Commit a Bugfix

**SPL-BE-200-DVCABF-1 - Version 1.0.5**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

This lab teaches you the fundamentals of using Git to manage code changes, which you learn by working on the Presidents application. You start by cloning the PresidentsApp AWS CodeCommit repository to the AWS Cloud9 instance that you use as a development environment. The Presidents application has a bug in the code that prevents the banner image from being displayed. You work on fixing the code to resolve this issue.

After you make the necessary code updates, you push your changes to the CodeCommit repo. The repo is connected to an AWS CodePipeline pipeline that’s called Presidents-Pipeline. CodePipeline is a managed continuous delivery service that’s designed to automate release pipelines for fast and reliable application and infrastructure updates. With CodePipeline, you can build, test, and deploy your code every time there’s a code change, based on the release process models that you define.

The repo’s connection to the Presidents-Pipeline invokes the pipeline for any new commit that’s made to the main branch. The process models have been set up to run unit tests, and if those tests pass, to deploy the updates by using AWS CodeDeploy. CodeDeploy is a fully managed deployment service that automates software deployments to various compute services, including Amazon Elastic Compute Cloud (Amazon EC2), AWS Fargate, AWS Lambda, and on-premises servers. For this lab, the compute environment is an Amazon EC2 Linux instance. You follow the pipeline stages until the application is updated and the banner image is showing as expected.

OBJECTIVES

By the end of this lab, you will be able to do the following:

* View the Presidents application for issues.
* Clone a CodeCommit repo to an AWS Cloud9 instance.
* Identify and fix a bug in the Presidents application.
* Push code updates to the PresidentsApp CodeCommit repo and observe the CodePipeline stages that update the EC2 instance running the Presidents application.
* View the updated Presidents application to ensure that the banner image displays correctly.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab:

* Familiarity with the basic navigation of the AWS Management Console.
* Versed in editing and running scripts by using an AWS Cloud9 code editor and terminal.
* A basic understanding of and familiarity with Amazon API Gateway, AWS Lambda, AWS CodePipeline, AWS CodeDeploy, and AWS CloudFormation.
* Prior experience with Git, AWS services, and serverless computing is helpful, but isn’t required.

DURATION

This lab requires *60* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Review the Presidents application**

In this task, you review the application to see if there are any issues.

1. From the navigation pane to the left of these instructions, copy the **WebsiteURL** value and in a new browser tab, open the website.

The website should show a banner image at the top of a US flag, with text that reads *Presidents App*. The website lists the names of several US presidents, and it also includes their dates of birth and death.

It appears that there’s an issue with the banner image because you see a broken image icon instead of the image itself. You address this next.

 Congratulations! You have tested the Presidents application by reviewing the frontend website and determined that the banner image for the Presidents application isn’t showing as expected.

**CHALLENGE A: Clone the Presidents CodeCommit repo and update the application code**

Before you clone the PresidentsApp repo and update the application in this challenge, review the following list of concepts that form the foundation of Git. These concepts are essential to using Git effectively:

* *Repository*: A repository (repo) is a collection of files, directories, and version history for a project.
* *Clone*: By cloning a Git repo, you create a local copy of a remote repository. When you clone a repo, you download the entire repo from a Git server to your local machine. A cloned repo means that you can work on the codebase locally, make changes, and push those changes back to the remote repo when you’re ready. Through cloning, multiple people can work simultaneously on the same code base by making their own local copies of the remote repo.
* *Commit*: A commit is a snapshot of changes that are made to the repo. Commits are identified by a unique ID and they contain a description of the changes that were made.
* *Branch*: A branch is a parallel version of the repo. You can use branches to make changes independently of the main branch. Branches are often used for feature development and bug fixes.
* *Merge*: Merging is the process of combining changes from one branch into another branch. A merge is often done when a feature or bug fix is complete and ready to be added to the main branch.
* *Pull*: A pull is the process of fetching changes from a remote repo and merging them into the local branch.
* *Push*: A push is the process of sending changes from the local branch to a remote repo.
* *Remote*: A remote is a connection to a repo that’s hosted on a server. Remotes can be used to share changes between developers and to store backups of the repo.

1. From the navigation pane to the left of these instructions, copy the **Cloud9Environment** URL value and in a new browser tab, open the AWS Cloud9 environment.
2. Close the **Welcome** tab and increase the size of the **terminal session** (at the bottom of the IDE) by expanding the section upwards.

**Note:** Git has already been installed on this AWS Cloud9 instance during the lab setup process.

To use Git, you first need to configure Git with your name and email address. The *git config --global user.name* and *git config --global user.email* commands are used to configure Git with the user’s name and email address. These settings identify the author of the changes that are made to the Git repo. When a user commits changes to a Git repo, Git records the author information and the commit message. By setting the *user.name* and *user.email* values, you help ensure that the correct author information is recorded for each commit, and that the information is consistent across all your Git repositories.

The --global flag in the command tells Git to set these values globally on your system, so they are used by all Git repos on your machine. If you don’t set these values, Git might prompt you for them each time you make a commit.

**Note:** This environment is a lab environment, and you don’t use your actual name and email address in this task.

1. **Command:** To configure a **generic name**, run the following command:

git config --global user.name "Student"

**Expected output:**

*None, unless there is an error.*

1. **Command:** To configure an **email address**, run the following command:

git config --global user.email "student@example.com"

**Expected output:**

*None, unless there is an error.*

You can authenticate to CodeCommit with an SSH key, generated git credentials, or with a signature created from AWS credentials. Your AWS Cloud9 instance is configured with the CodeCommit Git Remote Helper, which authenticates with a signature built from your AWS credentials when you interact with CodeCommit. After you authenticate, you can clone the repo by using the *git clone* command and the repo’s *URL*.

This authentication has already been configured with the lab build process. A role is attached to the AWS Cloud9 instance to grant this authentication.

1. **Command:** Clone the **PresidentsApp** repo with the following command:

git clone codecommit://PresidentsApp

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Cloning into 'PresidentsApp'...

remote: Counting objects: 29, done.

Unpacking objects: 100% (29/29), 24.27 KiB | 2.43 MiB/s, done.

Now that the application files are on your machine, you can troubleshoot the issues with application, fix the issue, and push the changes to the *PresidentsApp* repo.

1. Challenge yourself to see if you can identify the issue and solution on your own. If you get stuck, refer to the following hints.

* The code for displaying the image is in the *PresidentsApp/app/templates/main.html* file.
* The image is stored in the *PresidentsApp/app/static* folder.

**First Hint**

**Second Hint**

**Last Hint**

 Congratulations! You’ve successfully configured Git with authoring details by providing a name and email address. You’ve also identified the issue with the application code that prevented the image from displaying correctly. Finally, you’ve updated the code to complete the challenge.

**Task 2: Push the updated code changes to the repo and test the application**

Now that you updated the application code, you can push those changes to the PresidentsApp repo by using Git. Then, you observe how CodePipeline is invoked, how it performs a unit test on the code, and how it deploys the update to the EC2 instance by using CodeDeploy.

1. **Command:** Change directories to the **PresidentsApp** folder by running the following command:

cd PresidentsApp

**Expected output:**

AWSLabsUser:~/environment/PresidentsApp (main) $

The

git status

 command shows the current state of the local repo. It shows any changes that were made to the files in the repo since the last commit, such as files that were added, modified, or deleted. It also shows any files that are staged for commit or are untracked, and any branches or tags that might be checked out. This information can help users determine which files need to be committed, which files need to be added to the staging area, and which files need to be ignored.

1. **Command:** Check the status of files that Git is tracking for changes with the following command:

git status

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

On branch main

Your branch is up to date with 'origin/main'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: app/templates/main.html

no changes added to commit (use "git add" and/or "git commit -a")

This output shows that Git knows about the updates for the *app/templates/main.html* file, but that they have *not been staged for commit*.

**Note:** A visual cue for this status is the message in red text that reads modified: app/templates/main.html.

1. **Command:** Add the **main.html** file to the **staging area** by running the following command:

git add app/templates/main.html

**Expected output:**

*None, unless there is an error.*

1. **Command:** Check the **status** again with the following command:

git status

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

On branch main

Your branch is up to date with 'origin/main'.

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: app/templates/main.html

The message indicates that the changes to be committed include changes to the */app/templates/main.html* file.

**Note:** A visual cue for this status is the “modified” message, which is now in green text: modified: app/templates/main.html.

By adding a commit message, you help identify the changes that were made to the file. Other people can also read the commit message to understand the update you made, and why you made it.

1. **Command:** Commit the changes to the /app/templates/main.html file with the following command:

git commit -m "Updated main.html for a typo in the banner.png file name."

**Expected output:**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*

**\*\*\*\*** This is OUTPUT ONLY. **\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*

[main 7d05e89] Updated main.html for a typo in the banner.png image name.

1 file changed, 3 insertions(+), 1 deletion(-)

**Note:**

* It’s important to understand that when you make a commit, the *-m* stands for *message*. The value of the commit message needs to be inside of *opening* and *closing* quotation marks (“”). If the message isn’t in this format, an error can occur.
* If you include the opening quotation mark but forget to include the closing quotation mark, Git will add another line to the commit message each time you press Enter. Git is programmed to interpret this action as though you are entering a *multiline* commit message. It waits for the closing quotation mark before it adds the commit message.

1. **Command:** To see what Git shows for the status of your local repo, run the following **git status** command again:

git status

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

This message indicates that all changes are accounted for, and you can now push your local changes to the remote branch. Recall that when you push changes, the CodePipeline pipeline is invoked. After the pipeline is invoked, it will start running unit tests, and then deploy the new code changes by using CodeDeploy

1. **Command:** Push the changes to the remote branch and invoke the pipeline by running the **git push** command:

git push

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enumerating objects: 9, done.

Counting objects: 100% (9/9), done.

Delta compression using up to 2 threads

Compressing objects: 100% (4/4), done.

Writing objects: 100% (5/5), 482 bytes | 482.00 KiB/s, done.

Total 5 (delta 3), reused 0 (delta 0), pack-reused 0

remote: Validating objects: 100%

To codecommit://PresidentsApp

f69890a..7d05e89 main -> main

The output shows that the changes were pushed to the PresidentsApp CodeCommit repo.

**Note:** Observe the text on the last line of the push command output before *main -> main*. The *f69890a…7d05e89* text string in this example, which differs from yours, is known as your *commit ID*.

1. At the top of the browser tab opened to the AWS Management Console, in the search bar, search for and choose

CodePipeline

.

1. Choose the pipeline link with the name **Presidents-Pipeline**.
2. Review the first stage (the **Source** stage) and look at its details.

Notice that the **commit ID** is listed like it was listed in the terminal output. The details also contain the **commit message**. This stage shows as **Succeeded**.

1. Move to the **Test** stage and review its details.

The status for the Test stage might already show as **Succeeded**. If not, it could take a few moments. Notice that the **commit ID** and **commit message** are also displayed. This information helps identify that the change is the latest commit that you ran. Wait for the pipeline to complete before continuing.

1. Move to the **Deploy** stage and review its details.

Similar to the previous stages, the Deploy stage shows the **commit ID** and the **commit message**.

1. In the **Deploy** stage, choose to open the **Details** in a new browser tab and review the following information:

* The deployment status shows as *Succeeded*.
* The *Application* value is set to *presidents-app*.
* A *Deployment ID* is listed, which you can use to reference for this particular deployment. Each deployment has its own **Deployment ID**.
* The *Deployment group* is used as a container for all the instances that updates will be deployed to. In this case, the deployment group includes only one EC2 instance.
* In the **Deployment lifecycle events section**, you can see the EC2 instance ID and a link to view the *Events*. This link shows you all the actions that were taken during the CodeDeploy deployment.

1. Switch back to the browser tab that’s opened to the **Presidents application**.
2. Refresh the page.

The Presidents App image that wasn’t displaying previously is now visible.

 Congratulations! You have updated the application code and pushed the changes to the PresidentsApp CodeCommit repo. The new files that were added to the repo invoked the presidents-app CodePipeline pipeline. The pipeline ran unit tests and then issued a CodeDeploy deployment to update the application, which is running on an EC2 instance.

**Conclusion**

 Congratulations! You now have successfully:

* Viewed the Presidents application for issues.
* Cloned a CodeCommit repo to an AWS Cloud9 instance.
* Identified and fixed a bug in the Presidents application.
* Pushed code updates to the PresidentsApp CodeCommit repo and observed the CodePipeline stages as they updated the EC2 instance that hosts the Presidents application.
* Viewed the updated Presidents application to confirm that the banner image displays correctly.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).