Automating Infrastructure Deployment

Step 1: Creating an AWS CloudFormation template from scratch

1. Navigate to the AWS Cloud9 service and open the integrated development environment (IDE) of the existing AWS Cloud9 instance.
2. In the AWS Cloud9 IDE, choose File > New File, then choose File > Save, and save the new file as: S3.yaml
3. At the top of the file, add the following two lines:

```
AWSTemplateFormatVersion: "2010-09-09"
Description:
```

4. Next, add the following three lines to your template:

```
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
```

5. Add a description (such as "cafe S3 template") on the Description: line. Before you start your description, be sure that you have a space *after* the colon (:). After you enter the description, Save the changes to file.
6. In the Bash terminal, run these two lines of code:

```
aws configure get region
aws cloudformation create-stack --stack-name CreateBucket --template-body file://S3.yaml
```

7. The first line of code that you ran returned the default AWS Region of the AWS CLI client that is installed on the AWS Cloud9 instance. Leave the default Region.The second line of code that you ran created a stack that used the template you defined. Because you did not specify the Region in the command, the stack will be created in the default Region. If the create-stack command ran successfully, you should see some output that is formatted in JavaScript Object Notation (JSON). This output should indicate a *StackId*.
8. In the AWS Management Console, navigate to the AWS CloudFormation service and observe the details of the *CreateBucket* stack.For example, look at the information in the Events, Resources, Outputs, and Template tabs.
9. Navigate to the Amazon S3 service page to observe the bucket that your template created.

Step 2: Configuring the bucket as a website and updating the stack

10. Set bucket ownership controls and public access and then upload the static website assets to the bucket.

To do this task, run the following commands in the Bash terminal (replace all three occurences of *<BUCKET-NAME>* with your actual bucket name):

```
#1. Download the website files
wget
https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACACAD-2-91555/14-lab-mod10-challenge-CFn/s3/static-website.zip
unzip static-website.zip -d static
cd static
#2. Set the ownership controls on the bucket
aws s3api put-bucket-ownership-controls --bucket <BUCKET-NAME> --ownership-controls Rules=[{ObjectOwnership=BucketOwnerPreferred}]
#3. Set the public access block settings on the bucket
aws s3api put-public-access-block --bucket <BUCKET-NAME>
--public-access-block-configuration
"BlockPublicAcls=false,RestrictPublicBuckets=false,IgnorePublicAcls=false,BlockPublicPolicy=false"
#4. Copy the website files to the bucket
aws s3 cp --recursive . s3://<BUCKET-NAME>/ --acl public-read
```

11. If these operations are successful, you should see numerous *upload:<file_name>* messages in the command output.
12. In a new browser tab, open the AWS CloudFormation template documentation for defining S3 bucket resources.
    - Go to the AWS resource and property types reference documentation
    - Scroll down, choose Amazon S3, and then choose the AWS::S3::Bucket resource type.
13. Using the documentation as a reference, modify your S3.yaml template to set the following characteristics on the S3 bucket resource:
    - Attach a *deletion policy* that will retain the bucket
    - Configure the bucket to host a static website with *index.html* set as the index document
14. To your AWS CloudFormation template, add an *output* that provides the website URL. Again, consult the Examples section of the documentation as a reference.
15. Save the changes to your S3.yaml file.
16. Validate your template.Back in the Bash terminal, change the directory back to the location of the S3.yaml file and validate your template by running the following commands.

```
cd ../
aws cloudformation validate-template --template-body file://S3.yaml
```

17. If the output indicates that your template has syntax or other errors, correct them, and then run the command again to verify that they have been resolved.
18. Update the stack by running this command:

aws cloudformation update-stack --stack-name CreateBucket --template-body file://S3.yaml

19. Browse to the AWS CloudFormation service and confirm that your stack update completed successfully.
    - The stack should soon show status *UPDATE_COMPLETE*.
20. Verify success.
    - Does the stack's Outputs tab list an output with a URL value? If so, choose the link.
    - The static website should open

New business requirement: Storing templates in a version control system

Step 3: Cloning a CodeCommit repository that contains AWS CloudFormation templates

21. Browse to the CodeCommit service and in your account, notice the repository that is named *CFTemplatesRepo*.
22. Choose CFTemplatesRepo and then choose the templates folder.
23. Open the *CFTemplatesRepo/templates/start-lab.yaml* file and analyze the contents.
    - Notice that this template defines a few of the resources that you observed in this AWS account.
24. In the breadcrumbs at the top of the page, choose Repositories and in the Clone URL column, choose HTTPS. This action copies the CodeCommit repository's HTTPS clone URL to your clipboard.
25. Return to the AWS Cloud9 IDE and clone the existing CodeCommit repository to your workspace (replace *<url>* with the clone URL that you copied)In the Bash terminal in the AWS Cloud9 IDE, enter this command:

git clone <url>

26. This command clones a copy of the CodeCommit repository that you just observed. The command creates a *CFTemplatesRepo* directory that should now appear in the navigation pane (which is the left pane in the IDE).Use the Git client software to analyze your local copy of the repository.

cd CFTemplatesRepo
git status
The git status command shows what branch of the repository you are connected to. It also shows that your local copy is up to date with the source branch in CodeCommit.

New business requirement: Using a continuous delivery service, create the network and application layers for the café.

Step 4: Creating a new network layer with AWS CloudFormation, CodeCommit, and CodePipeline

27. Create a new AWS CloudFormation template that will create a VPC, public subnet, and other resources.
    - In the navigation pane of the AWS Cloud9 IDE, expand the CFTemplatesRepo/templates directory.
    - In the templates directory, right-click template1.yaml and create a duplicate of it.
    - Rename the duplicate to: cafe-network.yaml
    - In the text editor, open cafe-network.yaml and set the description to: Network layer for the cafe
    - Observe the details of the seven resources that this template creates.
28. Observe the AWS CodePipeline details that were preconfigured in your account.
    - In the AWS Management Console, in the search box to the right of *Services*, search for and choose CodePipeline to open the CodePipeline console.
    - Choose Pipelines.
    - Notice that two pipelines have been predefined for you:
        - CafeAppPipeline
        - CafeNetworkPipeline
29. Analyze the *Source* stage of the CafeNetworkPipeline.
    - Choose CafeNetworkPipeline and observe the pipeline details.In the Source area, you can see that this pipeline's SourceAction is *AWS CodeCommit*.
    - To the right of the SourceAction heading, choose the details in the Configuration window show that the source is the *CFTemplatesRepo* CodeCommit repository.
    - To return to the CafeNetworkPipeline page, choose Done.
30. Analyze the *Deploy* stage of the CafeNetworkPipeline.
    - Notice that the *Deploy* action will be performed by using AWS CloudFormation.
    - To the right of the RunChangeSet heading, choose
31. Return to the AWS Cloud9 instance and trigger the creation of the *update-cafe-network* by checking your AWS CloudFormation template into CodeCommit.
    - Observe how the local copy of the repository differs from the origin. In the Bash terminal, run the following command:

```
git status
```

    - The output should show that the cafe-network.yaml file that you created is currently untracked in Git.
    - Run these two commands to add the new file to the repository and then commit it to the repository with a comment.

```
git add templates/cafe-network.yaml
git commit -m 'initial commit of network template' templates/cafe-network.yaml
```

    - Check the status of your local copy of the repository:

```
git status
```

- The information that is returned should report that your branch is ahead of the origin by one commit.
- Finally, push the commit to the remote repository (this command actually copies the file to CodeCommit):

```
git push
```

32. Return to the CodePipeline console and choose the CafeNetworkPipeline.
    - Observe that the creation of the stack is automatically triggered.
33. In the AWS CloudFormation console, confirm that the *update-cafe-network* stack ran. It should have a status *CREATE_COMPLETE* or *UPDATE_COMPLETE*. Also, check the Outputs tab for the stack. It currently shows no outputs. Soon, however, you will update the stack so that it creates outputs.
34. In the Amazon VPC console, observe that the resources defined in the cafe-network.yaml template were created in the AWS account. For example, the console should list a VPC named *Cafe VPC*, and a subnet named *Cafe Public Subnet*.
35. You have now successfully created the network resources that are needed to run the café website.

Step 5: Updating the network stack

36. Add the following lines to the bottom of *cafe-network.yaml*.

```
Outputs:
 PublicSubnet:
   Description: The subnet ID to use for public web servers
   Value:
     Ref: PublicSubnet
   Export:
     Name:
       'Fn::Sub': '${AWS::StackName}-SubnetID'
 VpcId:
   Description: The VPC ID
   Value:
     Ref: VPC
   Export:
```

Name:

'Fn::Sub': '${AWS::StackName}-VpcID'

37. Save the change and in the Bash terminal, add and commit the code, and then push it to CodeCommit by using Git.
38. Verify that the AWS CloudFormation stack update occurs. Also verify that the Outputs tab now lists two keys with export names.

Step 6: Defining an EC2 instance resource and creating the application stack

39. Back in AWS Cloud9, duplicate the template2.yaml file in the templates directory and rename the duplicate as cafe-app.yaml.
40. In the cafe-app.yaml template, analyze the existing template contents:
    - In the Parameters area, the LatestAmiId performs a lookup. It finds the latest Amazon Linux 2 Amazon Machine Image (AMI) ID in the AWS Region where you create the stack. It can be referenced when you define an Amazon Elastic Compute Cloud (Amazon EC2) instance.
    - Also in the Parameters area, the CafeNetworkParameter defines a string value. The value defaults to the name of the stack that you created when you ran the cafe-network.yaml AWS CloudFormation template. Setting this string as a parameter provides you with the flexibility to point to a different stack name if you must reference resources in another stack.
    - In the Mappings area, the RegionMap mapping can be referenced when you define an EC2 instance. Using this mapping can help ensure that the correct key pair will be used for the instance. However, use of this feature depends on the AWS Region where you run the template.
    - In the Resources area, an EC2 security group is defined. It opens TCP ports 80 and 22 for inbound network traffic. It is created in the VPC that the *update-cafe-network* stack created.
    - In the Outputs area, an output named WebServerPublicIP returns the public IPv4 address of the EC2 instance that you will define next.
41. In the cafe-app.yaml template, define a third *parameter* so that a user can choose between different instance types when they launch an EC2 instance.
    - Browse to the AWS Documentation. Under the *Defining a parameter in a template* section, copy the example YAML parameter.
    - Paste the parameter into your template. Then, modify the parameter so that the permitted instance types are *t2.micro*, *t2.small*, *t3.micro*, and *t3.small*. Also, set the default to t2.small and update the description so that it reflects the options that a user can choose.
42. In a new browser tab, open the AWS Documentation and use the information in that page as a reference.
43. Back in the cafe-app.yaml template, create a new *EC2 instance* resource that has the following characteristics:
    - Set the Logical ID to CafeInstance (see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resources-section-structure.html for reference, if needed)

- Include an ImageId that references the *LatestAmiId* parameter
- For instance type, reference the instance type parameter that you defined in the previous step.
- For KeyName, use the following line of code, which references the RegionMap mapping that is already defined in the template:

```
KeyName: !FindInMap [RegionMap, !Ref "AWS::Region", keypair]
```

- For the instance profile (the AWS Identity and Access Management, or IAM, role that is attached to the instance), specify CafeRole
- In the Properties section, include the following lines of code:

```
NetworkInterfaces:
  - DeviceIndex: '0'
    AssociatePublicIpAddress: 'true'
    SubnetId: !ImportValue
      'Fn::Sub': '${CafeNetworkParameter}-SubnetID'
    GroupSet:
      - !Ref CafeSG
```

- Set a tag with a *key* of *Name* and a *value* of *Cafe Web Server*
- In the Properties section, include the following additional UserData code:

```
 UserData:
    Fn::Base64:
      !Sub |
        #!/bin/bash
        yum -y update
        yum install -y httpd mariadb-server wget
        amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
        systemctl enable httpd
        systemctl start httpd
        systemctl enable mariadb
        systemctl start mariadb
        wget
https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACACAD-2-91555/14-lab-mod10-challenge-CFn/s3/cafe-app.sh
        chmod +x cafe-app.sh
        ./cafe-app.sh
```

44. After you are satisfied with your template updates, save the changes. To validate the template format in the Bash terminal, run the following command:

aws cloudformation validate-template --template-body file:///home/ec2-user/environment/CFTemplatesRepo/templates/cafe-app.yaml

45. If you receive a JSON-formatted response that includes the three parameters that were defined at the top of your template, then your template passed the validation. However, if you received a *ValidationError* response (or some other error response), you must correct the issue. Then, save the changes and run the validate-template command again.
46. If your template passed the validation check, add the file to CodeCommit. In the Bash terminal, run git commands to add the file, commit it, and push it to the repository.
47. Return to the CodePipeline console and choose the CafeAppPipeline. Eventually, the *Deploy* stage status should show *Succeeded - Just now*.
48. In the AWS CloudFormation console, confirm that the *update-cafe-app* stack ran successfully and has a status of *CREATE_COMPLETE*.
49. Go to the Amazon EC2 console. Observe that the EC2 instance and security group resources (which were defined in the cafe-app.yaml template) were created.
50. After the EC2 instance has started and passed both status checks, test the café website. In a browser tab, load the following URL, where <*public-ip-address*> is the *public IPv4 address* of the EC2 instance that you defined: http://<public-ip-address>/cafe
    You should see the café website.

New business requirement: Duplicating the network and application resources in a second AWS Region

Step 7: Duplicating the café network and website to another AWS Region

51. Back in the AWS Cloud9 IDE, run the following command to duplicate the café *network* to another AWS Region:

aws cloudformation create-stack --stack-name update-cafe-network --template-body file:///home/ec2-user/environment/CFTemplatesRepo/templates/cafe-network.yaml --region us-west-2

52. It should return a *StackId*. Notice that you could override the default Region for the creation of this stack by specifying the Region when you ran the command.
53. Browse to the AWS CloudFormation console and change the Region to US West (Oregon) us-west-2.
    - The *update-cafe-network* stack should be listed
    - Verify that the status of the second *update-cafe-region* stack eventually changes to *CREATE_COMPLETE*
54. Browse to the Amazon VPC service page, and also confirm that you are using the Oregon Region (*us-west-2*).
    You should be able to observe the network resources that were created.
55. In the Oregon Region (*us-west-2*), create an EC2 key pair named cafe-oregon.
    - Browse to the Amazon EC2 console and confirm that you are in the Oregon Region.

- From the navigation pane, choose Network & Security > Key Pairs.
- Choose Create key pair.
- Name the key pair cafe-oregon and choose Create key pair again.

56. Revisit the application template details.
    - Return to the AWS Cloud9 IDE and observe the cafe-app.yaml template details in the text editor.
    - Notice the KeyName property in the resource definition for the EC2 instance. It references the RegionMap mapping that is defined in the template.
    - The mapping indicates that if the instance is launched in the *us-east-1 (N. Virginia) Region*, it should use the *vockey key pair*. However, if the instance is launched in the *us-west-2 (Oregon) Region*, it should use the *cafe-oregon key pair* that you just created.
    - Also notice the InstanceTypeParameter that you defined earlier. It provides a few instance type options in the AllowedValues area, but it also sets *t2.small* as the default. You will use this configuration in a moment.
57. In the AWS Cloud9 IDE, copy the template file to an S3 bucket. (In the following command, replace <*repobucket-bucketname*> with the actual S3 bucket name in your account. Its name should contain the string *repobucket*.)

aws s3 cp templates/cafe-app.yaml s3://<repobucket-bucketname>/

58. In the Amazon S3 console, copy the Object URL (which is an *https* address) of the file that you just uploaded.
59. In the AWS CloudFormation console, change the Region to Oregon (*us-west-2*).
60. Create a stack (with new resources).
    - In the Amazon S3 URL box of the Create stack screen, paste the object URL that you just copied.
    - In the next screen (Specify stack details) -
        - Stack name: Enter an appropriate name
        - InstanceTypeParameter: *t3.micro*
    - Advance through the remaining screens (accepting all the default settings), and finish creating the stack.
    - Verify that the stack created successfully.
61. Browse to the Amazon EC2 console and observe the created resources.
    - Be sure to give the web server a few minutes to finish booting and to run the user data script.
    - Notice the *key pair* that is used by the instance, and the instance type. These settings are different than the settings on the web server that runs in the *us-east-1* Region. You used the same template, *without modifying it*, to launch this stack.
    - After the server has fully started, you should be able to access the website at http://<public-ip-address>/cafe (where <*public-ip-address*> is the public IPv4 IP address of the EC2 instance.
    - Notice that the server information on the website shows that this second instance of the café website is running in the *us-west-2* Region. The first web server that you created shows it is running in the *us-east-1* Region.