda

In the previous section, we created and published AWS Lambda using WMC. In this section, we will perform testing for our AWS Lambda function. We are going to test with two methods:

- A test-event tool from AWS Lambda Management Console
- The AWS CLI

The first approach is to use AWS Lambda Management Console. Follow these steps:

1. Open your browser and navigate to AWS Lambda Management Console.
2. Open your Lambda function, then click on the **Test** button (see *Figure 5.9*). You should get the following screen:

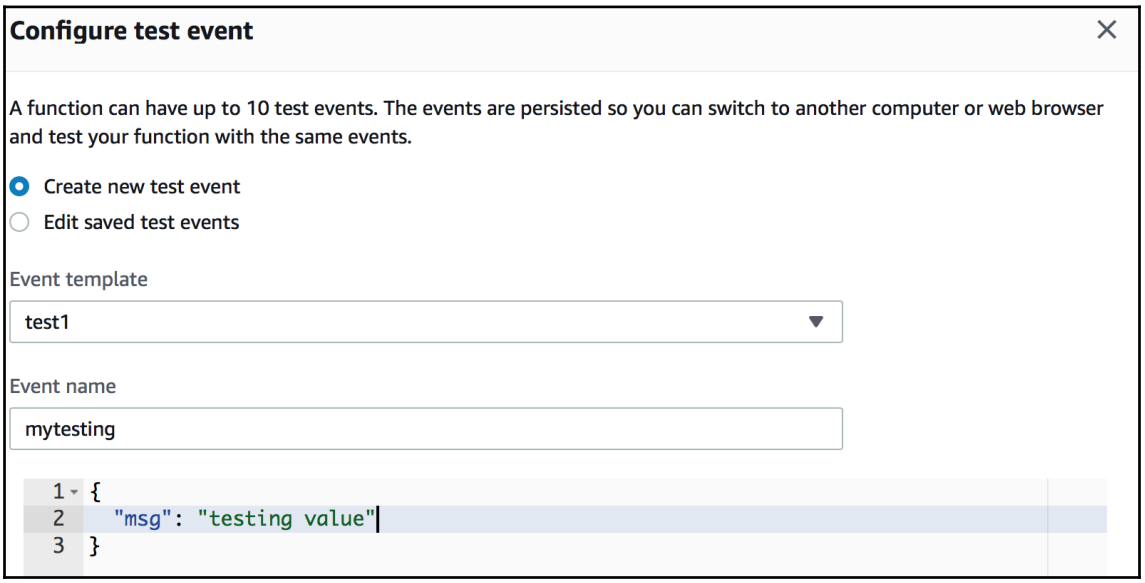


Figure 5.11: Creating the test event

3. Select the **Create new test event** option and fill in your event name.
4. On the code form, you can write the following script:

```
{
  "msg": "testing value"
}
```

5. Once done, click on the Create button.
6. Test your Lambda function by clicking on the **Test** button. If successful, you should get a response from the Lambda, as shown in the following screenshot:

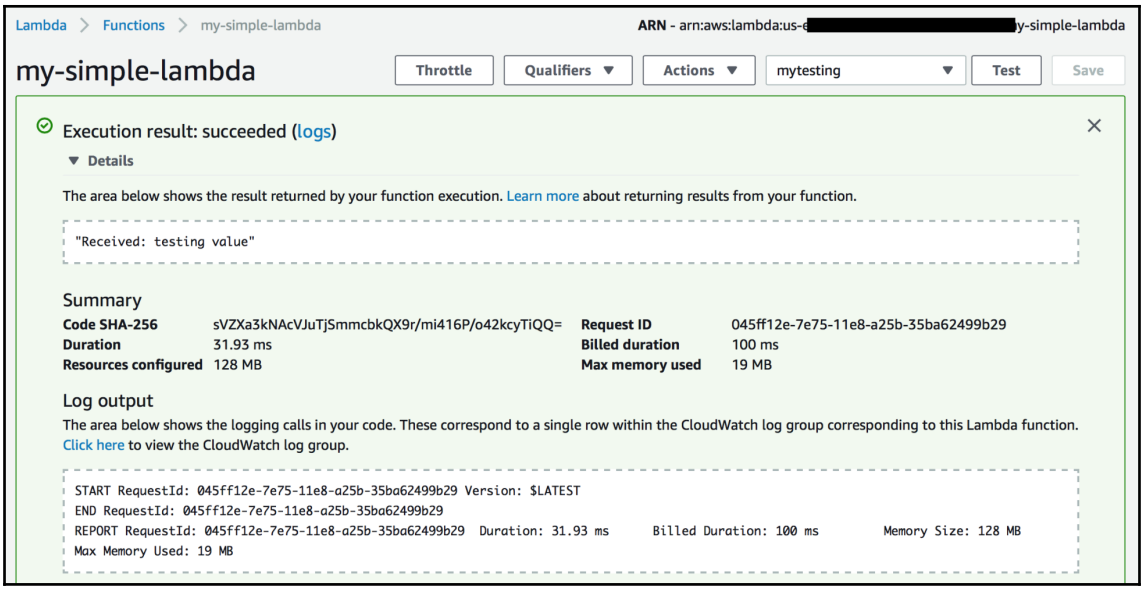


Figure 5.12: Showing a test result

You have finished testing the Lambda function through the management console.

Next, we can perform testing using the AWS CLI. To install the AWS CLI, read the instructions at `https://aws.amazon.com/cli/`. For Windows, you can download it from `https://s3.amazonaws.com/aws-cli/AWSCLI64.msi` for 64-bit, and `https://s3.amazonaws.com/aws-cli/AWSCLI32.msi` for 32-bit:

1. If you run with Linux or macOS, you can install the AWS CLI using `pip`. You can type the following command:

```
$ pip install awscli
```

2. Now, we can perform testing for AWS Lambda; the Lambda function name is `my-simple-lambda`. Type the following command:

```
$ aws lambda invoke --invocation-type RequestResponse --
function-name my-simple-lambda --payload '{"msg": "this is AWS
CLI"}' output.txt
```

3. Once successful, you should see the following response output:

```
[agusk$ aws lambda invoke --invocation-type RequestResponse --function-name my-si]
mple-lambda --payload '{"msg": "this is AWS CLI"}' output.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
agusk$
```

Figure 5.13: Testing AWS Lambda from the AWS CLI

4. Open the output file, `output.txt`, using your text editor. For instance, I use `nano` and type the following command:

```
$ nano output.txt
```

5. You should see the contents of the output file, which is a response from AWS Lambda. For instance, here is my output file:

```
  GNU nano 2.0.6                File: output.txt

"Received: this is AWS CLI"


                              [ Read 1 line ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 5.14: Opening the output file by invoking the Lambda

You have finished testing the AWS Lambda function. Next, we'll build AWS Lambda through AWS CloudFormation.

CloudFormation template for AWS Lambda functions

AWS CloudFormation provides an **Infrastructure as Code** (**IaC**) solution to build a dynamic infrastructure. In the previous section, we learned how to build AWS Lambda. Now, we continue our journey with CloudFormation. We will build AWS Lambda through AWS CloudFormation.

The CloudFormation template for AWS Lambda can be found at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html`. The template can be described in JSON as follows:

```
{
  "Type" : "AWS::Lambda::Function",
  "Properties" : {
    "Code" : Code,
    "DeadLetterConfig" : DeadLetterConfig,
    "Description" : String,
    "Environment" : Environment,
    "FunctionName" : String,
    "Handler" : String,
    "KmsKeyArn" : String,
    "MemorySize" : Integer,
    "ReservedConcurrentExecutions" : Integer,
    "Role" : String,
    "Runtime" : String,
    "Timeout" : Integer,
    "TracingConfig" : TracingConfig,
    "VpcConfig" : VPCConfig,
    "Tags" : [ Resource Tag, ... ]
  }
}
```

For YAML, we can define the CloudFormation template for AWS Lambda as follows:

```
Type: "AWS::Lambda::Function"
Properties:
  Code:
    Code
  DeadLetterConfig:
    DeadLetterConfig
```

```
Description: String
Environment:
  Environment
FunctionName: String
Handler: String
KmsKeyArn: String
MemorySize: Integer
ReservedConcurrentExecutions: Integer
Role: String
Runtime: String
Timeout: Integer
TracingConfig:
  TracingConfig
VpcConfig:
  VPCConfig
Tags:
  Resource Tag
```

Not all attributes from the CloudFormation template of the Lambda are required. The following are required attributes:

- `Code`: The contents of the source code from your Lambda function. You can put the source code as inline text. You also can put it into Amazon S3.
- `Handler`: The name of the Lambda function (within your source code) that the Lambda calls to start running your code.
- `Role`: An IAM role that is applied to execute the AWS Lambda function. It's the **Amazon Resource Name** (**ARN**) of the AWS Identity. Make sure your IAM role has access rights to execute the Lambda function, including your other required resources.
- `Runtime`: A runtime environment for the Lambda function that you are uploading. For instance, it could be Python or Node.js.

Next, we'll deploy AWS Lambda using CloudFormation.

Deploying Lambda functions using AWS CloudFormation

In this section, we'll learn how to deploy Lambda functions using AWS CloudFormation. Our Lambda function scenario is similar to the `my-simple-lambda` function from the previous demo.

First, we make a CloudFormation template to create AWS Lambda. Then, we upload the template to CloudFormation. Then, we get AWS Lambda running. We'll perform these tasks in the next sections.

Creating a CloudFormation template for the Lambda function

Before we deploy the Lambda function through CloudFormation, we should prepare the CloudFormation template. We can write the template in JSON or YAML.

In this demo, we create the Lambda function as in the previous demo. Our Lambda function receives the `msg` input and then passes it to the function output. We also define a parameter, `LambdaFunctionName`, to get the Lambda function name. For the IAM role, we create a new IAM role, called `TestLambdaExecutionRole`, that will be passed to the Lambda function.

The following is a complete CloudFormation template in JSON:

```
{
    "Description" : "This is a simple Lambda function is Node.js",
    "Parameters": {
        "LambdaFunctionName":{
            "Description": "AWS Lambda function name",
            "Type": "String"
        }
      },
    "Resources":{
        "TestLambdaFunction" : {
            "Type" : "AWS::Lambda::Function",
            "Properties" : {
                "FunctionName" : {
                    "Ref": "LambdaFunctionName"
                },
                "Handler" : "index.handler",
                "Role" : { "Fn::GetAtt" : ["TestLambdaExecutionRole",
"Arn"] },
                "Code" : {
                    "ZipFile" : { "Fn::Join" : [ "\n", [
                        "exports.handler = (event, context, callback) =>
{",
                        " var data = event['msg'];",
                        " callback(null, 'Received: ' + data);",
                        "}"
                    ]]}
```

```
                },
                "Timeout" : "10",
                "Runtime" : "nodejs6.10"
            }
        },
        "TestLambdaExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "lambda.amazonaws.com"
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ]
                        }
                    ]
                },
                "Path": "/"
            }
        }
    }
}
```

Save these scripts into a file called `Lambda-CloudFormation.json`.

Here is the CloudFormation template in YAML:

```
Description: This is a simple Lambda function is Node.js
Parameters:
  LambdaFunctionName:
    Description: AWS Lambda function name
    Type: String
Resources:
  TestLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName:
        Ref: LambdaFunctionName
      Handler: index.handler
      Role:
        Fn::GetAtt:
```

```
        - TestLambdaExecutionRole
        - Arn
      Code:
        ZipFile:
          Fn::Join:
          - "\n"
          - - exports.handler = (event, context, callback) => {
            - " var data = event['msg'];"
            - " callback(null, 'Received: ' + data);"
            - "}"
      Timeout: '10'
      Runtime: nodejs6.10
  TestLambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal:
            Service:
            - lambda.amazonaws.com
          Action:
          - sts:AssumeRole
      Path: "/"
```

Save these scripts into a file called `Lambda-CloudFormation.yaml`. After we have created the CloudFormation template file, we will deploy it to CloudFormation.

Deploying AWS Lambda to CloudFormation

We continue to deploy the AWS Lambda function to CloudFormation. We upload the `Lambda-CloudFormation.json` or `CloudFormation.yaml` file for the CloudFormation template. In this section, we'll create a CloudFormation stack and then put the template file into the stack.

To implement our demo, perform the following steps:

1. Open your browser and navigate to AWS CloudFormation at `https://console.aws.amazon.com/cloudformation/home`. You should get the following screen:

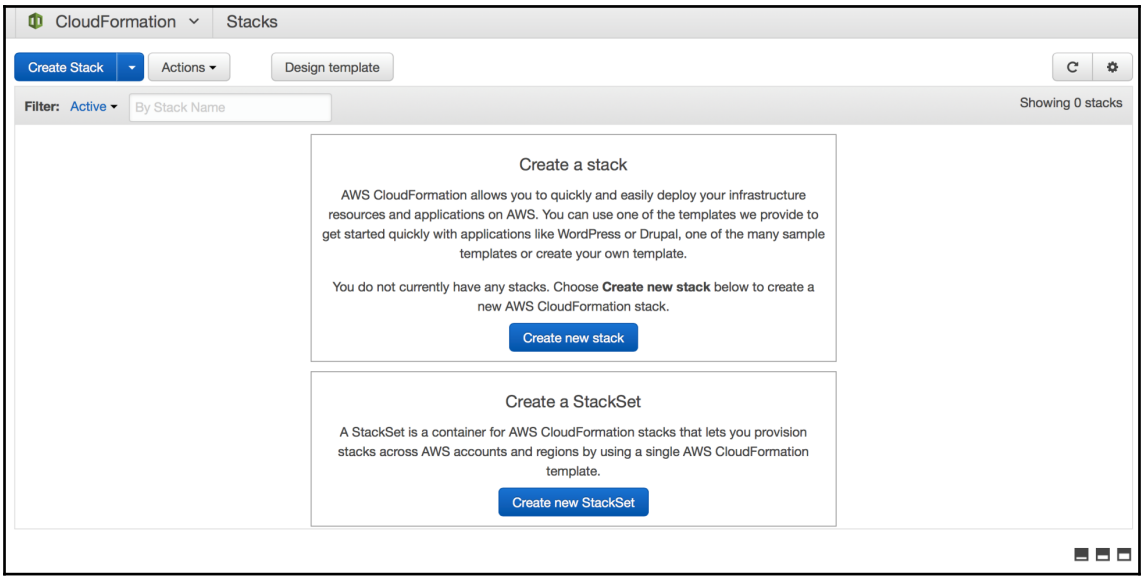


Figure 5.15: A form from the CloudFormation management console

2. To create a new stack, click on the **Create new stack** button; you'll get the screen shown in *Figure 5.16*.
3. Select the **Upload a template to Amazon S3** option.
4. Select our CloudFormation template file, which has already been created, and then click on the **Next** button:

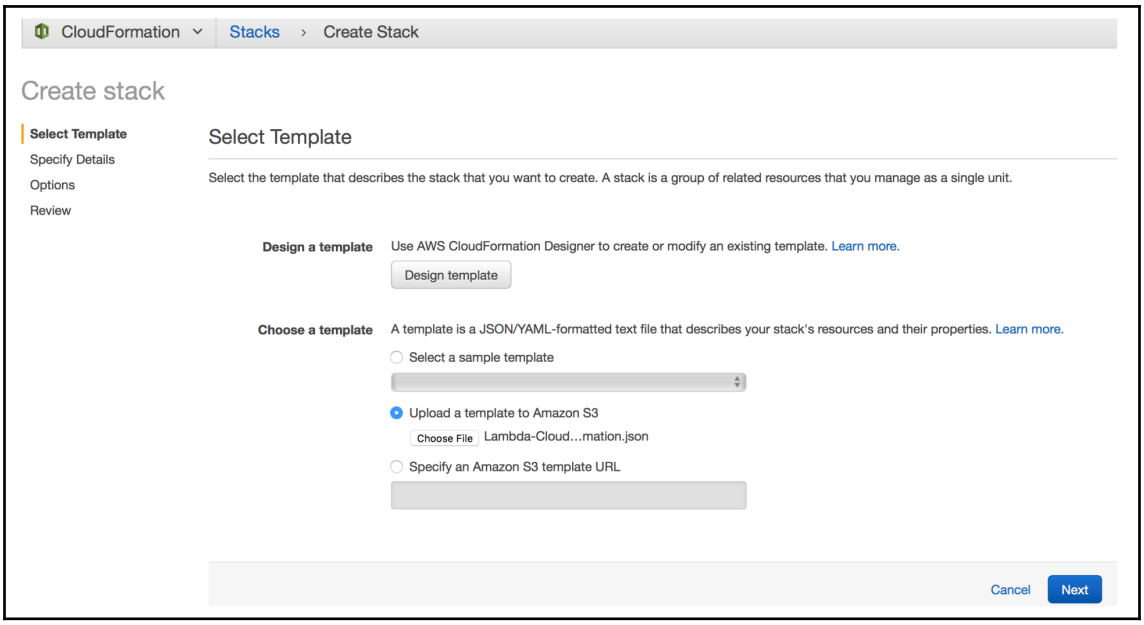


Figure 5.16: Selecting the CloudFormation template

5. You should get the screen shown in *Figure 5.17*. Fill in the stack name and function name. For instance, I set my stack name as `test-lambda` and my function name as `my-lambda-cloudformation`. Once done, click on the **Next** button:

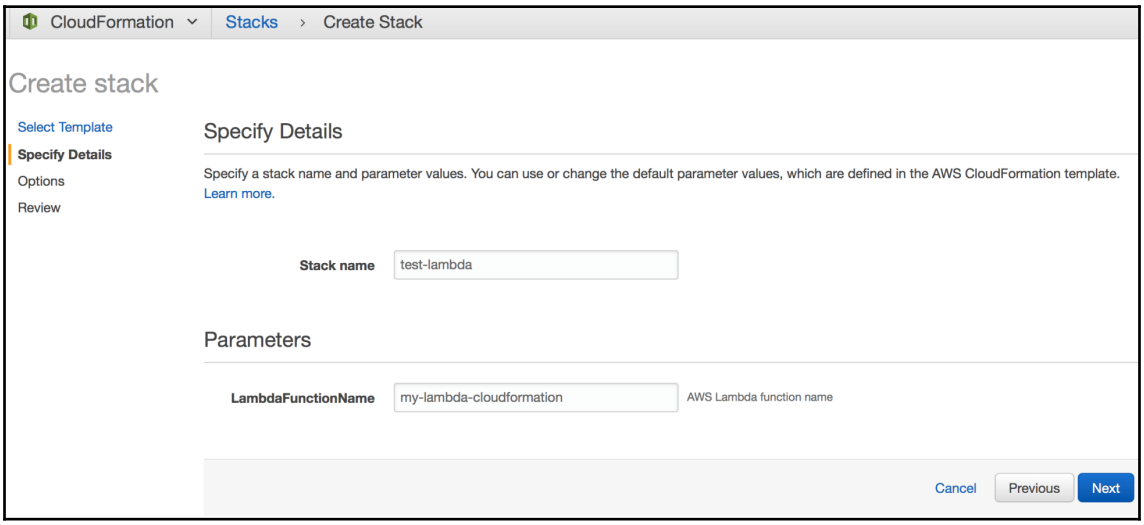


Figure 5.17: Filling in the stack and Lambda function names

6. You should get the following screen. We don't do anything on this screen. Just click on the **Next** button:

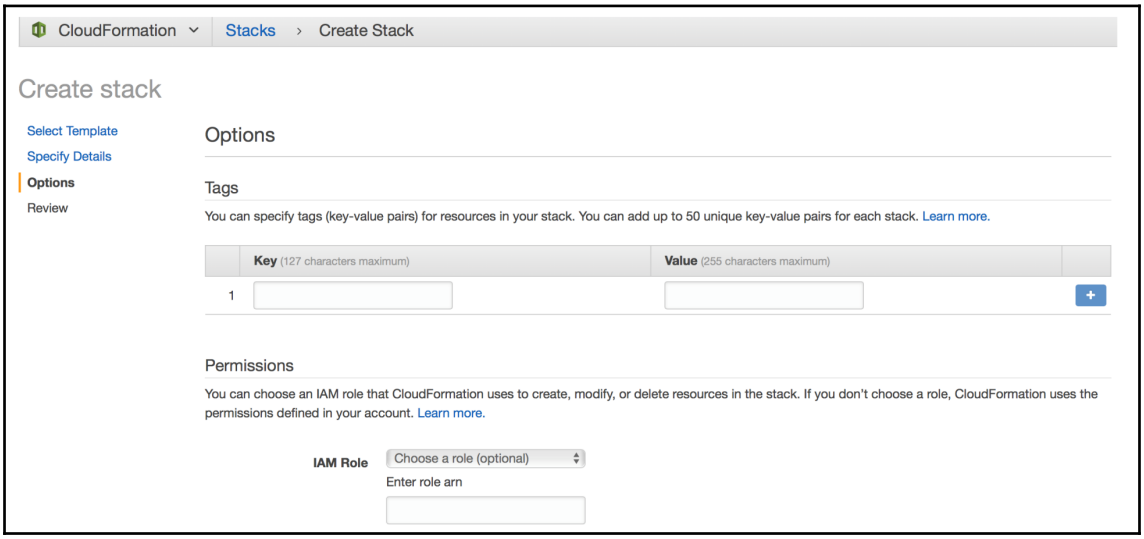


Figure 5.18: Setting the CloudFormation parameters

7. You should get the following review screen. Review all input and then click on the **Create** button to create a stack:

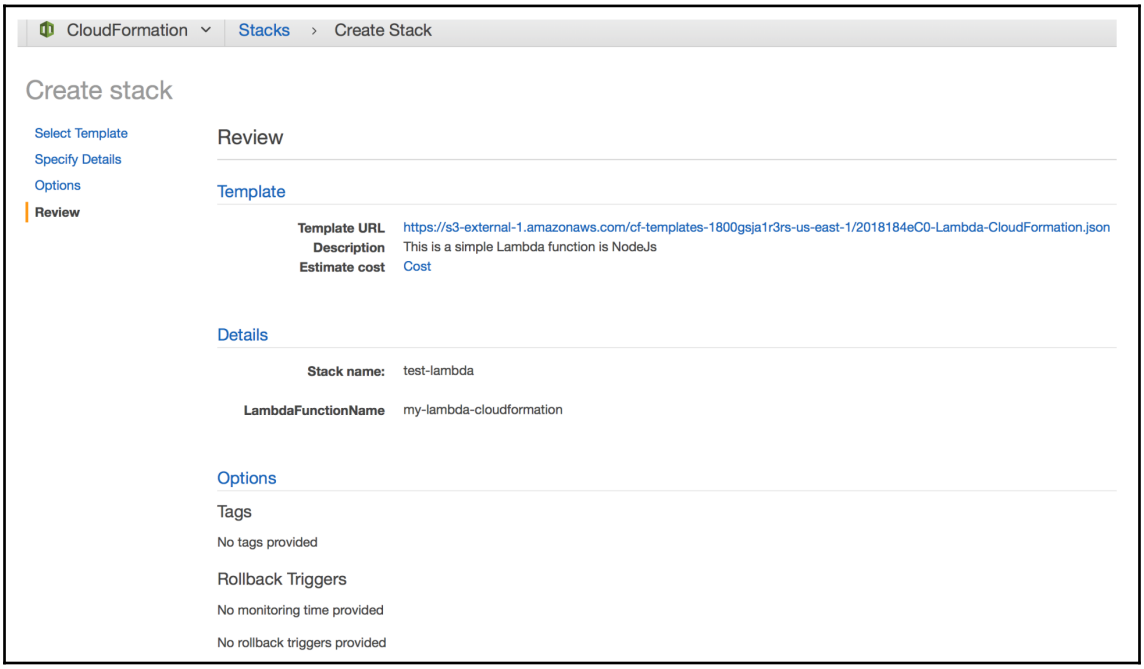


Figure 5.19: Review for creating a stack

8. After you have clicked the **Create** button, you should get the CloudFormation dashboard. Ensure your stack has already been created with the **CREATE_COMPLETE** status. The following screenshot confirms my stack was created:

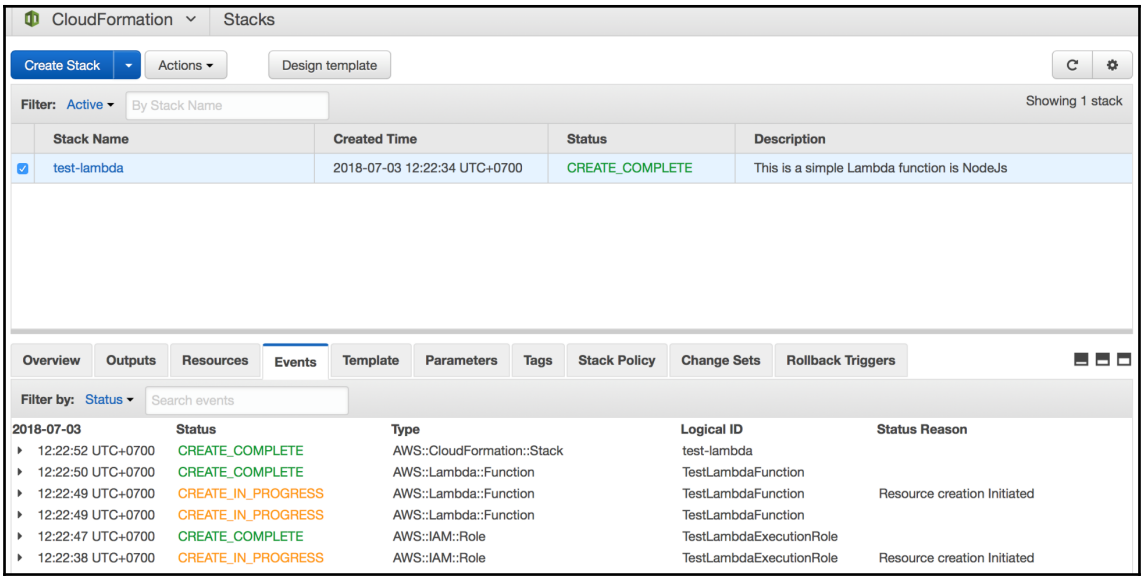


Figure 5.20: Showing the stack status

9. You also can verify your Lambda function on AWS Lambda Management Console.
10. Open your browser and navigate to `https://console.aws.amazon.com/lambda/home`. You should see your Lambda function that was created through CloudFormation:

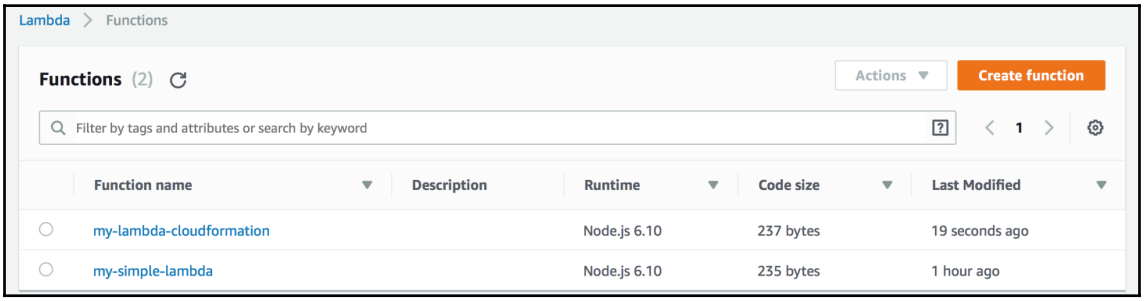


Figure 5.21: Showing a list of AWS Lambda

Now we can test using the AWS CLI. For instance, the Lambda function name is `my-lambda-cloudformation`. We can perform this test by typing the following command:

```
$ aws lambda invoke --invocation-type RequestResponse --function-name my-
lambda-cloudformation --payload '{"msg": "this is AWS CLI"}' output-
lambda.txt
```

If successful, you get the following response output:

```
agusk$ aws lambda invoke --invocation-type RequestResponse --function-name my-la
mbda-cloudformation --payload '{"msg": "this is AWS CLI"}' output-lambda.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
agusk$
```

Figure 5.22: Calling the AWS Lambda function from the AWS CLI

If you have security problems when invoking the Lambda function from the AWS CLI, verify your account permissions. You can check on the IAM management console at `https://console.aws.amazon.com/iam/`. Your account should have **AWSLambdaExecute** and/or **AWSLambdaRole** permissions.

After invoking the Lambda function, open the output file so you get the contents of the output file. Type the following command:

```
$ nano output-lambda.txt
```

You should see the following response output:

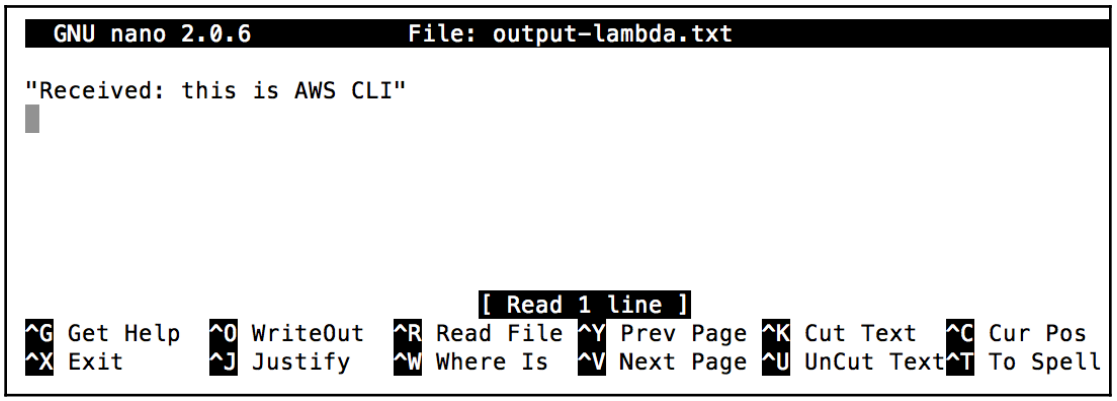


Figure 5.23: Opening the output file, output-lambda.txt

You have learned how to build AWS Lambda using CloudFormation. Next, we'll explore the AWS Lambda function with the DynamoDB database through the CloudFormation template.

CloudFormation for AWS Lambda and DynamoDB

Sometimes we want to build a Lambda function with storage capabilities. Amazon AWS provides storage services that we can integrate with AWS Lambda. In this section, we'll explore how to use DynamoDB in AWS Lambda.

AWS DynamoDB is a storage service from AWS. It's storage based on NoSQL. You can review DynamoDB's features on its official website at `https://aws.amazon.com/dynamodb/`. AWS DynamoDB provides various SDK APIs to enable you to access DynamoDB in your own programs, such as Java, JavaScript, Node.js, .NET, PHP, Python, and Ruby.

Technically, DynamoDB can be operated easily. You can use the DynamoDB Management Console at `https://console.aws.amazon.com/dynamodb/`. Then, you can create a table with key attributes. You can deploy DynamoDB in various regions. It takes benefits in accessibility from your customers.

In this section, we'll build a Lambda function to insert data into DynamoDB. We implement a Lambda function using Node.js through CloudFormation for the deployment method:

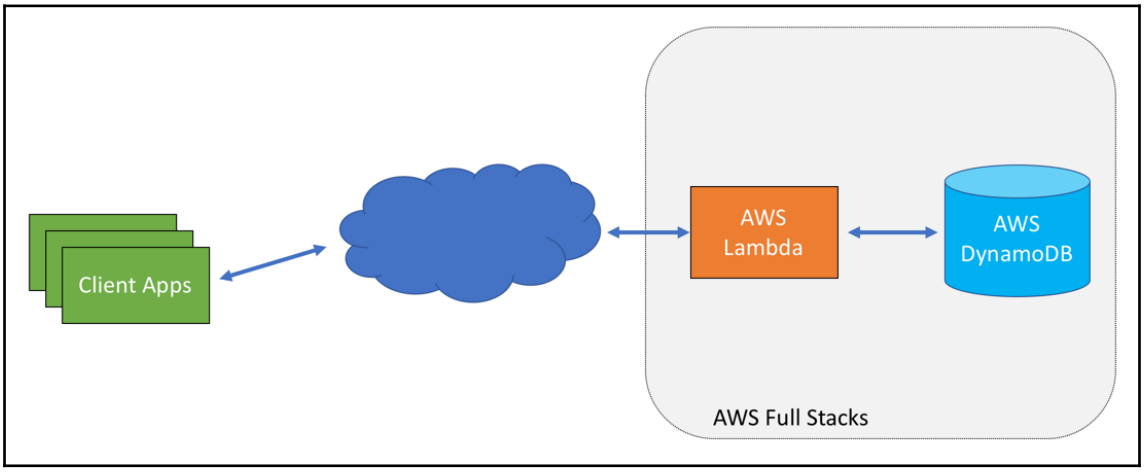


Figure 5.24: Lambda and DynamoDB interaction

To implement our demo, we will perform the following tasks:

1. Creating a CloudFormation template for Lambda and DynamoDB
2. Deploying the CloudFormation template
3. Configuring the Lambda-invoking policy
4. Testing the Lambda function

These tasks will be explored in the next sections.

Creating the CloudFormation template for AWS DynamoDB

The CloudFormation template for AWS DynamoDB can be found at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-dynamodb-table.html`. The template can be described in JSON as follows:

```
{
  "Type" : "AWS::DynamoDB::Table",
  "Properties" : {
    "AttributeDefinitions" : [ AttributeDefinition, ... ],
    "GlobalSecondaryIndexes" : [ GlobalSecondaryIndexes, ... ],
    "KeySchema" : [ KeySchema, ... ],
    "LocalSecondaryIndexes" : [ LocalSecondaryIndexes, ... ],
    "PointInTimeRecoverySpecification" : PointInTimeRecoverySpecification,
    "ProvisionedThroughput" : ProvisionedThroughput,
    "SSESpecification" : SSESpecification,
    "StreamSpecification" : StreamSpecification,
    "TableName" : String,
    "Tags" : [ Resource Tag, ... ],
    "TimeToLiveSpecification" : TimeToLiveSpecification
  }
}
```

We can define the template in YAML as follows:

```
Type: "AWS::DynamoDB::Table"
Properties:
  AttributeDefinitions:
    - AttributeDefinition
  GlobalSecondaryIndexes:
    - GlobalSecondaryIndexes
  KeySchema:
    - KeySchema
  LocalSecondaryIndexes:
    - LocalSecondaryIndexes
  PointInTimeRecoverySpecification:
    PointInTimeRecoverySpecification
  ProvisionedThroughput:
    ProvisionedThroughput
  SSESpecification:
    SSESpecification
  StreamSpecification:
    StreamSpecification
  TableName: String
  Tags:
    - Resource Tag
```

```
TimeToLiveSpecification:
  TimeToLiveSpecification
```

You should define three required attributes—`AttributeDefinitions`, `KeySchema`, and `ProvisionedThroughput`:

- `AttributeDefinitions`: This consists of a list of attributes that describe the key schema for the table and indexes.
- `KeySchema`: This consists of attributes that make up the primary key for the table. You can define the KeySchema type in `HASH` or `RANGE`.
- `ProvisionedThroughput`: This describes the throughput values, which are values for the `ReadCapacityUnits` and `WriteCapacityUnits` attributes.

Next, we'll create a CloudFormation template to build Lambda and DynamoDB resources.

Building a CloudFormation template for Lambda and DynamoDB

To build a CloudFormation template for Lambda and DynamoDB, we can modify our previous CloudFormation template for Lambda. We add DynamoDB resources and a policy. We replace our Lambda function in order to access DynamoDB.

Next, we'll start to create the Lambda function.

Accessing DynamoDB from Lambda functions

Our scenario in the Lambda function is to get JSON data from the parameter input of the Lambda function, and then insert it into DynamoDB.

We can use the `DynamoDB` object from our Node.js application through the AWS SDK. After we get data from the Lambda function, we can insert the data into DynamoDB using the `putItem()` API. You can read more about this API at `https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html`.

For instance, we insert data into the DynamoDB table, called `mydynamodb`. We can implement our Lambda function as follows:

```
var AWS = require('aws-sdk');
var ddb = new AWS.DynamoDB();
exports.handler = (event, context, callback) => {
    var params = {
        TableName: 'mydynamodb',
        Item: {
           'id': {S:new Date().getTime().toString()},
           'email': {S:event.email},
           'name': {S:event.name},
           'country' : {S:event.country},
           'age' : {N:event.age},
        }
    };
    ddb.putItem(params, function(err, data) {
       if (err) {
          callback(err, 'Error');
       } else {
          callback(null, 'Insert data was successful');
       }
    });
}
```

This Lambda function will return the `Insert data was successful` message in the event of a successful operation. Otherwise, we get an `Error` message with a detailed error description.

Creating the CloudFormation template

We modify our CloudFormation template for the Lambda from the previous section. We replace the content of the `Code` attribute with our Lambda function code. We also define the DynamoDB resource, called `myDynamoDBTable`.

Since our Lambda function accesses DynamoDB, we should configure an additional policy inside `TestLambdaExecutionRole`. We add the `ManagedPolicyArns` attribute, which gives access rights to call the `dynamodb:PutItem` action.

The following is our modified CloudFormation template for Lambda and DynamoDB in JSON (`lambda-dynamodb.json`):

```
{
  .....
        },
        "myDynamoDBTable" : {
            "Type" : "AWS::DynamoDB::Table",
            "Properties" : {
                "TableName": "mydynamodb",
                "AttributeDefinitions": [
                  {"AttributeName" : "id", "AttributeType" : "S"}
                ],
                "KeySchema": [
                  { "AttributeName": "id", "KeyType": "HASH" }
                ],
                "ProvisionedThroughput" : {
                   "ReadCapacityUnits" : "5",
                   "WriteCapacityUnits" : "5"
                }
            }
        },
        "TestLambdaExecutionRole": {
...
                            "Resource": [
                                {"Fn::Join" : ["", ["arn:aws:dynamodb:",
{"Ref": "AWS::Region"}, ":", {"Ref": "AWS::AccountId"},
":table/mydynamodb"]]}
                            ]
                        }]
                    }
                }],
...
        }
    }
  }
```

The following is the CloudFormation template in YAML (`lambda-dynamodb.yaml`):

```
...
Resources:
  ...
      Code:
        ZipFile:
          Fn::Join:
          - "\n"
          -
      Timeout: '10'
      Runtime: nodejs6.10
  myDynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
 ...
      - AttributeName: id
        KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: '5'
        WriteCapacityUnits: '5'
  TestLambdaExecutionRole:
...
      ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      Policies:
      - PolicyName: dynamodb
        PolicyDocument:
          Version: '2012-10-17'
...
```

Save all CloudFormation template files, `lambda-dynamodb.json`, and/or `lambda-dynamodb.yaml`.

Next, we'll deploy our the template file to AWS CloudFormation.

Deploying the CloudFormation template

After we prepare our CloudFormation template file, we can continue to deploy the template in AWS CloudFormation. In this section, we'll use CloudFormation Management Console.

We use the same method to upload the CloudFormation template file to AWS CloudFormation. Perform the following steps:

1. Open your browser and navigate to CloudFormation Management Console at `https://console.aws.amazon.com/cloudformation`.
2. Upload the CloudFormation template file that we have created to access the Lambda and DynamoDB resources.
3. You can upload `lambda-dynamodb.json` or `lambda-dynamodb.yaml`.
4. Once done, you will be asked to fill in the stack name and the Lambda function name:

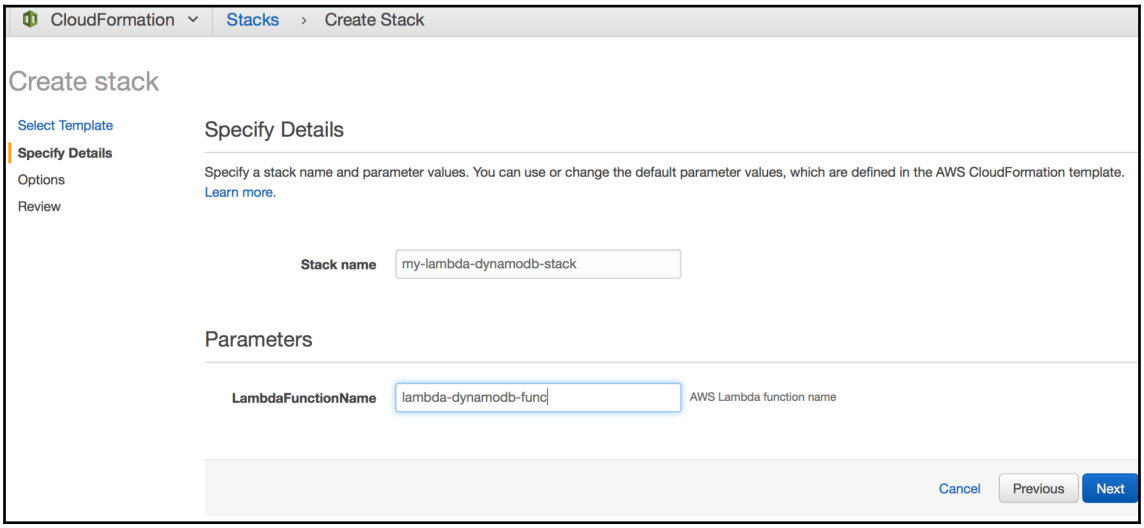


Figure 5.25: Creating a new CloudFormation stack

5. Click on the **Next** button and follow the instructions until the end of the upload process.
6. Once the process is complete, check your CloudFormation status on the CloudFormation dashboard:

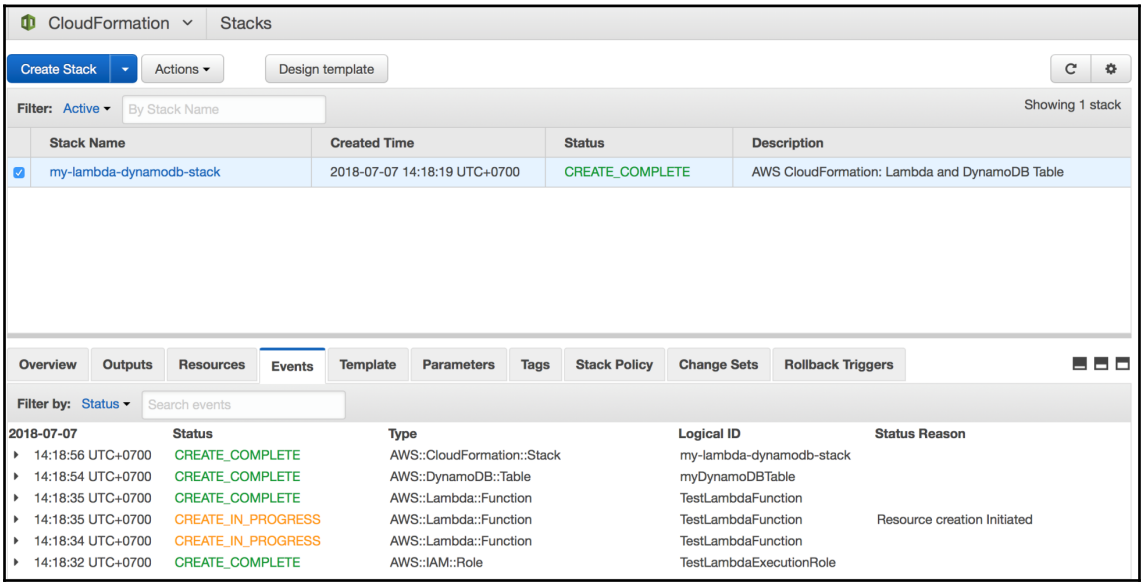


Figure 5.26: The CloudFormation stack was created

7. Make sure your CloudFormation stack has the **CREATE_COMPLETE** status in order to continue to the next step.

8. You can verify the DynamoDB table with the name `mydynamodb` on DynamoDB Management Console at `https://console.aws.amazon.com/dynamodb/`. You should see your DynamoDB table, `mydynamodb`, as shown in the following screenshot:

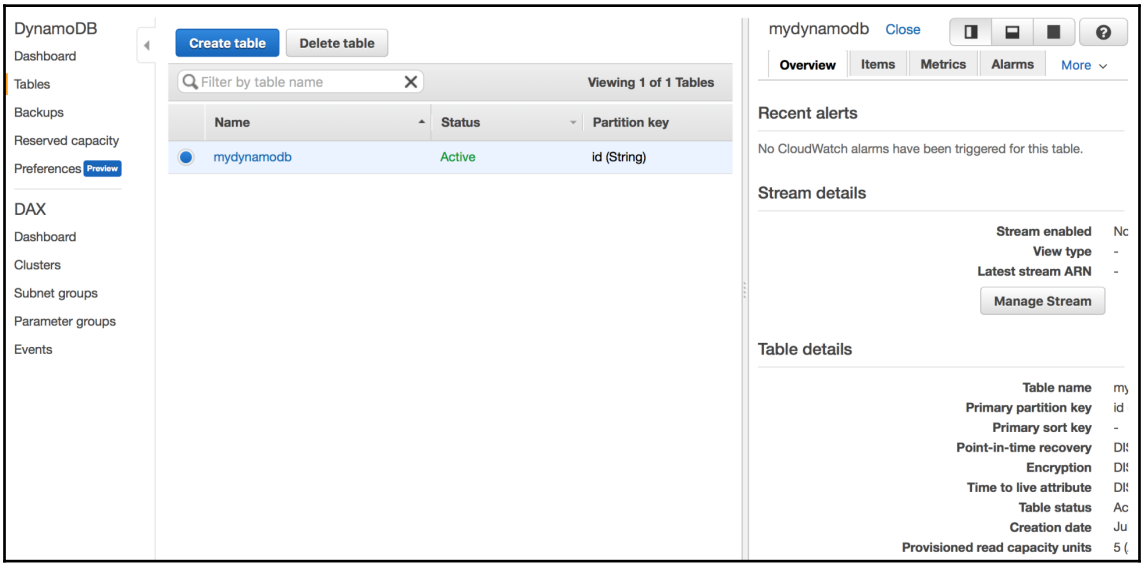


Figure 5.27: The DynamoDB table, mydynamodb, was created from CloudFormation

9. Verify your Lambda function on the Lambda Management Console at `https://console.aws.amazon.com/lambda/`. You should see your Lambda function, **lambda-dynamodb-func**, in the following screenshot:

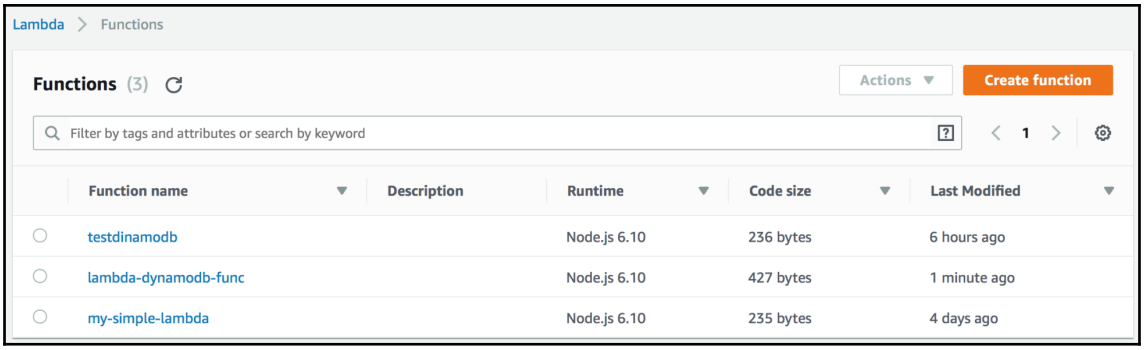


Figure 5.28: The Lambda function, lambda-dynamodb-func was created from CloudFormation

Now, you have finished deploying your Lambda function with the DynamoDB resource through CloudFormation.

Next, we'll configure the user policy in order to invoke the Lambda function.

Configuring the Lambda invocation policy

We'll continue to invoke our Lambda function through the AWS CLI. To enable invoking the Lambda function, your AWS CLI user should have the access policy to access Lambda and DynamoDB.

In this section, we add DynamoDB permissions to access this resource. Follow these steps:

1. Open your browser and navigate to IAM Management Console at `https://console.aws.amazon.com/iam/`.
2. Open your account, from the **Users** menu, and add permissions.
3. Add **AmazonDynamoDBFullAccess** permission:

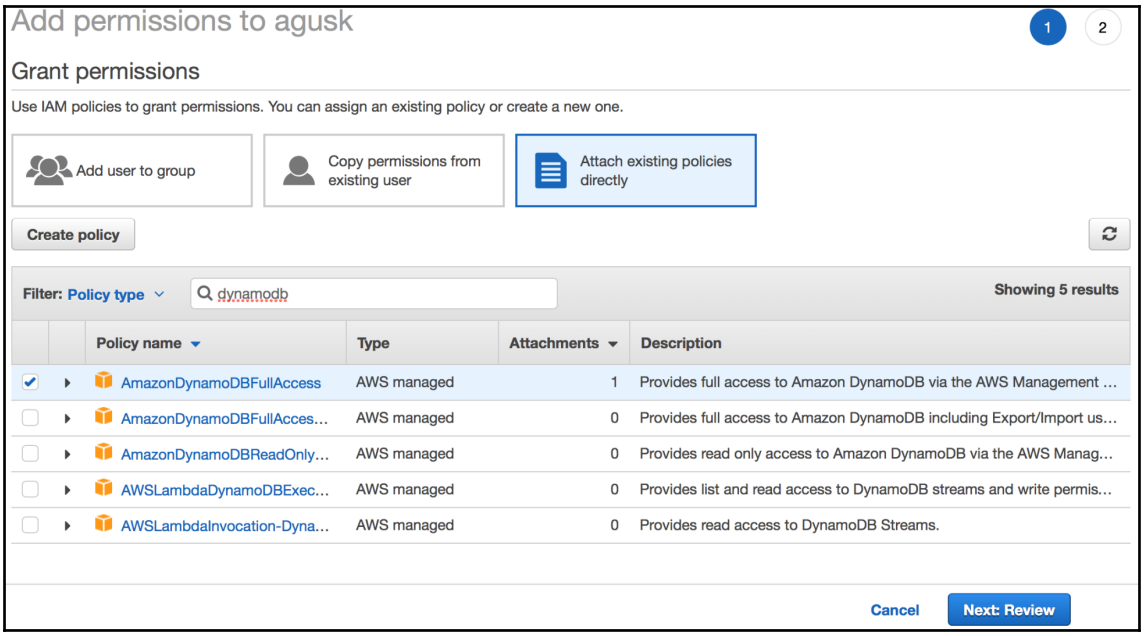


Figure 5.29: Adding permission to the IAM user

4. Once done, click on the **Next: Review** button. You should get the following review screen:

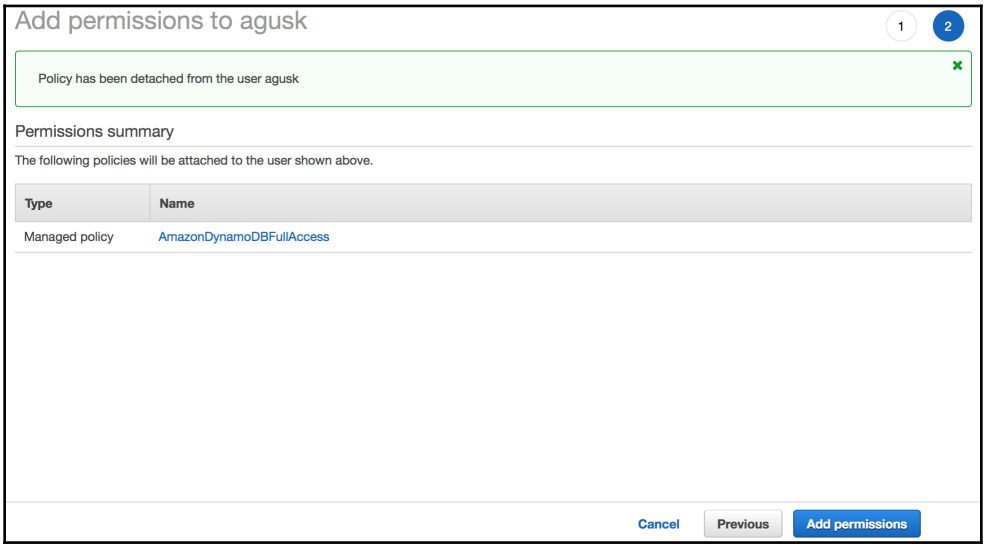


Figure 5.30: Adding AmazonDynamoDBFullAccess

5. If you think it's finished, click on the **Add permissions** button. Your account should have DynamoDB permission:

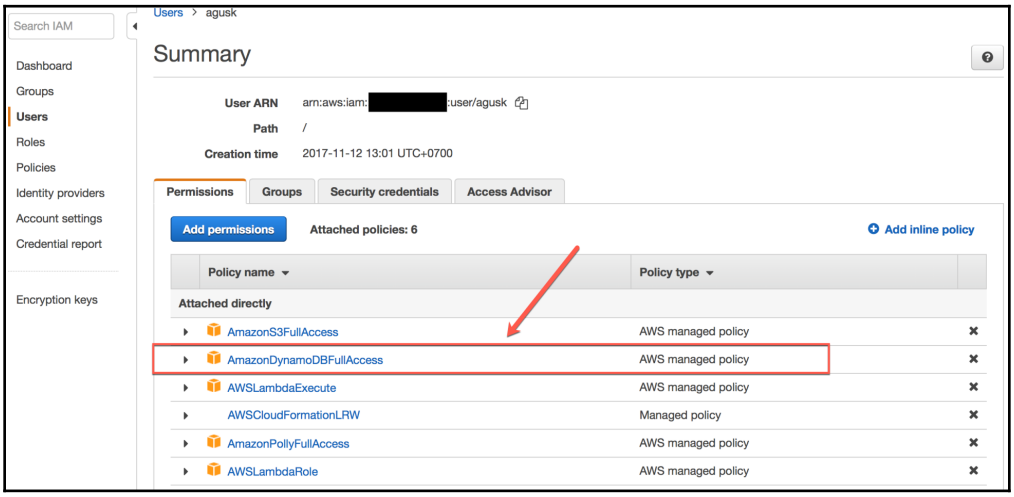


Figure 5.31: The AmazonDynamoDBFullAccess permission was added to an IAM user

You have finished adding permissions to invoke the Lambda function from the AWS CLI. Next, we'll test invoking the Lambda function.

Testing our Lambda function

To test our Lambda function using the AWS CLI, open the Terminal. We send JSON data as follows:

```
{"email": "user1@email.com", "name":"hessa", "country":"DE", "age":"32"}
```

We pass this data while invoking the AWS Lambda function. The Lambda function is `lambda-dynamodb-func`. We also set the output file with `lambda-dynamodb.txt`.

You can type this command to invoke our Lambda function:

```
$ aws lambda invoke --invocation-type RequestResponse --function-name
lambda-dynamodb-func --payload '{"email": "user1@email.com",
"name":"hessa", "country":"DE", "age":"32"}' lambda-dynamodb.txt
```

If successful, you should get the following `200` status code:

```
agusk$ aws lambda invoke --invocation-type RequestResponse --function-name lambd
a-dynamodb-func --payload '{"email": "user1@email.com", "name":"hessa", "country
":"DE", "age":"32"}' lambda-dynamodb.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
agusk$
```

Figure 5.32: Invoking the Lambda function

You also get an output file, `lambda-dynamodb.txt`. You can open it using the `nano` command:

```
$ nano lambda-dynamodb.txt
```

Then, you should see a response message from our Lambda function. For instance, here is my output file:

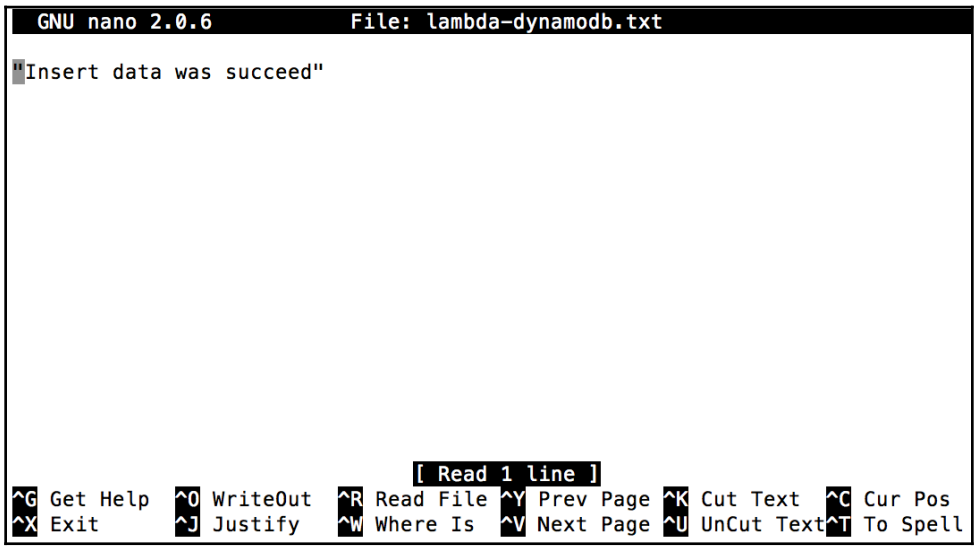


Figure 5.33: Displaying the output file, lambda-dynamodb.txt

You can verify the data was inserted into the DynamoDB table. You can open DynamoDB Management Console at `https://console.aws.amazon.com/dynamodb/`. You should see the data. You can see my data output in the following screenshot:

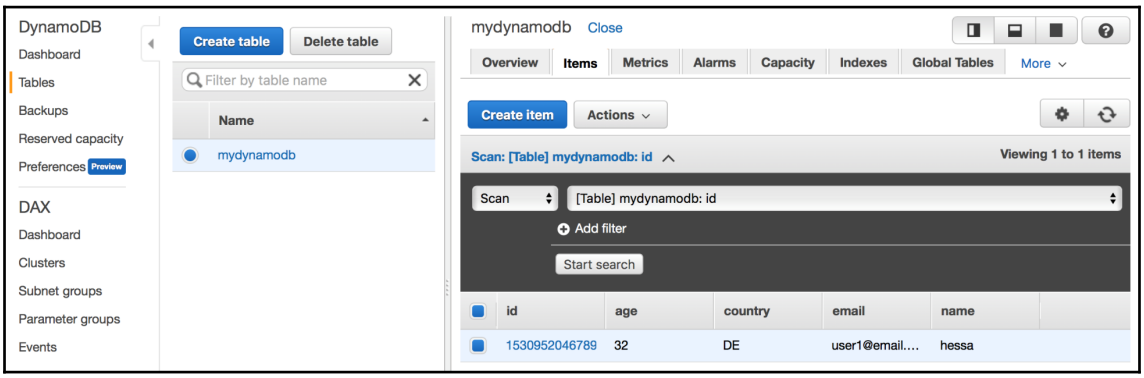


Figure 5.34: Displaying data on the DynamoDB Management Console

You have finished building the Lambda function with access to DynamoDB through CloudFormation.

Next, we'll deploy the Lambda function on various regions through CloudFormation.

Deploying the Lambda function to multiple regions

We have learned how to deploy Lambda functions through CloudFormation. We also accessed DynamoDB resources inside the Lambda function. In this section, we'll deploy a Lambda function to various regions with CloudFormation.

You can follow the next steps to implement our demo and deploy the Lambda function to multiple regions.

Preparation

To work with certain regions on CloudFormation, we can use a CloudFormation StackSet. You learned about this in `Chapter 4`, *AWS CloudFormation StackSets*. In this demo, we'll build a Lambda function to be deployed to certain regions.

To follow this demo, you should have already configured security policies for CloudFormation StackSet. You did that in `Chapter 4`, *AWS CloudFormation StackSets*. You should deploy these template files to CloudFormation Stack in the `AWSCloudFormationStackSetAdministrationRole.yml` and `AWSCloudFormationStackSetExecutionRole.yml` files.

Please read `Chapter 4`, *AWS CloudFormation StackSets* to learn how to deploy them to CloudFormation template files. In the following screenshot, you can see that the `AWSCloudFormationStackSetAdministrationRole.yml` and `AWSCloudFormationStackSetExecutionRole.yml` CloudFormation template files were deployed:

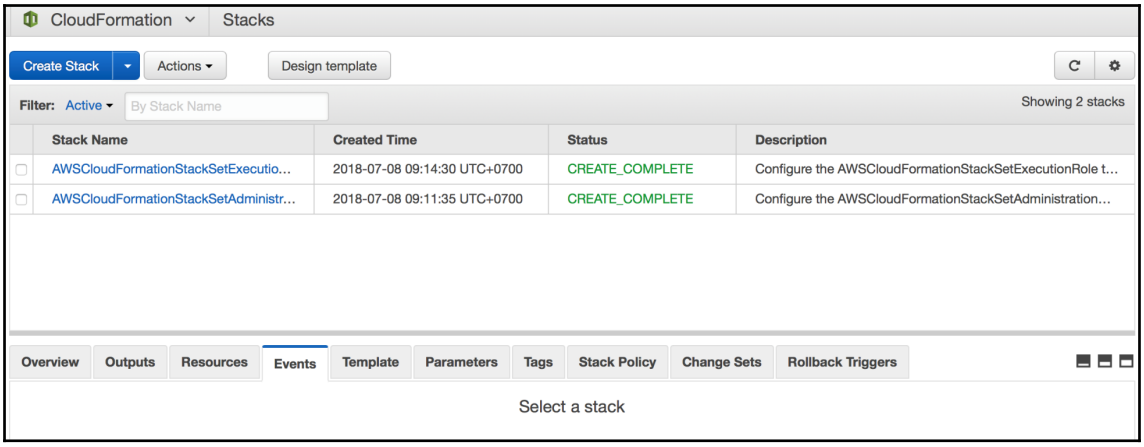


Figure 5.35: Applying security permissions for CloudFormation StackSet

Next, we'll develop a CloudFormation template for the Lambda function that targets multiple regions.

Developing a CloudFormation template for the Lambda function

In this demo, we'll modify our CloudFormation template file `Lambda-CloudFormation.json/Lambda-CloudFormation.yaml`.

The following is our Lambda function code:

```
var region = process.env.AWS_REGION;
exports.handler = (event, context, callback) => {
     var data = event['msg'];
     callback(null, 'Received: ' + data + ' . Region: ' + region);
}
```

This function returns a message with the region information included. It's done by calling the `process.env.AWS_REGION` API. Save the CloudFormation template file as `Lambda-multi-regions.json` or `Lambda-multi-regions.yaml`.

Next, we'll deploy this CloudFormation template to a CloudFormation StackSet.

Deploying the Lambda function to multiple regions

Once we've prepared a CloudFormation StackSet and created the CloudFormation template file, we can continue to deploy the template to StackSet.

In this demo, we use the CloudFormation StackSet Management Console. Follow these steps:

1. Open your browser and navigate to the CloudFormation StackSet Management Console at `https://console.aws.amazon.com/cloudformation/stacksets/`.
2. Click on the **Create StackSet** button and you should get the following screen:

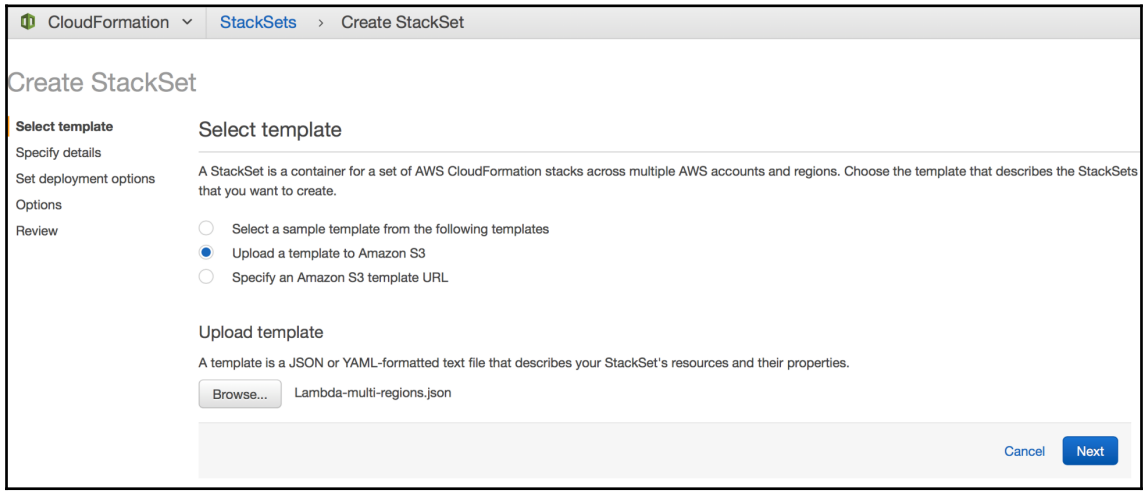


Figure 5.36: Selecting a template for CloudFormation StackSet

3. Select the **Upload a template to Amazon S3** option and upload `Lambda-multi-regions.json` or `Lambda-multi-regions.yaml` by clicking on the **Browse** button.

4. Click on the **Next** button to get the following screen:

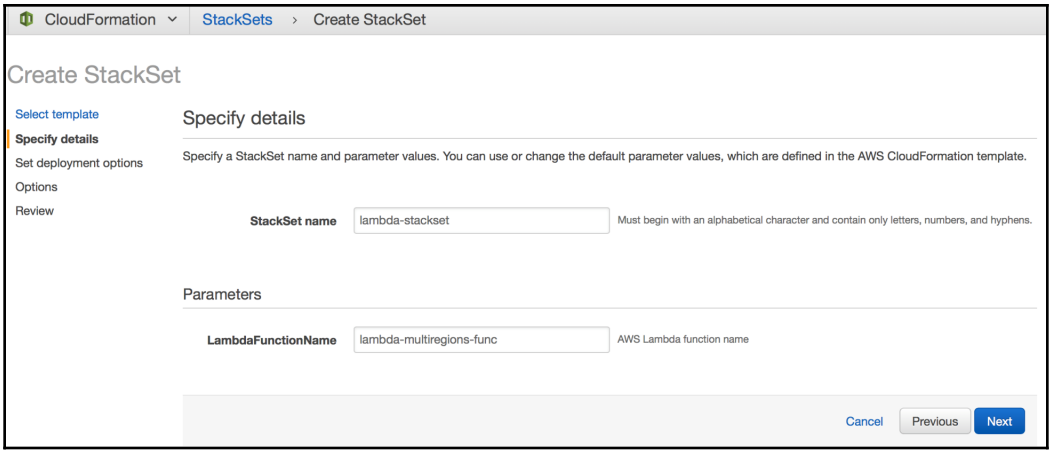


Figure 5.37: Filling in the StackSet and Lambda function names

5. Fill in the StackSet and Lambda function names and click on the **Next** button. You will get the following screenshot:

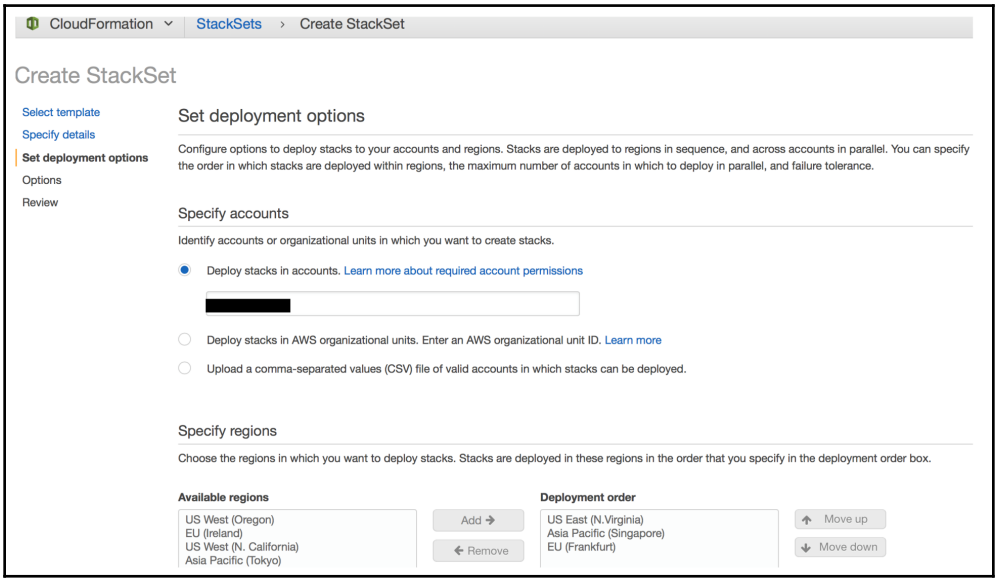


Figure 5.38: Setting the account and target regions

6. Fill in the ARN code from your account in the **Deploy stacks in accounts** option. You will also be asked to fill in regions for deployment targets. In this demo, I selected three regions—**US East (N.Virginia)**, **Asia Pacific (Singapore)**, and **EU (Frankfurt)**.
7. Click on the **Next** button until you get the following screen:

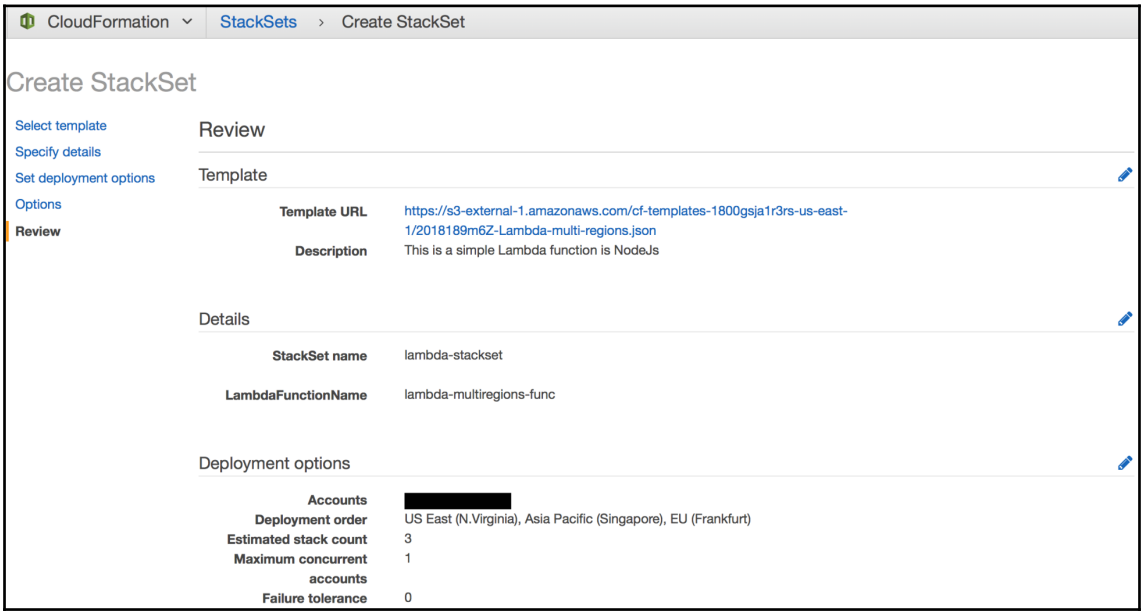


Figure 5.39: Reviewing the CloudFormation StackSet

8. Review all input and, after completing the review, click on the **Create** button.
9. CloudFormation provisions your template for the targeted regions. If successful, you should see **SUCCEEDED** in your StackSet:

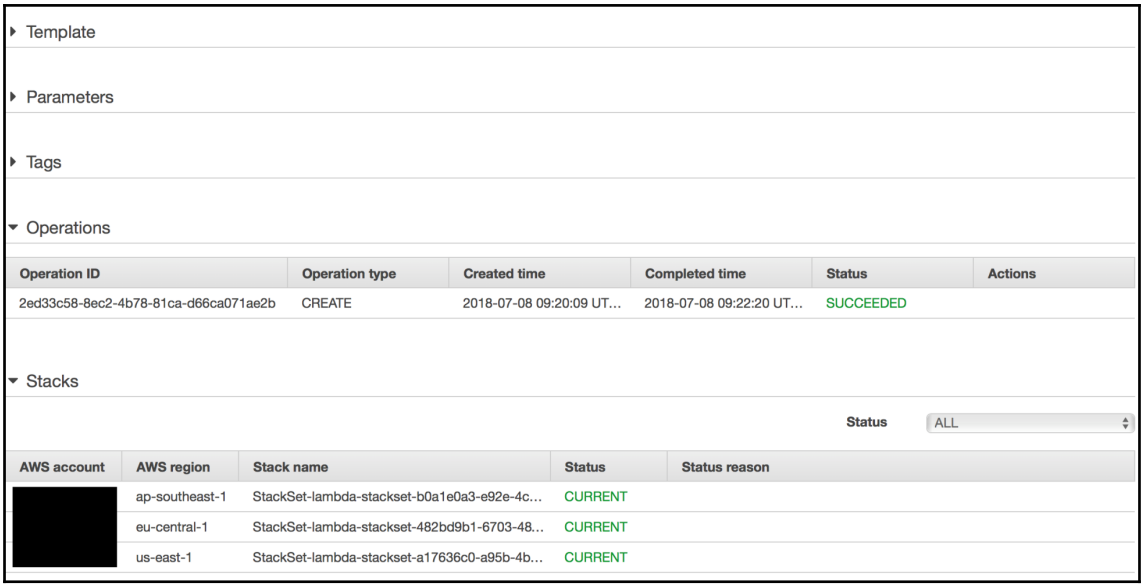


Figure 5.40: CloudFormation StackSet was created with three regions

Now you have finished deploying an AWS CloudFormation StackSet with the Lambda function. Next, we will invoke this function.

Invoking the Lambda function

Now, you can invoke your Lambda function. In this demo, I use the AWS CLI. We send this data as follows:

```
{"msg": "this is AWS CLI"}
```

The Lambda function is `lambda-multiregions-func`. You can invoke this Lambda function from the AWS CLI as follows:

```
$ aws lambda invoke --invocation-type RequestResponse --function-name
lambda-multiregions-func --payload '{"msg": "this is AWS CLI"}' output-
multiregion.txt
```

If successful, you'll get a status code of `200`, as shown in the following screenshot:

```
agusk$ aws lambda invoke --invocation-type RequestResponse --function-name lambd
a-multiregions-func --payload '{"msg": "this is AWS CLI"}' output-multiregion.tx
t
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
agusk$
```

Figure 5.41: Invoking the Lambda function with the US East region

Open your output file to get a response from the AWS Lambda function:

```
$ nano output-multiregion.txt
```

The sample output is shown as follows:

```
  GNU nano 2.0.6          File: output-multiregion.txt

"Received: this is AWS CLI . Region: us-east-1"

                          [ Read 1 line ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 5.42: Opening the output file

You have finished deploying the Lambda function in certain regions through CloudFormation.

Summary

We learned how to deploy Lambda functions through CloudFormation. Then, we included DynamoDB resources inside Lambda function codes. Finally, we deployed a Lambda function to various regions with CloudFormation.

In the next chapter, we'll learn how to work with an AWS IoT deployment through CloudFormation.

Questions

Use these questions to test your understanding of this chapter:

1. List the steps to deploy the AWS Lambda function with CloudFormation.
2. How do you develop a CloudFormation template for the Lambda function?
3. How do you deploy the Lambda function tocertain regions using CloudFormation?

6
AWS CloudFormation Security

Designing and deploying infrastructure need some care when it comes to security issues. In this chapter, we explore how to build IaC with security compliances. Some security best practices and recommendations are explored in order to apply secure AWS CloudFormation.

The following is a list of topics that we will explore:

- Security threads and models for AWS CloudFormation
- Best practices for AWS security
- Managing all AWS resource securities
- Reducing security accesses to CloudFormation Stacks
- Stack Policies
- IAM conditions for CloudFormation
- AWS security checklist

Security threats and models for AWS CloudFormation

Amazon AWS consists of AWS services. The more you use AWS services on your system, the more security risks your system will have. Each AWS service needs special attention to address security problems.

Understanding security threats and models from our system can take advantages to address security issues. Amazon AWS provides AWS security resources to help us to harden our system. Refer to `https://aws.amazon.com/security/security-resources/`.

AWS security operates on a Shared Security Responsibility model. This means that Amazon secures its infrastructure while you have your own security controls in place for the data and applications you deploy and store in the cloud. You can find details of the AWS Security Responsibility model on this link, `https://aws.amazon.com/compliance/shared-responsibility-model/`.

AWS protects the infrastructure running all the services offered in the AWS Cloud. This infrastructure is composed of the hardware, software, networking, and facilities that run AWS Cloud services. Otherwise, customers take responsibility for meeting AWS configuration security requirements on each AWS resource that deploys into their system. We describe this model in *Figure 6-1*. As customers, we should give more attention to our system implementation such as data and configurations/settings.

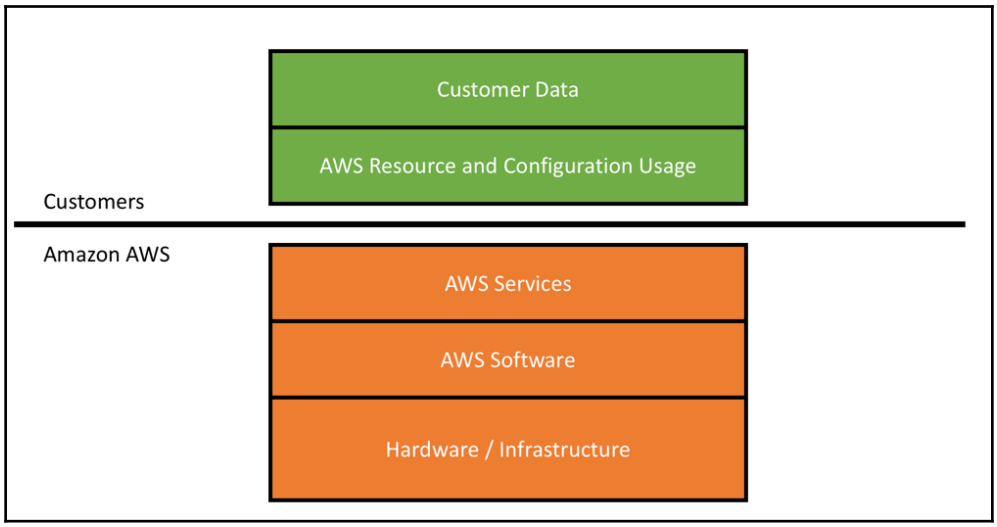


Figure 6-1: A simple of AWS Security Responsibility model

We also review some security threat modeling in order to understand your system risks. A simple security threat modeling example is STRIDE, created by Microsoft. Technically, this model is applied to computer systems but we can apply this model for our AWS security threat model. A brief STRIDE model is described in the following table. Further information about STRIDE is available from `https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)`.

Security Threat	Mitigation	Mitigation Samples
Spoofing	Authentication	Passwords Multi-factor authentication Digital signatures
Tampering	Integrity	Permission/ACLs Digital signatures
Repudiation	Non-Repudiation	Secure logging and auditing Digital signatures
Information disclosure	Confidentiality	Encryption Permissions/ACLs
Denial of service (DoS)	Availability	Permission/ACLs Filtering Quotas
Elevation of privilege	Authorization	Permissions/ACLs Input validation

Amazon AWS also provides security services to help you to investigate your system implementation. They can analyze your system to identify security threats and risks. They can perform penetration testing on your AWS platform. If you are interested, you should request this service on this site, `https://aws.amazon.com/security/penetration-testing/`.

The following is a list of AWS resources on which AWS can help you to perform penetration testing.

- EC2
- RDS
- Aurora
- CloudFront
- API Gateway
- Lambda
- Lightsail
- DNS Zone Walking

Some third-party local and global companies also can help you to investigate your security threats. For instance, ThreatModeler provides tools to find your security threats and then build its AWS threat model. You can read a summary report from ThreatModeler about the AWS threat model for web application on this link, `https://threatmodeler.com/wp-content/uploads/2018/04/AWS-Basic-Web-App-Hosting-Summary-Report.pdf`.

Best practices for AWS security

The easier method to reduce security risks on our AWS system is to follow best practices. In general, best practices consist of security recommendations from security experts based on their experience in addressing security.

Amazon AWS provides AWS security best practices to help their customers harden security problems while deploying their systems on the AWS platform. You can read it on `https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf`.

Managing all AWS resource securities

AWS Amazon provides a security central to manage security accesse to all AWS resources. We can review our users, roles, and their permissions while accessing AWS resources using AWS IAM. Check this out on `https://console.aws.amazon.com/iam/`.

Figure 6-2 show one IAM user. We can configure its permissions and policies to ensure the user is safe. If you think this user is not used, you should remove it from AWS IAM.

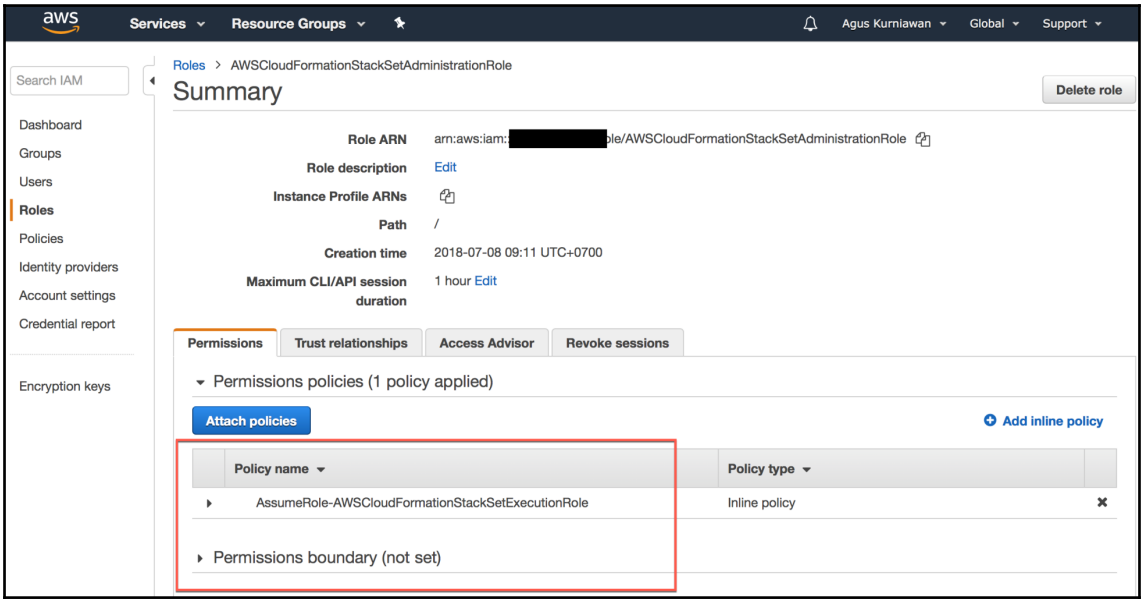


Figure 6-2: Managing role permissions on AWS IAM

You also should pay attention to your IAM roles. You should review all role permissions. Remove permissions if IAM roles do not use them. *Figure 6-3* shows an IAM role with its permissions:

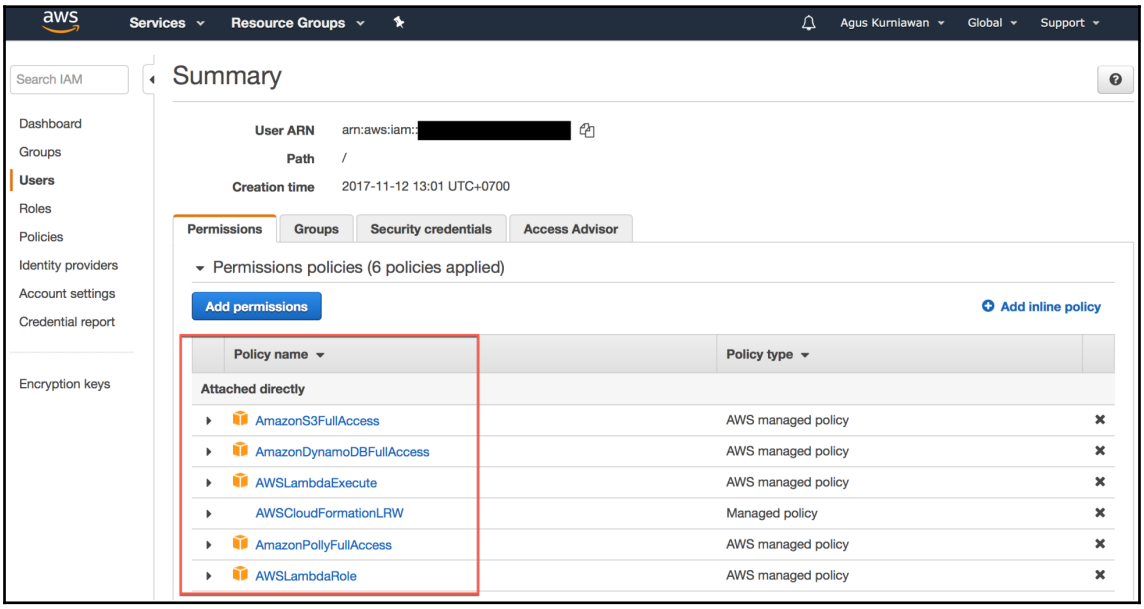


Figure 6-3. Managing user permissions on AWS IAM

Reducing security access to CloudFormation stacks

All AWS resources can be managed through AWS IAM, including policies and users or roles. AWS CloudFormation uses IAM to control its security for template deployment. We can use an IAM policy on our CloudFormation template. A CloudFormation policy can be defined as follows.

```
{
  "Statement" : [
    {
      "Effect" : "Deny_or_Allow",
      "Action" : "update_actions",
      "Principal" : "*",
      "Resource" : "LogicalResourceId/resource_logical_ID",
      "Condition" : {
```

```
          "StringEquals_or_StringLike" : {
            "ResourceType" : [resource_type, ...]
          }
        }
      }
    ]
}
```

It's recommended you limit security access on your resources on CloudFormation. This method applies the principle of least privilege. For instance, we remove updating and deleting access on CloudFormation stacks from IAM users or roles. A template sample is as follows.

```
{
    "Version":"2012-10-17",
    "Statement":[{
        "Effect":"Allow",
        "Action":[
            "cloudformation:*"
        ],
        "Resource":"*"
    },
    {
        "Effect":"Deny",
        "Action":[
            "cloudformation:UpdateStack",
            "cloudformation:DeleteStack"
        ],
        "Resource":"arn:aws:cloudformation:us-
east-1:123456789012:stack/MyStack/*"
    }]
}
```

This CloudFormation template allows access to all CloudFormation APIs, but denies `UpdateStack` and `DeleteStack` APIs access on your MyStack stack.

You should investigate all resource usages and analyze what kind of security access to be applied on the template. With regard to CloudFormation actions, we can use the following actions and apply the least privilege principle.

- CancelUpdateStack
- ContinueUpdateRollback
- CreateStack
- DeleteStack
- DescribeStackEvents

- DescribeStackResource
- DescribeStackResources
- DescribeStacks
- EstimateTemplateCost
- GetStackPolicy
- GetTemplate
- GetTemplateSummary
- ListExports
- ListImports
- ListStackResources
- ListStacks
- SetStackPolicy
- UpdateStack
- UpdateTerminationProtection
- ValidateTemplate

You can review each action on this site, `https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/Welcome.html`.

Stack policies

In some cases the you likely get framework down because of accidental changes to assets. Your group perform to change CloudFormation which makes your framework shaky. CloudFormation gives stack approaches which keep stack assets from unexpectedly being refreshed or erased amid stack refreshes. We can incorporate with IAM and stack arrangements to address accidental and malevolent changes to your stack assets.

You should perform to set or refresh the strategy, your IAM clients or parts ought to be able to call the `cloudformation:SetStackPolicy` activity. As a matter of course, setting a stack strategy secures all stack assets with a Deny to deny any updates except if you indicate an express Allow. For example, we shield a specific asset from refreshes activity after the framework go live. You can see the accompanying CloudFormation format.

```
{
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
```

```
        "Principal": "*",
        "Resource" : "<certain_resource>"
      },
      {
        "Effect" : "Allow",
        "Action" : "Update:*",
        "Principal": "*",
        "Resource" : "*"
      }
    ]
}
```

IAM conditions for CloudFormation

We can perform creation or delation of particular AWS resources through CloudFormation if we have IAM policies to do those tasks. Sometimes you do not need a creation task from CloudFormation operations. We can implement IAM conditions to apply this scenario.

In general, CloudFormation provides IAM policies related to IAM conditions. The following is a list of IAM condition policies:

- `cloudformation:TemplateURL`
- `cloudformation:ResourceTypes`
- `cloudformation:StackPolicyURL`

When you apply IAM conditions, you can ensure that API calls for stack actions, for instances creating, updating, on specific template or are limited to specific resources.

`cloudformation:TemplateURL` is one CloudFormation attribute that shows a CloudFormation template file location. It could be the .json, .yaml, and .template file formats. For instance, we apply an IAM condition on the CloudFormation template as follows.

```
{
    "Version":"2012-10-17",
    "Statement":[{
        "Effect": "Deny",
        "Action": [
            "cloudformation:CreateStack",
            “cloudformation:UpdateStack”
        ],
        "Resource": "*",
        "Condition": {
            "StringNotEquals": {
```

```
                "cloudformation:TemplateURL": [
"https://s3.amazonaws.com/cloudformation-templates-us-east-1/IAM_Users_Grou
ps_and_Policies.template"
                ]
            }
        }
    },
    {
        "Effect": "Deny",
        "Action": [
            "cloudformation:CreateStack",
            "cloudformation:UpdateStack"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "cloudformation:TemplateURL": "true"
            }
        }
    }]
}
```

This template ensures that, for all `CreateStack` or `UpdateStack` API calls, users must use the specified template. Otherwise, the operation will be denied.

`Condition:StackPolicyURL` enables your CloudFormation to apply a stack policy with it upon creation with the `StackPolicyURL` condition. The following is a CloudFormation template from AWS to use cloudformation:StackPolicyUrl in the template.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
            "Effect": "Deny",
            "Action": [
                "cloudformation:SetStackPolicy"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringNotEquals": {
                    "cloudformation:StackPolicyUrl": [
"https://s3.amazonaws.com/samplebucket/sampleallowpolicy.json"
                    ]
                }
            }
        },
       {
        "Effect": "Deny",
```

```
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringNotEquals": {
                    "cloudformation:StackPolicyUrl": [
    “https://s3.amazonaws.com/samplebucket/sampledenypolicy.json”
                    ]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                “cloudformation:SetStackPolicy”
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "cloudformation:StackPolicyUrl": "true"
                }
            }
        }]
    }
```

AWS security checklist

After you design a system via CloudFormation and deploy it to the AWS platform, the last task is to ensure your system complies with AWS Security. AWS provides a security checklist that consists of security checking actions. You can learn about the security checklist in the following document, `https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Checklist.pdf`. This document consists of the following three security checklists on various AWS resources:

- General security checklist
- Security checklist for EC2/VPC/EBS
- Security checklist for S3

With using AWS security checklist, your system probably has low security risks. But again, it's better to perform penetration testing regularly.

Summary

In this chapter, we learned how we can go about securing resources that are deployed using AWS CloudFormation. Finally, we delved into understanding the security threats and models for CloudFormation.

Assessment

Chapter 1

1. **Infrastructure as Code** (**IaC**) is one of IT infrastructure types that manages and provision automatically through code, rather than using a manual process.
2. Benefits of IaC are to minimize risks on building and deploying infrastructure and to manage infrastructure automatically.
3. The main objective of AWS CloudFormation is to optimize your IaC design with AWS resources and to provide a complete solution to build modern infrastructure based on Amazon AWS technology stacks.
4. AWS CloudFormation is designed to simplify IaC development. Starting from designing IaC using file (JSON or YAML) or designer to generate CloudFormation template file. This template file uploads to AWS CloudFormation server. Then, CloudFormation will generate all resources that we already defined on the template file. AWS will deploy them on certain container. This process runs automatically.
5. We can build CloudFormation template from writing scripts on file and using CloudFormation designer. We can develop CloudFormation template files in JSON and YAML.
6. In some cases, we want to deploy a certain resource in an infrastructure environment. This resource consists of various different resource that also are used for other resources. This scenario can be done by implementing nested stacks on CloudFormation. Resource modularity is a key to manage your infrastructure easily.
7. The idea of implementing CloudFormation StackSets is to deploy various a single of resource unit in different region. If we want to deploy the same resources on different region, we can CloudFormation StackSets. You also can configure certain settings on different region.

Chapter 2

1. CloudFormation stack is an instance of CloudFormation that consists of a collection of AWS resources to build infrastructure based on Amazon AWS technology stack.
2. The main benefit to use web management console to build CloudFormation is easier to use since we perform it graphically. We don't need to remember all commands from AWS CLI
3. Building CloudFormation through AWS CLI can take benefits for simplicity. Since AWS CLI works on Terminal, you don't need more bandwidth usages if we compare to use management console.

Chapter 3

1. A CloudFormation template is a script file that probably is written in JSON or YAML format to build an infrastructure based on Amazon AWS technology.
2. To develop a CloudFormation template, you should have knowledge to write JSON or YAML format. A CloudFormation template has a skeleton JSON format as follows:

```
{
 "AWSTemplateFormatVersion" : "version date",
 "Description" : "JSON string",
 "Metadata" : {
   template metadata
 },
 "Parameters" : {
   set of parameters
 },
 "Mappings" : {
   set of mappings
 },
 "Conditions" : {
   set of conditions
 },
 "Transform" : {
   set of transforms
 },
 "Resources" : {
   set of resources
 },
 "Outputs" : {
   set of outputs
```

```
 }
}
```

You also can write it in YAML:

```
AWSTemplateFormatVersion: "version date"
Description:
 String
Metadata:
 template metadata
Parameters:
 set of parameters
Mappings:
 set of mappings
Conditions:
 set of conditions
Transform:
 set of transforms
Resources:
 set of resources
Outputs:
 set of outputs
```

3. To implement CloudFormation template, you can write CloudFormation in JSON or YAML manually on file. You also can build the template via CloudFormation designer tool from AWS. Then, you should learn about AWS resource types. You can read it on `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-product-property-reference.html`. Last, you can do more practices. You can use template samples from AWS on `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html`.

Chapter 4

1. A CloudFormation StackSet is a collection of Stacks that are deployed cross accounts and regions. You can perform provisions various regions with some regions.
2. Based on AWS documentation, we can create a maximum of 20 stack sets in our administrator account, and a maximum of 500 stack instances per stack set. Reference at `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-limitations.html`

Chapter 5

1. In general, we can deploy AWS Lambda function using Management Console and AWS CLI with the following steps:

    - Create CloudFormation Template based on this template, `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html`
    - Deploy the template using Web Management Console or AWS CLI
    - Configure IAM to users that will invoke Lambda functions

2. To develop CloudFormation template for Lambda function, you should follow the template instructions on `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html`.

   The following is the template for JSON:

```
{
  "Type" : "AWS::Lambda::Function",
  "Properties" : {
    "Code" : Code,
    "DeadLetterConfig" : DeadLetterConfig,
    "Description" : String,
    "Environment" : Environment,
    "FunctionName" : String,
    "Handler" : String,
    "KmsKeyArn" : String,
    "MemorySize" : Integer,
    "ReservedConcurrentExecutions" : Integer,
    "Role" : String,
    "Runtime" : String,
    "Timeout" : Integer,
    "TracingConfig" : TracingConfig,
    "VpcConfig" : VPCConfig,
    "Tags" : [ Resource Tag, ... ]
  }
}
```

   The following is the template for YAML:

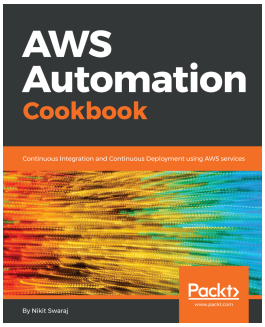
```
Type: "AWS::Lambda::Function"
Properties:
  Code:
    Code
  DeadLetterConfig:
    DeadLetterConfig
```

```
Description: String
Environment:
  Environment
FunctionName: String
Handler: String
KmsKeyArn: String
MemorySize: Integer
ReservedConcurrentExecutions: Integer
Role: String
Runtime: String
Timeout: Integer
TracingConfig:
  TracingConfig
VpcConfig:
  VPCConfig
Tags:
  Resource Tag
```

3. Firstly, we configure permissions on CloudFormation. You should deploy stacks: `AWSCloudFormationStackSetAdministrationRole.yml` and `AWSCloudFormationStackSetExecutionRole.yml` files. Read `Chapter 4`, *AWS CloudFormation StackSets* for further steps. Then, you can deploy StackSet as usual when you deploy a StackSet.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



AWS Automation Cookbook
Nikit Swaraj

ISBN: 978-1-78839-492-8

- Build a sample Maven and NodeJS Application using CodeBuild
- Deploy the application in EC2/Auto Scaling and see how CodePipeline helps you integrate AWS services
- Build a highly scalable and fault tolerant CI/CD pipeline
- Achieve the CI/CD of a microservice architecture application in AWS ECS using CodePipeline, CodeBuild, ECR, and CloudFormation
- Automate the provisioning of your infrastructure using CloudFormation and Ansible
- Automate daily tasks and audit compliance using AWS Lambda
- Deploy microservices applications on Kubernetes using Jenkins Pipeline 2.0

Effective DevOps with AWS
Nathaniel Felsen

ISBN: 978-1-78646-681-5

- Find out what it means to practice DevOps and what its principles are
- Build repeatable infrastructures using templates and configuration management
- Deploy multiple times a day by implementing continuous integration and continuous deployment pipelines
- Use the latest technologies, including containers and serverless computing, to scale your infrastructure
- Collect metrics and logs and implement an alerting strategy
- Make your system robust and secure

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

A

C

D

E

I

J

L

M

P

S

W

Y