

Challenge Lab: Automating Infrastructure Deployment

Scenario

Up to this point, the café staff members have created their Amazon Web Services (AWS) resources and manually configured their applications mostly by using the AWS Management Console. This approach worked well as a way for the café to get started with a web presence quickly. However, the staff members are finding it challenging to replicate their deployments to new AWS Regions so that they can support new café locations in multiple countries. They would also like to have separate development and production environments that reliably have matching configurations.

In this challenge lab, you take on the role of Sofía as you work to automate the café's deployments and replicate them to another AWS Region.

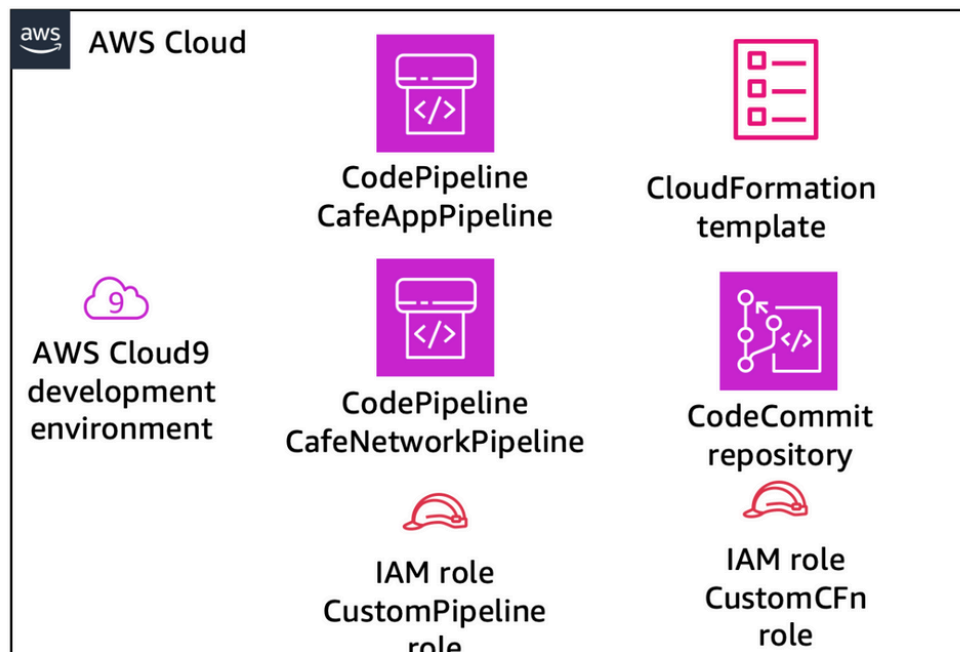
Lab overview and objectives

In this lab, you gain experience with creating AWS CloudFormation templates. You use the templates to create and update CloudFormation stacks. The stacks create and manage updates to resources in multiple AWS service areas in your AWS account. You practice by using AWS CodeCommit to control the version of your templates. You also observe how you can use AWS CodePipeline to automate stack updates.

After completing this lab, you should be able to do the following:

- Deploy a virtual private cloud (VPC) networking layer by using a CloudFormation template.
- Deploy an application layer by using a CloudFormation template.
- Use Git to invoke CodePipeline and to create or update stacks from templates that are stored in CodeCommit.
- Duplicate network and application resources to another AWS Region by using CloudFormation.

When you start the lab, the following resources are already created for you in the AWS account:



In this challenge lab, you will encounter a few tasks where step-by-step instructions are not provided. You must figure out how to complete the tasks on your own.

Duration

This lab requires approximately **90 minutes** to complete.

A business request: Creating a static website for the café by using CloudFormation (challenge 1)

The café would like to start using CloudFormation to create and maintain resources in the AWS account. As a first attempt at this process, you take on the role of Sofía and create a CloudFormation template that can be used to create an Amazon Simple Storage Service (Amazon S3) bucket. Then, you add more detail to the template so that when you update the stack, it configures the bucket to host a static website for the café.

Task 1: Creating a CloudFormation template from scratch

In this task, you create a CloudFormation template that creates an S3 bucket. You then run an AWS Command Line Interface (AWS CLI) command that created the CloudFormation stack. (The stack is the resource that creates the bucket.)

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

`AWS::TemplateFormatVersion: "2010-09-09"`

Description:

1. Next, add the following three lines to your file:

Resources:

S3Bucket:

Type: `AWS::S3::Bucket`

Tip: Make sure that you keep the correct number of spaces for each indentation level. The **Resources** line should have no indentation. The **S3Bucket** line should be indented by two spaces. Finally, the **Type: AWS::S3::Bucket** line should be indented by four spaces.

CloudFormation supports the YAML version 1.1 specification with a few exceptions. For more information about YAML, see [the YAML website](#).

1. 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 (:).
2. After you enter the description, save the changes to the file.

In the guided lab earlier in this module, you used the AWS Management Console to create a CloudFormation stack. Here, you use the AWS CLI instead.

1. In the Bash terminal, run the following two lines of code:

```
aws configure get region
```

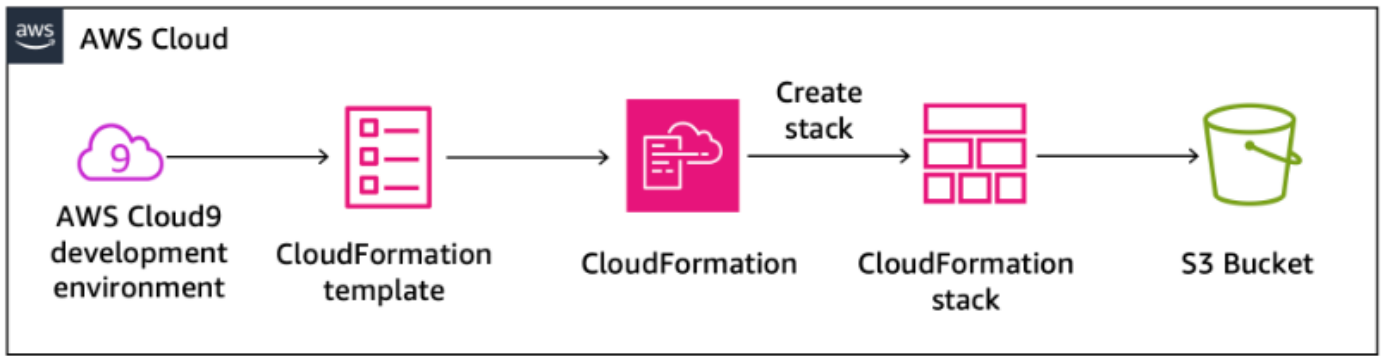
```
aws cloudformation create-stack --stack-name CreateBucket --template-body file:///S3.yaml
```

The first line of code returns the default AWS Region of the AWS CLI client that is installed on the AWS Cloud9 instance. You can modify the default AWS Region by running the `aws configure` command. However, for this lab, you should leave the default Region.

The second line of code creates a stack that uses the template you defined. Because you did not specify the Region in the command, the stack is created in the default Region.

If the `create-stack` command ran successfully, you should see some output that is formatted in JSON. This output should indicate a **StackId**.

The following diagram illustrates the actions that you just completed.



1. In the AWS Management Console, navigate to the CloudFormation console, and observe the details of the **CreateBucket** stack.
2. For example, look at the information in the **Events**, **Resources**, **Outputs**, and **Template** tabs.
3. Navigate to the Amazon S3 console to observe the bucket that your template created.

Tip: The bucket has the bucket name **createbucket-s3bucket-`<random-string>`**.

Answering questions about the CloudFormation stack

Your answers are recorded when you choose **Submit** at the end of the lab.

1. To access the questions in this lab, at the top of these instructions, choose **AWS Details**.
2. Choose the **Access the multiple choice questions** link.
3. In the page that you loaded, submit answers to the following questions:
 - **Question 1:** Was an S3 bucket created, even if you did not specify a name for the bucket? If so, what name was it given?
 - **Question 2:** What Region was the bucket created in, and why was it created in this Region?
 - **Question 3:** To define an S3 bucket, how many lines of code did you need to enter in the 'Resources:' section of the template file?

Note: Leave the browser tab with the questions open so that you can return to it later in the lab.

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

In this next task, you update the CloudFormation template. The update configures the S3 bucket to host a static website. This task is similar to the results from the module 3 challenge lab. In that challenge lab, you created and configured the S3 bucket manually by using the AWS Management Console. However, in this lab, you configure the bucket by using a CloudFormation template.

Next, you set bucket ownership controls and public access, and then upload the static website assets to the bucket.

1. To complete this task, run the following commands in the Bash terminal (replace all three occurrences 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-3-113230/15-lab-mod11-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
```

If these operations are successful, you should see numerous `upload:<file_name>` messages in the command output.

Next, you open the CloudFormation template documentation for defining S3 bucket resources.

1. Go to the [AWS Resource and Property Types Reference](#) documentation.
2. From the **Service resource type** list, choose **Amazon S3**.
3. From the **Resource types** list, choose **AWS::S3::Bucket**.
4. By 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 retains the bucket.
 - Configure the bucket to host a static website with index.html set as the index document.
 - **Tip:** You can accomplish this task by adding as few as four additional lines of code to your template. See the code in the **Examples** section of the documentation page that you opened in the previous steps.
5. To your CloudFormation template, add an output that provides the website URL.
6. Again, consult the **Examples** section of the documentation as a reference.
7. Save the changes to your S3.yaml file.

- Next, you validate your template.
- 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
```

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.

```
voclabs:~/environment $ aws cloudformation validate-template --template-body file://S3.yaml
{
  "Parameters": [],
  "Description": "cafe S3 template"
}
```

- To update the stack, run the following command:
- ```
aws cloudformation update-stack --stack-name CreateBucket --template-body file://S3.yaml
```
- Tip:** Proper YAML syntax is important. If you receive a *ValidationError* message when you run the update-stack command, review your use of colons and confirm that you indented each line appropriately. The example templates in the documentation provide a good reference for well-structured YAML templates.
- Browse to the CloudFormation console, and confirm that your stack update completed successfully.
  - The stack should soon show a **Status** of *UPDATE\_COMPLETE*.

The screenshot displays the AWS CloudFormation console. On the left, the 'Stacks (3)' list shows a stack named 'CreateBucket' with a status of 'UPDATE\_COMPLETE'. The main panel shows the 'CreateBucket' stack details, with the 'Outputs' tab selected. The output table lists the 'WebsiteURL' as 'http://createbucket-s3bucket-dt5p0hwxbk3d.s3-website-us-east-1.amazonaws.com'.

| Key        | Value                                                                                                                                                                   | Description                  | Export name |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|-------------|
| WebsiteURL | <a href="http://createbucket-s3bucket-dt5p0hwxbk3d.s3-website-us-east-1.amazonaws.com">http://createbucket-s3bucket-dt5p0hwxbk3d.s3-website-us-east-1.amazonaws.com</a> | URL for website hosted on S3 | -           |

YOU WILL see

# Café



The Café offers an assortment of delicious and delectable pastries and coffees that will put a smile on your face. From cookies to croissants, tarts and cakes, each treat is specially prepared to excite your tastebuds and brighten your day!

Frank bakes a rich variety of cookies. Try them all!

Tea

Our tarts are always a customer favorite!

- If the stack does not attain a status of *UPDATE\_COMPLETE*, try the following troubleshooting tips.
  - If you see that the stack has a *ROLLBACK* status of some kind, choose the **Events** tab, and search for an **UPDATE\_FAILED** entry. (Read the **Status reason** for that event to understand why the stack update failed.)
    - After you think that you resolved any errors, run the update-stack command again. In the console, return to the CloudFormation stack, and choose the **Events** tab to confirm whether you successfully updated the stack.
    - Repeat as necessary.
1. To verify success, answer the following questions:
- Does the stack's **Outputs** tab list an output with a URL value? If so, choose the link.
  - Does the static website open? (You previously copied the website assets into the bucket.)
  - **Note:** If the stack does not have any output or if the output hyperlink does not display the contents of the café website, you can try the following troubleshooting steps.
    - Browse to the Amazon S3 console, and choose your bucket. The **Objects** tab should list the **index.html** file and two folders that are named **css** and **images**. If these resources are not listed, revisit the first step in this challenge section.
    - Choose the **index.html** file, and then choose **Permissions**. For **Everyone (Public access)**, the value in the **Object** column should be **Read**.
    - Return to the bucket view. In the **Properties** tab, confirm that **Static website hosting** is **Enabled**, with a **Hosting type** of **Bucket hosting**.
  - All of the permissions and properties that are described in this list were either set by the AWS CLI S3 commands you ran or were set in your S3.yaml template. If necessary, adjust the details in the template and run the update-stack AWS CLI command again.
  - **Note:** In this first challenge, you manually copied the website files into the bucket. You can also perform this action by using a custom resource from CloudFormation, combined with an AWS Lambda function. Both of these resources can be defined in a



CloudFormation template. This approach is a more advanced use of CloudFormation beyond the scope of this lab. However, if you are interested in this topic, see the [AWS Lambda-Backed Custom Resources](#) page in the AWS documentation.

## New business requirement: Storing templates in a version control system (challenge 2)

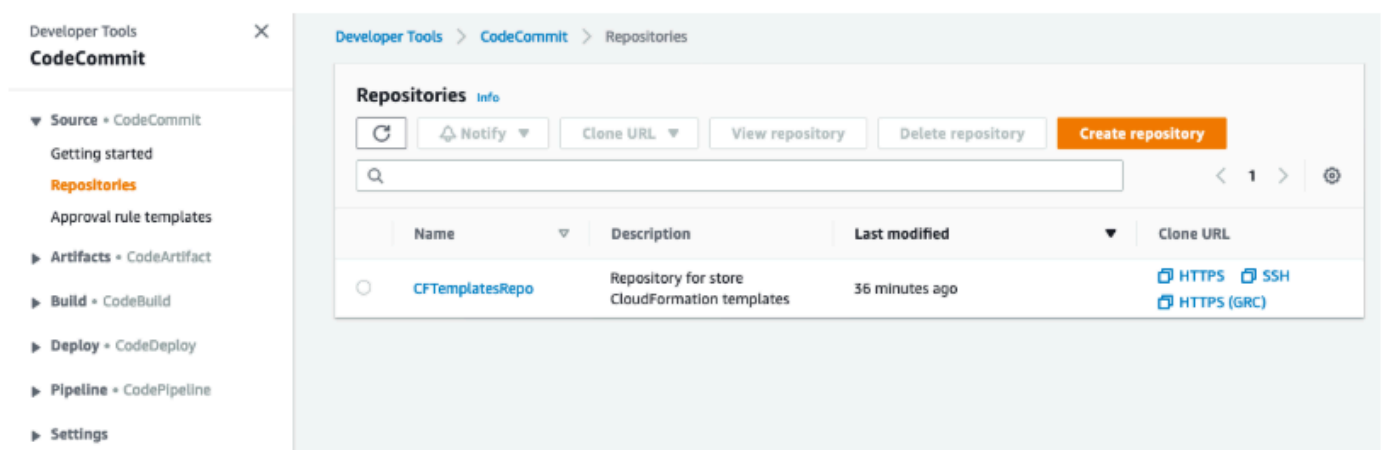
The café team is impressed that Sofía configured an entire static website by using a CloudFormation template. Given this success, the team decided that they would like to expand their use of infrastructure as code (IaC) to build out other application resources in the AWS account.

The team understands that it is a best practice to store IaC templates in a version control system, so they asked Sofía to take on this challenge. Sofía spoke with Mateo about this new business requirement when he stopped by the café. He mentioned that CodeCommit would be a good choice for storing templates and managing version control for them. Mateo created a CodeCommit repository with some sample CloudFormation templates in it. Sofía is eager to start using this code repository.

### Task 3: Cloning a CodeCommit repository that contains CloudFormation templates

In this task, you work as Sofía to clone a CodeCommit repository. The café team will use the repository to store and control the versions of the CloudFormation templates.

1. Browse to the CodeCommit console, and in your account, notice the repository that is named **CFTemplatesRepo**.
2. Your CodeCommit repository should look similar to the following.



CodeCommit is a source control service that you can use to host Git-based repositories. It can be used in a way that's similar to GitHub repositories. For more information about



CodeCommit, see the [AWS CodeCommit Documentation](#).

1. Choose **CFTemplatesRepo**.
2. Choose the **templates** folder.
3. Notice that the folder has CloudFormation templates in it.
4. In this part of the lab, you store your IaC CloudFormation templates in CodeCommit.
5. 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.
  - For example, notice the following:
    - Starting on line 6, the template defines an AWS Cloud9 instance.
    - Starting on line 12, the template defines the CodeCommit repository that you now have open.
  - The lab platform that hosts this lab created a CloudFormation stack when you started the lab. The CloudFormation template that it ran includes the resource definitions that are contained in this template. However, this example template does not contain all the resource definitions that are in the actual template that was used to start this lab.
6. In the breadcrumbs at the top of the page, choose **Repositories**.
7. Select **CFTemplatesRepo**, choose **Clone URL**, and then choose **Clone HTTPS**.

This action copies the CodeCommit repository's HTTPS clone URL to your clipboard.

1. Return to the AWS Cloud9 IDE.
2. To clone the existing CodeCommit repository to your workspace, enter the following command into the Bash terminal. In the command, replace `<url>` with the clone URL that you copied.

```
git clone <url>
```

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.

1. To use the Git client software to analyze your local copy of the repository, enter the following commands:

```
cd CFTemplatesRepo
```

```
git status
```

Earlier command provides the output similar to the following.

```
bash - "ip-172-31-29-19" x Immediate x (+)
voclabs:~/environment $ git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/CFTemplatesRepo
Cloning into 'CFTemplatesRepo'...
remote: Counting objects: 6, done.
Unpacking objects: 100% (6/6), 1.52 KiB | 780.00 KiB/s, done.
voclabs:~/environment $ cd CFTemplatesRepo/
voclabs:~/environment/CFTemplatesRepo (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
voclabs:~/environment/CFTemplatesRepo (main) $
```

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 (challenge 3)

The next challenge is for Sofía to use CloudFormation to create all the network resources that the dynamic website café application can be deployed to. Then, she must deploy the café application itself.

Also, Sofía would like to find an easier way to update stacks when she updates a CloudFormation template. She is now updating templates regularly, and she thinks that she should be able to automate stack updates.

Sofía spoke with Mateo about this issue. He mentioned that CodePipeline provides the continuous integration and continuous delivery (CI/CD) service capabilities that she is looking for. Mateo then created two pipelines for Sofía, and she is eager to start working with them.

In this challenge, you work as Sofía and make use of these pipelines. You also define—in CloudFormation templates—all the resources that are needed to deploy the dynamic café website.

### Task 4: Creating a new network layer with CloudFormation, CodeCommit, and CodePipeline

In this task, you use a CloudFormation template to create a VPC with a public subnet along with other network resources. You gain experience with using a CI/CD pipeline. When you use Git to push the template into a CodeCommit repository, it activates a pipeline that creates a CloudFormation stack.

1. In the navigation pane of the AWS Cloud9 IDE, expand the **CFTemplatesRepo/templates** directory.
2. In the **templates** directory, select (right-click) **template1.yaml**, and create a duplicate of the file.
3. Rename the duplicate **cafe-network.yaml**.
4. In the text editor, open the **cafe-network.yaml** file, and for the description, enter Network layer for the cafe, and save your changes.
5. Observe the details of the seven resources that this template creates.
6. Next, you observe the CodePipeline details that were preconfigured in your account.
7. At the top of the AWS Management Console, in the search box, enter and choose CodePipeline to open the CodePipeline console.
8. Choose **Pipelines**.

Notice that two pipelines have been predefined for you:

- CafeAppPipeline
- CafeNetworkPipeline

**Important:** The status of the most recent attempt to run each pipeline shows that they *Failed*. However, this status is expected. The CloudFormation template files that the pipelines reference do not exist in their expected location.

Next, you analyze the source stage of CafeNetworkPipeline.

1. To observe the pipeline details, choose **CafeNetworkPipeline**.

In the **Source** section, you can see that this pipeline's **SourceAction** is **AWS CodeCommit**.

1. In the **SourceAction** section, choose **View details**.
2. Choose **Configuration**.
3. This tab shows that the source is the **CFTemplatesRepo** CodeCommit repository.
4. To return to the **CafeNetworkPipeline** page, choose **Done**.
5. Analyze the **Deploy** stage of **CafeNetworkPipeline**.
6. Notice that the **Deploy** action will be performed by using CloudFormation.
7. In the **RunChangeSet** section, choose **View details**.

**Analysis:** The details on the **Configuration** tab show that a stack named **update-cafe-network** will be run or be updated. To perform these actions, the stack will use the **cafe-network.yaml** CloudFormation template. This Deploy action receives the template from the Source stage, which found the template in the CodeCommit repository.

Developer Tools > CodePipeline > Pipelines > CafeNetworkPipeline

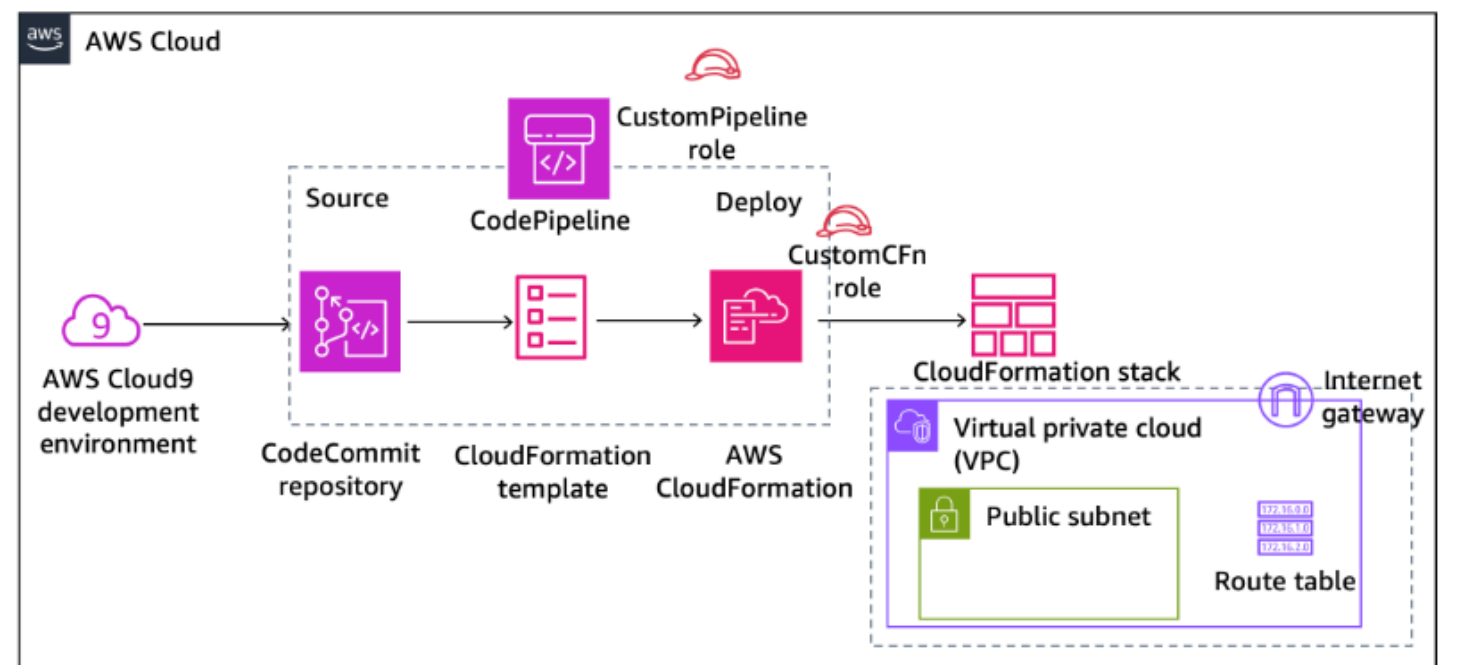
## CafeNetworkPipeline

Edit Stop execution

Pipeline Executions Triggers Settings Tags Stage

The screenshot shows the AWS CodePipeline console for the 'CafeNetworkPipeline'. The pipeline has two stages: 'Source' and 'Deploy'. The 'Source' stage is successful, with the action 'SourceAction' using 'AWS CodeCommit' as the provider. The 'Deploy' stage has failed, with the action 'RunChangeSet' using 'AWS CloudFormation' as the provider. The failure message indicates '1 of 1 action failed'. The console also shows the pipeline ID '89e10f06-fd6d-4777-a2be-6253634e4ecf' and the execution ID '355550fb'.

The following diagram illustrates how you will activate this pipeline and what the pipeline will do. It also shows some of the AWS account resources that the resulting CloudFormation stack will create or update.



For more information about CodePipeline, see the [AWS CodePipeline Documentation](#).

1. Return to the AWS Cloud9 IDE.

Next, you invoke the creation of the **update-cafe-network** by checking your CloudFormation template into CodeCommit.

1. To 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.

```
voclabs:~/environment/CFTemplatesRepo (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
 (use "git add <file>..." to include in what will be committed)
 templates/cafe-network.yaml

nothing added to commit but untracked files present (use "git add" to track)
```

1. To add the new file to the repository and then commit it to the repository with a comment, run the following commands:

```
git add templates/cafe-network.yaml
```

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

1. To check the status of your local copy of the repository, run the following command:

```
git status
```

The information that is returned should report that your branch is **ahead of origin/main by 1 commit**.

1. Finally, to push the commit to the remote repository, run the following command:

```
git push
```

```

voclabs:~/environment/CFTemplatesRepo (main) $ git add templates/cafe-network.yaml
voclabs:~/environment/CFTemplatesRepo (main) $ git commit -m 'initial commit of network template' templates/cafe-network
[main bb05dcc] initial commit of network template
Committer: EC2 Default User <ec2-user@ip-172-31-53-89.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

 git config --global user.name "Your Name"
 git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

 git commit --amend --reset-author

1 file changed, 72 insertions(+)
create mode 100644 templates/cafe-network.yaml
voclabs:~/environment/CFTemplatesRepo (main) $ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
voclabs:~/environment/CFTemplatesRepo (main) $ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 816 bytes | 816.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/CFTemplatesRepo
355550f..bb05dcc main -> main
voclabs:~/environment/CFTemplatesRepo (main) $

```

This command actually copies the file to CodeCommit.

1. Return to the CodePipeline console, and choose **CafeNetworkPipeline**.

Observe that the creation of the stack is automatically activated.

**Note:** It might take a minute or two for the **Source** stage to update and for the **Deploy** stage to show that it is *In progress*. Eventually, the **Deploy** stage status should show *Succeeded*.

Developer Tools > CodePipeline > Pipelines > CafeNetworkPipeline

## CafeNetworkPipeline

Edit Stop execution

Pipeline Executions Triggers Settings Tags Stage

✓ Source

83a24479-f60e-43e9-a634-4241e4574b6f

All actions succeeded.

SourceAction

✓ AWS CodeCommit

1 minute ago

bb05dcce SourceAction: initial co ...

✓ Deploy

83a24479-f60e-43e9-a634-4241e4574b6f

All actions succeeded.

RunChangeSet

✓ AWS CloudFormation

1 minute ago

bb05dcce SourceAction: initial co ...

- Notice that the details for both **Source** and **Deploy** show the commit number that was returned when you ran the git push command. The details also show the comment that you added to the commit.
- Your Pipeline shows details similar to the following.

#### Troubleshooting tips:

- If the **Deploy** step has a status of *Failed - Just now*, access the error details by opening the **Execution details** link. For example, you could have a template-formatting error that must be resolved.
  - After you update the template, you can update the stack by running the appropriate git commit and git push commands again.
    - You can also choose **Release change** to activate the pipeline to run again. It does so even if you do not make changes to the CodeCommit repository (such as by issuing a git push command).
    - Similarly, you can use the **Retry** button in the **Deploy** stage of the pipeline. It will retry the **Deploy** stage without retrying the **Source** stage.
  - If the stack fails to roll back and prevents you from performing additional updates to the stack, you can delete the stack. To do so, go to the stacks page in the CloudFormation console and delete the stack. If you delete the network stack, push a new update to Git. This action activates the re-creation of the stack.
1. In the CloudFormation console, confirm that the **update-cafe-network** stack ran. It should have a **Status** of *CREATE\_COMPLETE* or *UPDATE\_COMPLETE*.
  2. Also, check the **Outputs** tab for the stack. It currently shows no outputs. Soon, however, you update the stack so that it creates outputs.
  3. Go to the Amazon VPC console, and observe that the resources defined in the cafe-network.yaml template were created in the AWS account.
  4. For example, the console should list a VPC named **Cafe VPC** and a subnet named **Cafe Public Subnet**.
  5. Congratulations! You have successfully created the network resources that are needed to run the café website.

vpc-0e84ffb9842b3fef1 / Cafe VPC



## Task 5: Updating the network stack



In this task, you update the network stack so that it exports essential information about two of the resources that it creates. These two outputs can then be referenced by the application stack that you create later.

1. In the AWS Cloud9 IDE, add the following lines to the bottom of **cafe-network.yaml**, and save your changes.

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'

1. In the Bash terminal, commit the code, and then push it to CodeCommit by using Git.
2. Verify that the CloudFormation stack update occurs.
3. Verify that the **Outputs** tab now lists two keys with export names.

| Name         | Export Name                  |
|--------------|------------------------------|
| PublicSubnet | update-cafe-network-SubnetID |
| VpcId        | update-cafe-network-VpcID    |

Following image shows the output after updating the CloudFormation stack.

update-cafe-network

Delete

Update

Stack actions ▼

Create stack ▼

Stack Info

Events

Resources

Outputs

Parameters

Template

Change sets

Outputs (2)

🔍 Search outputs

⚙️

| Key ▲        | Value ▼                  | Description ▼                               | Export name ▼                |
|--------------|--------------------------|---------------------------------------------|------------------------------|
| PublicSubnet | subnet-013645777108dae96 | The subnet ID to use for public web servers | update-cafe-network-SubnetID |
| VpcId        | vpc-0d816f213d93fe4c6    | The VPC ID                                  | update-cafe-network-VpcID    |

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

In this task, you create a new CloudFormation template that will be used to create a stack. The new stack deploys a dynamic website for the café. The **CafeAppPipeline** pipeline (which you observed earlier) creates or updates the **update-cafe-app** stack when you push the **cafe-app.yaml** template to the CodeCommit repository.

1. In AWS Cloud9, duplicate the **template2.yaml** file in the templates directory, and rename the duplicate **cafe-app.yaml**.
2. 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** 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 define next.

Next, in the `cafe-app.yaml` template, you define a third parameter so that a user can choose between different instance types when they launch an EC2 instance.

1. In the [AWS CloudFormation Documentation](#), in the **Defining a parameter in a template** section, copy the example YAML parameter.
2. Paste the parameter into your template, make the following adjustments, and then save your changes.
  - Modify the parameter so that the permitted instance types are **t2.micro**, **t2.small**, **t3.micro**, and **t3.small**.
  - Change the default to **t2.small**.
  - Update the description so that it reflects the options that a user can choose.

```
13
14 InstanceTypeParameter:
15 Description: Enter t2.micro, t2.small, t3.small, or t3.micro. Default is t2.micro.
16 Type: String
17 Default: t2.micro
18 AllowedValues:
19 - t2.micro
20 - t2.small
21 - t3.micro
22 - t3.small
23 Mappings:
```

1. In a new browser tab, open the [AWS CloudFormation Documentation](#), and use the information in that page as a reference for the next steps.
2. In the **cafe-app.yaml** template, create a new EC2 instance resource with the following configuration settings:
  - For **Logical ID**, enter `CafeInstance`. For more information, see [Resources](#) in the *AWS CloudFormation User Guide*.
  - Include an **ImageId** that references the **LatestAmiId** parameter.
  - For **InstanceType**, 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 **IamInstanceProfile** (the AWS Identity and Access Management, or IAM, role that is attached to the instance), specify `CafeRole`.
- **Note:** The `CafeRole` IAM role already exists in your account. Attaching it grants your EC2 instance the permissions to retrieve Parameter Store values from AWS Systems Manager.
- In the **Properties** section, include the following lines of code:
  - **NetworkInterfaces:**
    - `- DeviceIndex: '0'`
    - `AssociatePublicIpAddress: 'true'`
    - `SubnetId: !ImportValue`
    - `'Fn::Sub': '${CafeNetworkParameter}-SubnetID'`
    - **GroupSet:**
      - `- !Ref CafeSG`

These lines help ensure that your instance deploys to the Public Subnet that you created when you ran the café network stack. Recall that at the beginning of this task, you updated the network stack to define outputs with export names. In the preceding code, you import the value for the **SubnetId**. The preceding code also helps ensure that the instance you create will be in the CafeSG security group that is already defined for you in this template.

- Set a **Tag** with a **Key** of **Name** and a **Value** of **Cafe Web Server**.

**Tip:** Observe how a **Name** tag was applied to the security group resource that is already defined in the template.

- 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-3-113230/15-lab-mod11-challenge-CFn/s3/cafe-app.sh**
- **chmod +x cafe-app.sh**
- **./cafe-app.sh**

```
24
25 RegionMap:
26 us-east-1:
27 "keypair": "vockey"
28 us-west-2:
29 "keypair": "cafe-oregon"
30
31 Resources:
32 CafeInstance:
33 Type: AWS::EC2::Instance
34 Properties:
35 ImageId: !Ref LatestAmiId
36 InstanceType: !Ref InstanceTypeParameter
37 KeyName: !FindInMap [RegionMap, !Ref "AWS::Region", keypair]
38 IamInstanceProfile: CafeRole
39 NetworkInterfaces:
40 - DeviceIndex: '0'
41 AssociatePublicIpAddress: 'true'
42 SubnetId: !ImportValue
43 'Fn::Sub': '${CafeNetworkParameter}-SubnetID'
44 GroupSet:
45 - !Ref CafeSG
46 Tags:
47 - Key: Name
48 Value: Cafe Web Server
49 UserData:
50 Fn::Base64:
51 !Sub |
52 #!/bin/bash
53 yum -y update
54 yum install -y httpd mariadb-server wget
55 amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
56 systemctl enable httpd
57 systemctl start httpd
58 systemctl enable mariadb
59 systemctl start mariadb
60 wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACACAD-3-113230/15-lab-mod11-challenge-CFn/s3/cafe-app.sh
61 chmod +x cafe-app.sh
62 ./cafe-app.sh
63
```

- These lines of code run on the instance at the end of the boot process. It installs an Apache HTTP web server, a MariaDB database, and PHP on the Amazon Linux instance. Next, it starts the web server and the database. Then, it downloads a script named `cafe-app.sh` and runs it. The `cafe-app` script configures the database and installs the PHP code that makes the café website function.
1. After you are satisfied with your template updates, save the changes.
  2. 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
```

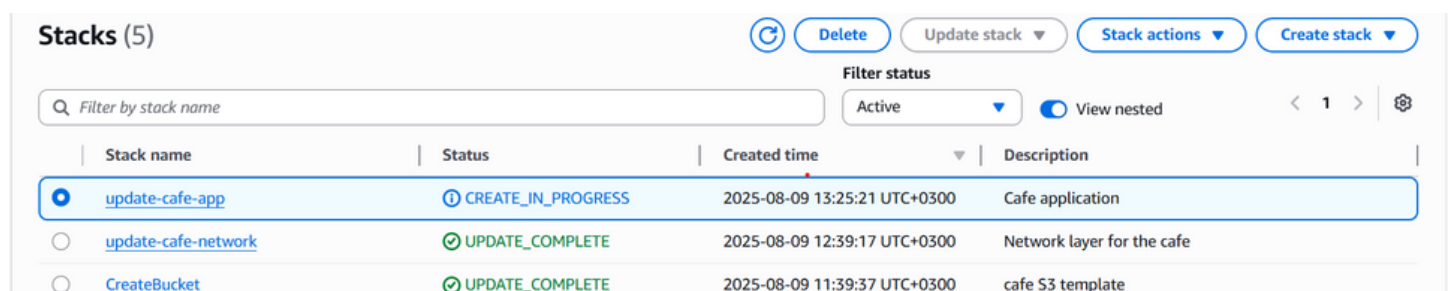
If you receive a JSON-formatted response that includes the three parameters that were defined at the top of your template, then your template passed the validation. However, if you received a *ValidationError* response (or some other error response), you must correct the issue. Then, save the changes and run the `validate-template` command again.

1. 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.

**Tip:** If it helps, refer back to the Git commands in task 3. However, remember that the name of the template that you want to push to CodeCommit for this task is different.

1. Return to the CodePipeline console, and choose **CafeAppPipeline**.

**Note:** It might take a minute or two for the **Source** stage to update and for the **Deploy** stage to show that it is *In progress*. Eventually, the **Deploy** stage status should show *Succeeded - Just now*.



The screenshot shows the AWS CloudFormation 'Stacks' console. At the top, there are buttons for 'Delete', 'Update stack', 'Stack actions', and 'Create stack'. Below these is a search bar 'Filter by stack name' and a 'Filter status' dropdown set to 'Active'. A 'View nested' toggle is also present. The main table lists three stacks:

| Stack name                          | Status             | Created time                 | Description                |
|-------------------------------------|--------------------|------------------------------|----------------------------|
| <a href="#">update-cafe-app</a>     | CREATE_IN_PROGRESS | 2025-08-09 13:25:21 UTC+0300 | Cafe application           |
| <a href="#">update-cafe-network</a> | UPDATE_COMPLETE    | 2025-08-09 12:39:17 UTC+0300 | Network layer for the cafe |
| <a href="#">CreateBucket</a>        | UPDATE_COMPLETE    | 2025-08-09 11:39:37 UTC+0300 | cafe S3 template           |

If the status shows a failure, try these troubleshooting tips:

- If you see that the **Deploy** stage has a status of *Failed - Just now*, choose **View details** to review the error information. It might provide a link that takes you to the CloudFormation stack details. Go to the **Events** tab to figure out which error was the first one that caused the stack to roll back.
- If the stack fails to roll back, or if it has a *ROLLBACK\_COMPLETE* status that prevents you from updating the stack with the pipeline, you can delete the stack from the

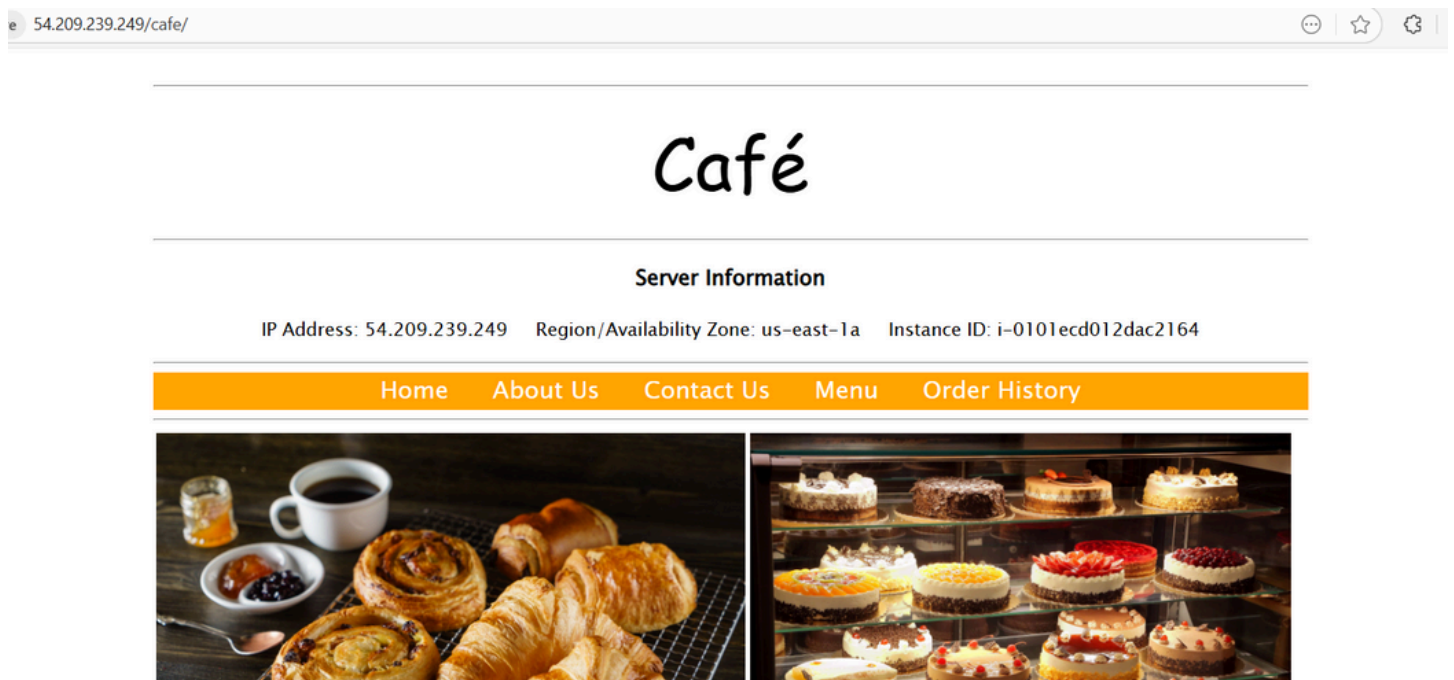
CloudFormation stacks page. Then, push a new update to Git to activate the stack to be created again.

1. In the CloudFormation console, confirm that the **update-cafe-app** stack ran successfully and has a status of *CREATE\_COMPLETE*.
2. In the Amazon EC2 console, observe that the EC2 instance and security group resources (which were defined in the cafe-app.yaml template) were created.

After the EC2 instance has started and passed both status checks, you test the café website.

1. 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.



**Tip:** It can take 2 minutes or so for the user data script details that you defined to finish running. Be patient if you do not see the website immediately.

Notice that the website shows server information, such as the Region and Availability Zone where the web server is running.

Congratulations! You deployed a network layer and an application layer by using a CI/CD pipeline and an IaC approach.

### Answering questions about the results of creating an application layer

1. Return to the browser tab with the multiple-choice questions for this lab, and answer the following questions:
  - **Question 4:** Go to the Parameters tab of the update-cafe-app stack. What value do you see for the LatestAmild?



- **Question 5:** Go to the Stack info tab of the update-cafe-app stack. What is the Amazon Resource Name (ARN) of the IAM role that grants the permissions to create and update the update-cafe-app stack?
- **Question 6:** In the AWS Management Console, navigate to the CodeCommit repository where your AWS CloudFormation templates are stored. Choose Commits and in the Commits list, open one of the commits by choosing its commit ID. What you do observe?

## New business requirement: Duplicating the network and application resources in a second AWS Region (challenge 4)

Sofía is pleased that she was able to create both the network layer and the application layer for the dynamic café website by using CloudFormation. Sofía also just learned that the café staff would like her to duplicate these resources into a second AWS Region, so she is even more pleased.

Sofía will soon experience the benefits of the hard work that she did to define the resources and configurations in CloudFormation templates. She will observe that it is easier to duplicate environments through an IaC approach instead of creating all the resources manually.

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

In this final lab task, you experience how quickly you can duplicate a deployment. A quick deployment is possible because you defined all your resources in CloudFormation templates.

In tasks 4, 5, and 6, the CloudFormation stacks were created or updated automatically. A pipeline was defined to monitor when the CodeCommit repository was updated. It then invoked CloudFormation to create or update the stack. However, in this task, you use the AWS CLI to duplicate the café network resources in another AWS Region. Then, you use the CloudFormation console to create the application stack in the second Region.

1. In the AWS Cloud9 IDE, to duplicate the café network to another AWS Region, run the following command:

```
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
```

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.

1. Browse to the 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**.



**Tip:** Use the refresh icon to see the status change more quickly.

1. Browse to the Amazon VPC console, and confirm that you are in the **US West (Oregon) us-west-2** Region.
2. You should be able to observe the network resources that were created.
3. Next, you create an Amazon EC2 key pair.
4. Browse to the Amazon EC2 console, and confirm that you are in the **US West (Oregon) us-west-2** Region.
5. In the left navigation pane, choose **Key Pairs**.
6. Choose **Create key pair**.
7. For **Name**, enter `cafe-oregon`.
8. Choose **Create key pair** again.
9. **Tip:** Optionally, you can save the key pair, or you can choose **Cancel**. You don't need to use the key pair in this lab. However, in a typical use case, you must save the key pair; you will not have another opportunity to do so.
10. 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 instance type parameter 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 use this configuration in a moment.
  - Next, you copy the template file to an S3 bucket.
11. In the AWS Cloud9 IDE, enter the following command. In the command, replace `<repobucket-bucketname>` with the actual S3 bucket name in your account. Its name should contain the string **repobucket**.

c168135a4327682111159420t1w666096111048-repobucket-3kyitmukgmsk [Info](#)

[Objects](#) | [Metadata](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

**Objects** (2) Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

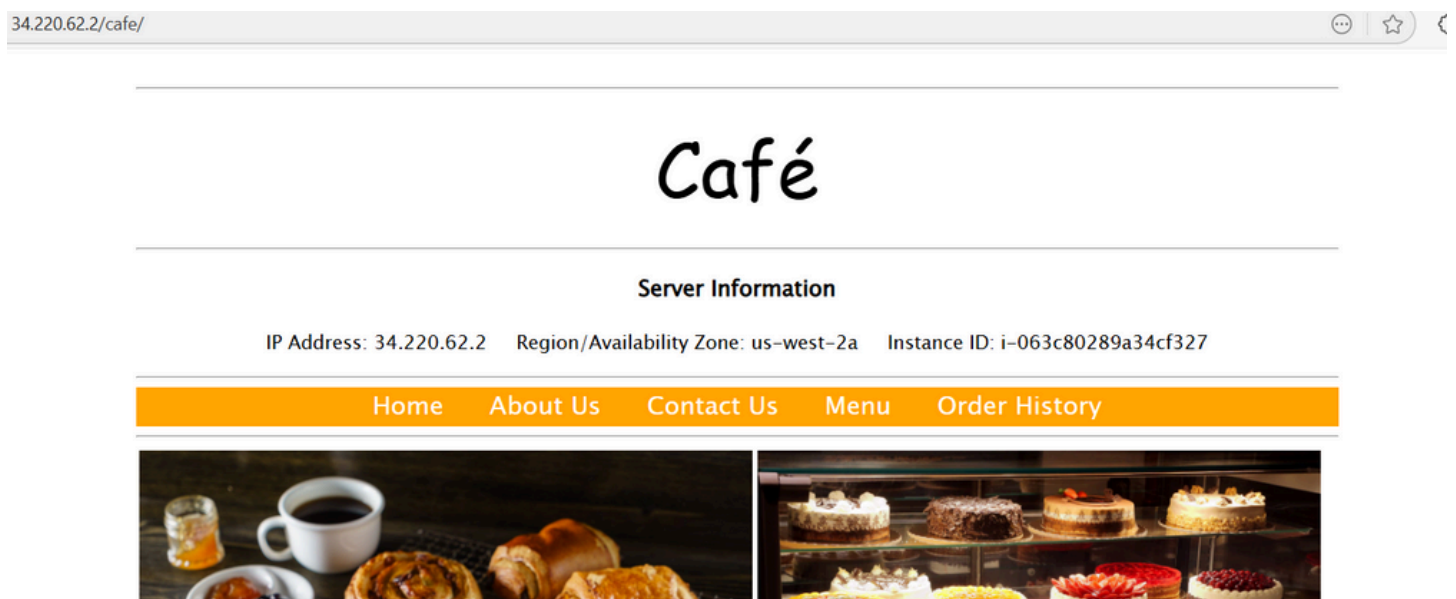
| <input type="checkbox"/> | Name                            | Type | Last modified                        | Size   | Storage class |
|--------------------------|---------------------------------|------|--------------------------------------|--------|---------------|
| <input type="checkbox"/> | <a href="#">cafe-app.yaml</a>   | yaml | August 9, 2025, 13:38:06 (UTC+03:00) | 2.3 KB | Standard      |
| <input type="checkbox"/> | <a href="#">m10cLabRepo.zip</a> | zip  | August 9, 2025, 11:28:58 (UTC+03:00) | 1.9 KB | Standard      |

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

1. In the Amazon S3 console, select the object that you just uploaded, and choose **Copy URL** to copy the object URL of the file.
2. In the CloudFormation console, ensure the Region is **US West (Oregon) us-west-2**.
3. Choose **Create stack**, and then choose **With new resources (standard)**.
4. On the **Create stack** page, for **Amazon S3 URL**, enter the object URL that you just copied.
5. Choose **Next**.
6. On the **Specify stack details** page, configure the following options:
  - **Stack name**: Enter a name.
  - **InstanceType**: Choose **t3.micro**.

Notice that you can set the instance type at stack creation because you defined it as a parameter in the CloudFormation template.

1. Continue through the remaining screens, accept all the default settings, and finish creating the stack.
2. Verify that the stack was created successfully.
3. 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.



## Update from the café

Sofía is full of ideas as a result of what she just learned how to do.

She used CloudFormation to deploy a static version of the café website successfully. She then deployed the dynamic café website as a web application successfully. For the dynamic website, Sofía used a CI/CD pipeline that used CodeCommit, CodePipeline, and CloudFormation. In addition, she quickly duplicated both the network resources and the café application resources to another AWS Region.

Sofía imagines how she could use the CloudFormation templates as part of a backup and disaster recovery (DR) solution. She experienced how quickly she was able to recreate the essential café infrastructure. If her production deployment ever experiences a failure for any reason, she can now recreate it or duplicate it quickly.

Sofía also thinks about how she can now spin up test environments quickly. She can be confident that the configuration details of the test environments will match the production environment. She can also specifically control the ways that the test environment differs from the production environment by using features such as parameters and mappings.

The benefits of DevOps automation are many. Sofía now plans to make automation and CI/CD pipelines central to the way her team develops, tests, and deploys updates to the café's cloud resources.

reference:

[DeletionPolicy attribute - AWS CloudFormation](#)

[AWS::S3::Bucket WebsiteConfiguration - AWS CloudFormation](#)

[CloudFormation template Parameters syntax - AWS CloudFormation](#)

[AWS::EC2::Instance - AWS CloudFormation](#)