aws

Hands-on tutorials

Create Continuous Delivery Pipeline

Create Continuous Delivery Pipeline: Hands-on tutorials

Table of Contents

Create Continuous Delivery Pipeline

AWS experience	Beginner
Time to complete	35 minutes
Cost to complete	Free Tier eligible
Services used	AWS Elastic Beanstalk AWS CodeBuild AWS CodePipeline
Last update	August 18, 2023

Overview

In this tutorial, you will create a continuous delivery pipeline for a simple web application. You will first use a version control system to store your source code. Then, you will learn how to create a continuous delivery pipeline that will automatically deploy your web application whenever your source code is updated.

What you will accomplish

This tutorial will walk you through the steps to create the continuous delivery pipeline discussed above. You will learn to:

- Set up a GitHub repository for the application code
- Create an AWS Elastic Beanstalk environment to deploy the application
- Configure AWS CodeBuild to build the source code from GitHub
- Use AWS CodePipeline to set up the continuous delivery pipeline with source, build, and deploy stages

Prerequisites

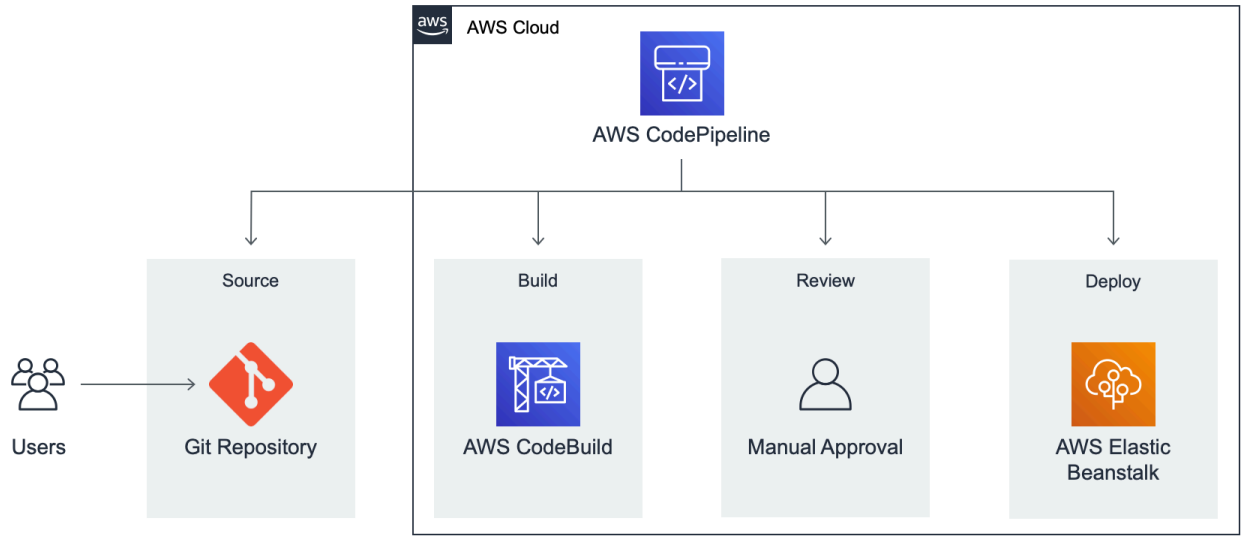
Before starting this tutorial, you will need:

- An AWS account
- A GitHub account
- Git installed on your computer

Application architecture

The following diagram provides a visual representation of the services used in this tutorial and how they are connected. This application uses GitHub, AWS Elastic Beanstalk, AWS CodeBuild, and AWS CodePipeline.

As we go through the tutorial, we will discuss the services in detail and point to resources that will help you get up to speed with them.



Modules

This tutorial is divided into five short modules. You must complete each module in order before moving on to the next one.

1. Module 1: Set Up Git Repo (5 minutes): Set up a GitHub repository to store the application code.
2. Module 2: Deploy Web App (10 minutes): Create the environment where the web application will be deployed using AWS Elastic Beanstalk.
3. Module 3: Create Build Project (5 minutes): Configure and start the build process for the application using AWS CodeBuild.
4. Module 4: Create Delivery Pipeline (10 minutes): Create a pipeline to automatically build and deploy the application using AWS CodePipeline.
5. Module 5: Finalize Pipeline and Test (5 minutes): Add a review stage to the pipeline and test the pipeline.

Module 1: Set Up Git Repo

Time to complete	5 minutes

Requires	• GitHub account
	• Git installed on your computer
	• A text editor. Here are a few options (in alphabetical order):
	  • Atom
	  • Notepad++
	  • Sublime
	  • Vim
	  • Visual Studio Code

Overview

In this module, you will set up a repository for your code so it can be easily accessed over the Internet. In this example, we will be using GitHub, but there are other Git-compatible options you can use, including AWS CodeCommit. In one of the following modules you will connect this hosted repo to our pipeline, so every time you push a new commit to it your build process is started.

What you will accomplish

- Fork a GitHub repository to create a new one
- Store code and metadata in GitHub
- Interact with a code repository using Git

Key concepts

Version control—A system for storing source code and tracking any changes to it. Changes are stored as versions, so a developer can easily compare versions or choose to revert to an older version.

Git—An open-source version control tool for managing changes to source code.

Git repository (repo)—All the files and directories that will be tracked by the version control system, including metadata. Each user will interact with a complete copy locally and push files to the hosted version to share changes.

Git commit—The method to add changes to a Git repository.

Pushing to a repository—Copying any changes stored via a commit from a local repository to a hosted one.

Forking a repository—Creating a copy of an existing repository.

Implementation

Step 1: Fork the starter repo

This tutorial assumes you have an existing GitHub account and Git installed on your computer. If you don't have either of these two installed, you can follow these step-by-step instructions.

1. Sign in to GitHub

   In a new browser tab, navigate to GitHub and make sure you are logged into your account.

2. Open the repo

   In that same tab, open the aws-elastic-beanstalk-express-js-sample repo.

3. Fork the repo

   Choose the white **Fork** button on the top right corner of the screen. Next, you will see a small window asking you where you would like to fork the repo.

4. Create the fork

   Verify it is showing your account and choose **Create a fork**. After a few seconds, your browser will display a copy of the repo in your account under **Repositories**.

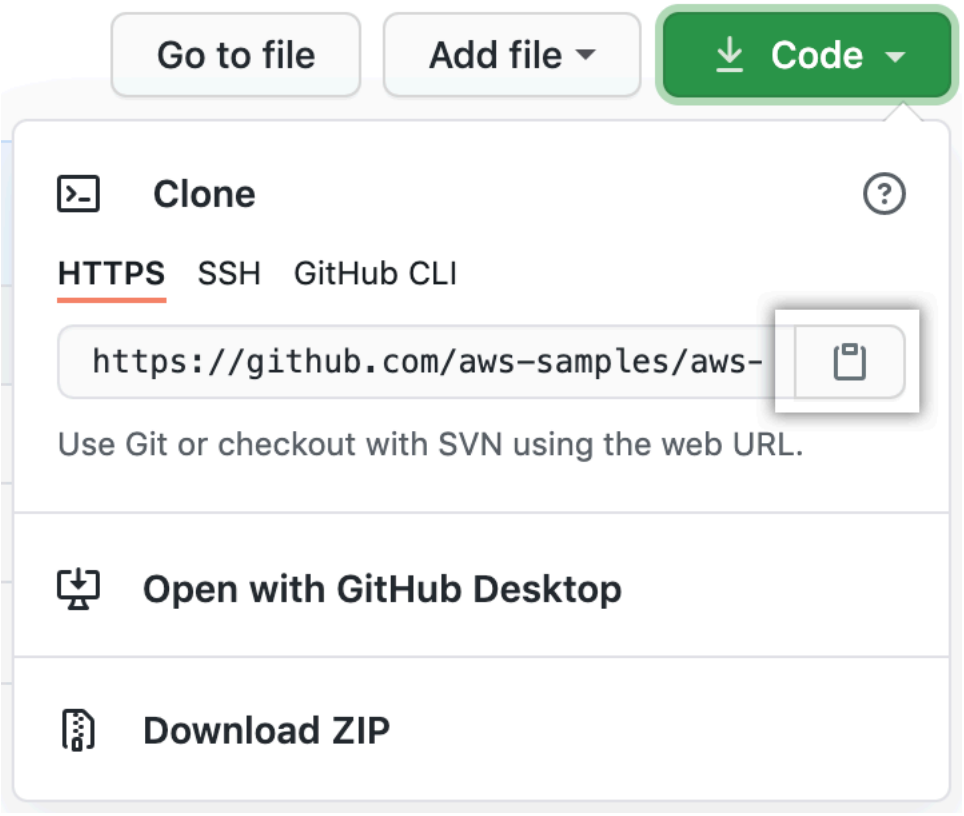
Step 2: Push a change to your new repo

1. Copy the repo URL

Go to the repository and choose the green **Code** button near the top of the page.

To clone the repository using HTTPS, confirm that the heading says **Clone with HTTPS.** If not, select the **Use HTTPS** link.

Choose the white button with a clipboard icon on it (to the right of the URL).



2. Open a terminal

   If you're on a Mac or Linux computer, open your terminal. If you're on Windows, launch Git Bash.

3. Clone the repo

   In the terminal or Bash platform, whichever you are using, enter the following command and paste the URL you just copied when you clicked the clipboard icon. Be sure to change "YOUR-USERNAME" to your GitHub username. You should see a message in your terminal that starts with **Cloning into.** This command creates a new folder that has a copy of the files from the GitHub repo.

   ```
   git clone https://github.com/YOUR-USERNAME/aws-elastic-beanstalk-express-js-sample
   ```

4. Modify the app.js file

   In the new folder there is a file named **app.js.** Open **app.js** in your favorite code editor.

   Change the message in line 5 to say something other than "Hello World!" and save the file.

5. Commit and push the change

   Go to the folder created with the name **aws-elastic-beanstalk-express-js-sample/** and **Commit** the change with the following commands:

   ```
   git add app.js
   git commit -m "change message"
   ```

   Push the local changes to the remote repo hosted on GitHub with the following command. Note that you need to configure **Personal access tokens (classic)** under **Developer Settings** in GitHub for remote authentication.

   ```
   git push
   ```

Step 3: Test your changes

- Verify the repo updated

  In your browser window, open GitHub.

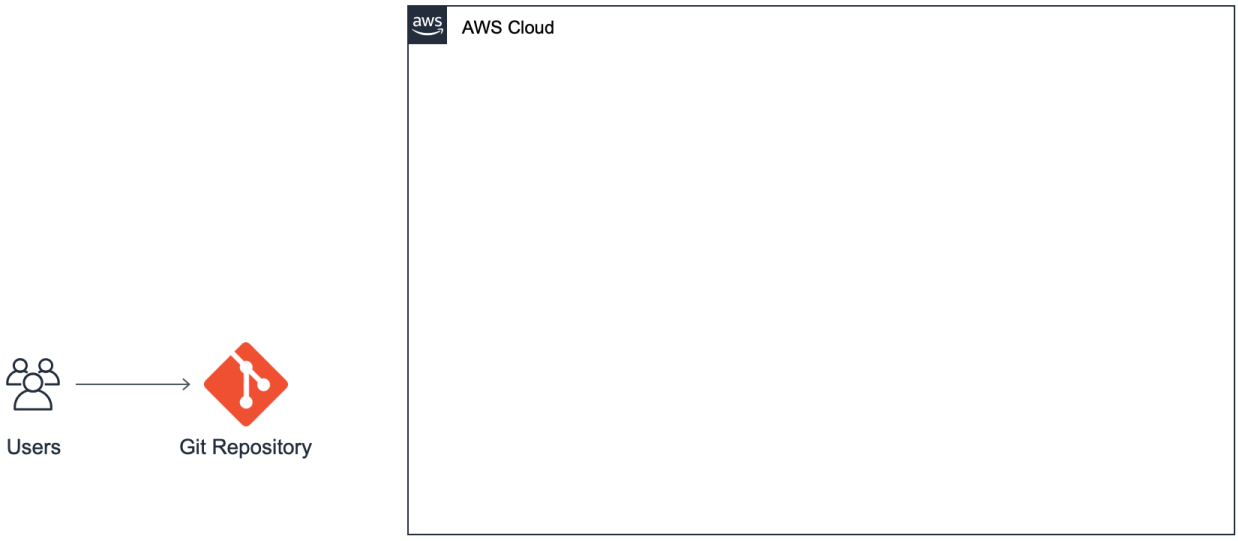
  In the left navigation panel, under **Repositories,** select **aws-elastic-beanstalk-express-js-sample**.

  Choose the **app.js** file. The contents of the file, including your change, should be displayed.

Application architecture

Here is what our architecture looks like right now.

We have created a code repository containing a simple web app. We will be using this repository to start our continuous delivery pipeline. It's important to set it up properly so we push code to it.


AWS Cloud
Users
Git Repository

Module 2: Deploy Web App

Time to complete	10 minutes
Services used	AWS Elastic Beanstalk

Overview

In this module, you will use the AWS Elastic Beanstalk console to create and deploy a web application. AWS Elastic Beanstalk is a compute service that makes it easy to deploy and manage applications on AWS without having to worry about the infrastructure that runs them. You will use the **Create web app** wizard to create an application and launch an environment with the AWS resources needed to run your application. In subsequent modules, you will be using this environment and your continuous delivery pipeline to deploy the Hello World! web app created in Module 1.

What you will accomplish

In this module, you will:

- Configure and create an AWS Elastic Beanstalk environment
- Deploy a sample web app to AWS Elastic Beanstalk
- Test the sample web app

Key concepts

AWS Elastic Beanstalk—A service that makes it easy to deploy your application on AWS. You simply upload your code and Elastic Beanstalk deploys, manages, and scales your application.

Environment—Collection of resources provisioned by Elastic Beanstalk that are used to run your application.

EC2 instance—Virtual server in the cloud. Elastic Beanstalk will provision one or more Amazon EC2 instances when creating an environment.

Web server—Software that uses the HTTP protocol to serve content over the Internet. It is used to store, process, and deliver web pages.

Platform—Combination of operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. Your application runs using the components provided by a platform.

Implementation

Step 1: Configure an AWS Elastic Beanstalk app

1. Configure the app

   In a new browser tab, open the AWS Elastic Beanstalk console.

   Choose **Create Application**.

   Choose **Web server environment** under the Configure environment heading.

   In the text box under the heading Application name, enter **DevOpsGettingStarted.**

   In the **Platform** dropdown menu, under the Platform heading, select **Node.js** . **Platform branch** and **Platform version** will automatically populate with default selections.

   Confirm that the radio button next to **Sample application** under the Application code headingis selected.

   Confirm that the radio button next to **Single instance (free tier eligible)** under the Presets heading is selected.

   Select **Next.**

Configure environment Info

Environment tier Info

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

- (•) Web server environment
  Run a website, web application, or web API that serves HTTP requests. Learn more
- ( ) Worker environment
  Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. Learn more

Application information Info

Application name

DevOpsGettingStarted

Maximum length of 100 characters.

▶ Application tags (optional)

Environment information Info

Choose the name, subdomain and description for your environment. These cannot be changed later.

Environment name

DevOpsGettingStarted-env

Must be from 4 to 40 characters in length. The name can contain only letters, numbers, and hyphens. It can't start or end with a hyphen. This name must be unique within a region in your account.

Domain

Leave blank for autogenerated value .us-east-1.elasticbeanstalk.com Check availability

Environment description

Platform Info

Platform type

- (•) Managed platform
  Platforms published and maintained by Amazon Elastic Beanstalk. Learn more
- ( ) Custom platform
  Platforms created and owned by you. This option is unavailable if you have no platforms.

2. Configure service access

   On the Configure service access screen, choose **Use an existing service role** for Service Role.

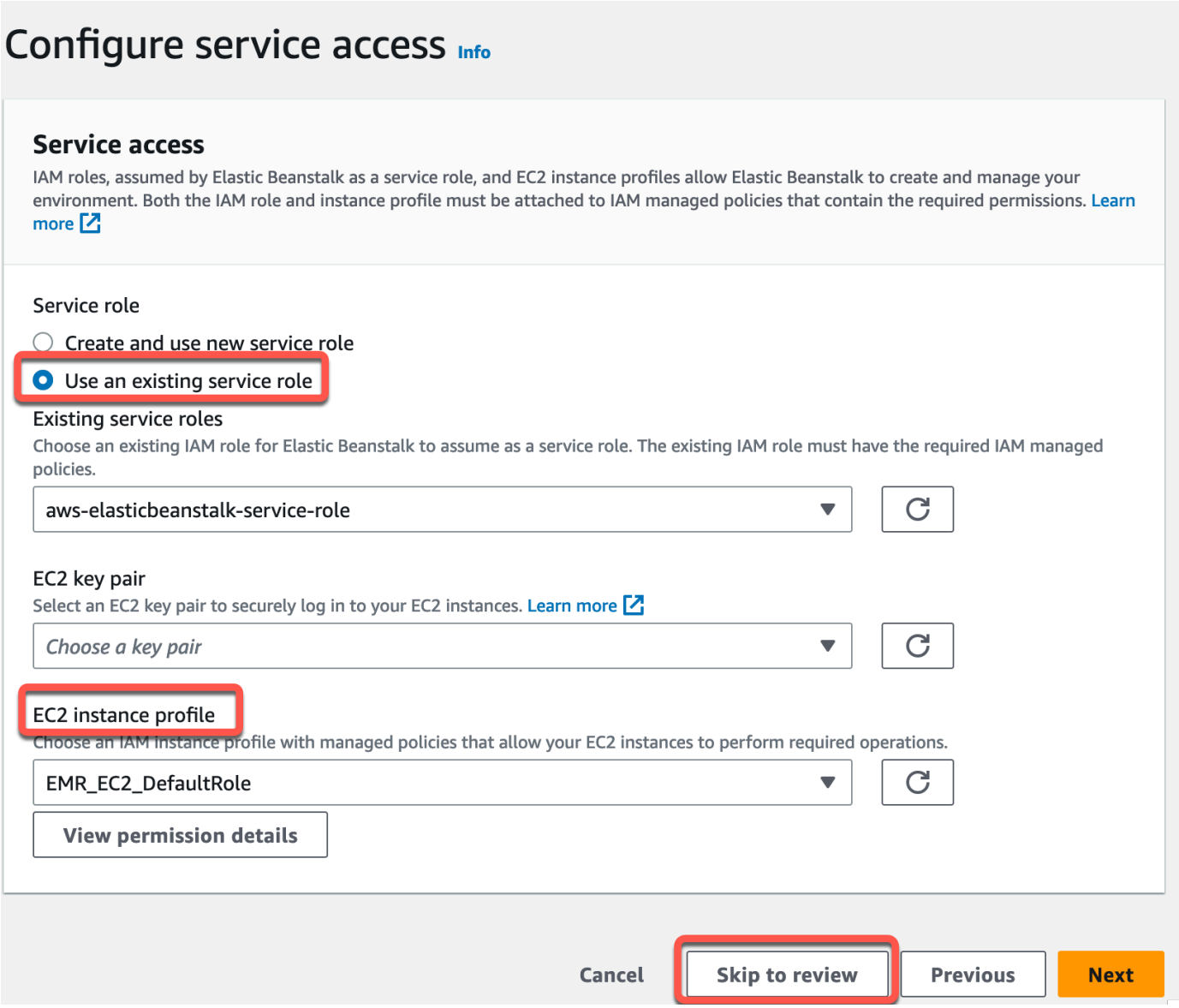
   For **EC2 instance profile** dropdown list, the values displayed in this dropdown list may vary, depending on whether you account has previously created a new environment.

   Choose one of the following, based on the values displayed in your list.

   - If **aws-elasticbeanstalk-ec2-role** displays in the dropdown list, select it from the EC2 instance profile dropdown list.
   - If another value displays in the list, and it's the **default EC2 instance profile** intended for your environments, select it from the EC2 instance profile dropdown list.
   - If the EC2 instance profile dropdown list doesn't list any values to choose from, expand the procedure that follows, **Create IAM Role for EC2 instance profile**.
     - Complete the steps in Create IAM Role for EC2 instance profile to create an IAM Role that you can subsequently select for the EC2 instance profile. Then, return back to this step.
     - Now that you've created an IAM Role, and refreshed the list, it displays as a choice in the dropdown list. Select the **IAM Role** you just created from the EC2 instance profile dropdown list.

   Choose **Skip to Review** on the Configure service access page.

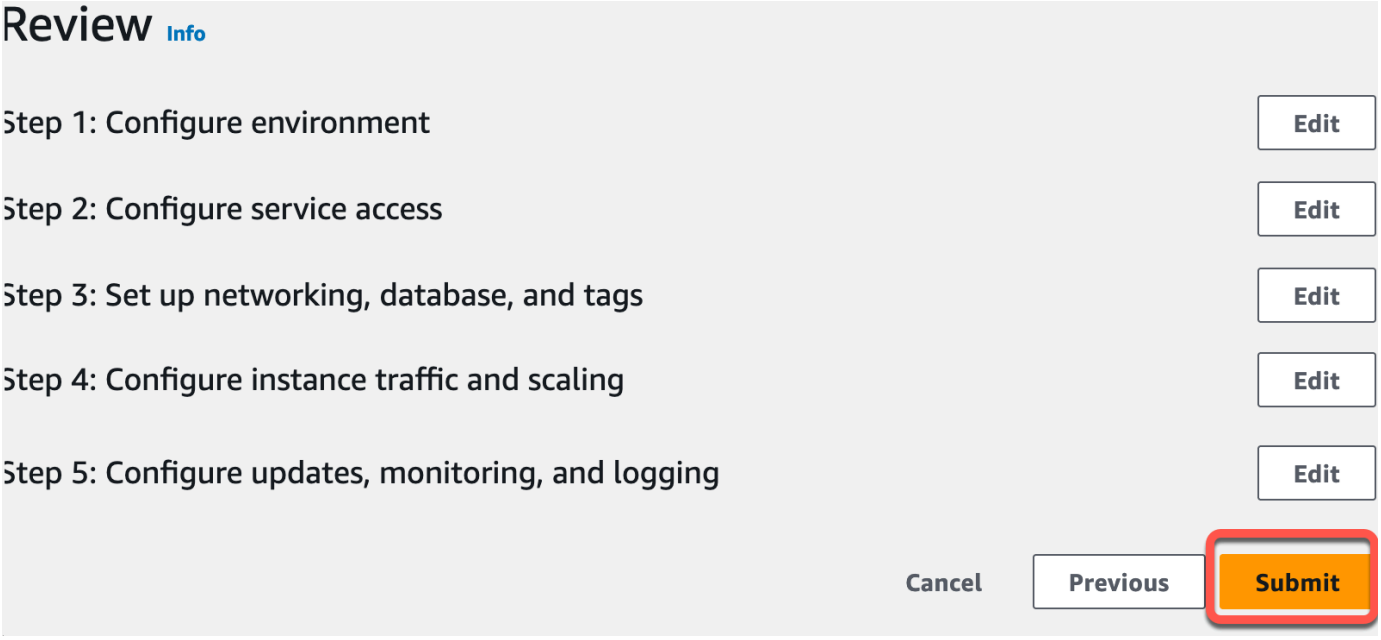
   This will select the default values for this step and skip the optional steps.



3. Create the app

   The Review page displays a summary of all your choices.

   Choose **Submit** at the bottom of the page to initialize the creation of your new environment.

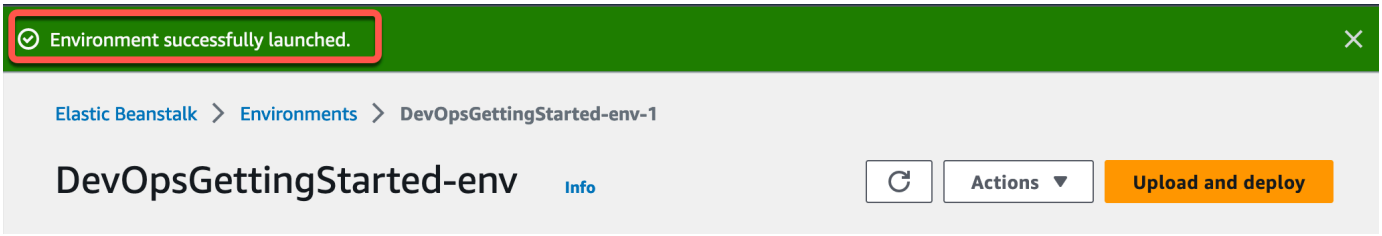


4. Verify successful creation

   While waiting for deployment, you should see:

   - A screen that will display status messages for your environment.
   - After a few minutes have passed, you will see a green banner with a checkmark at the top of the environment screen.

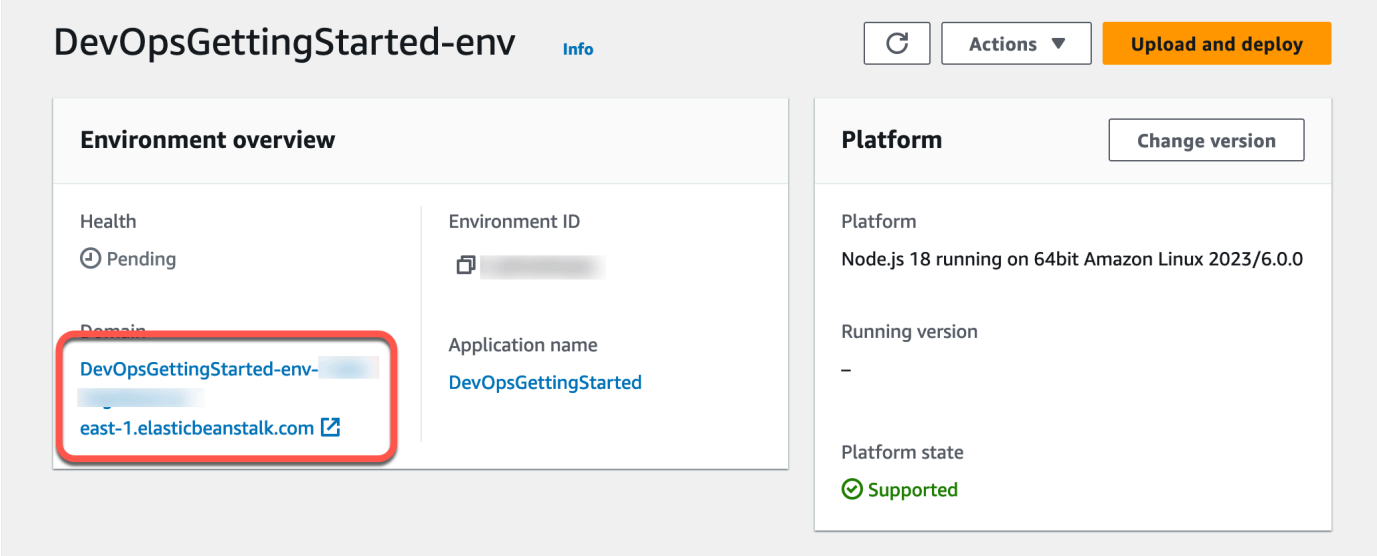
   Once you see the banner, you have successfully created an AWS Elastic Beanstalk application and deployed it to an environment.



Step 2: Test your web app

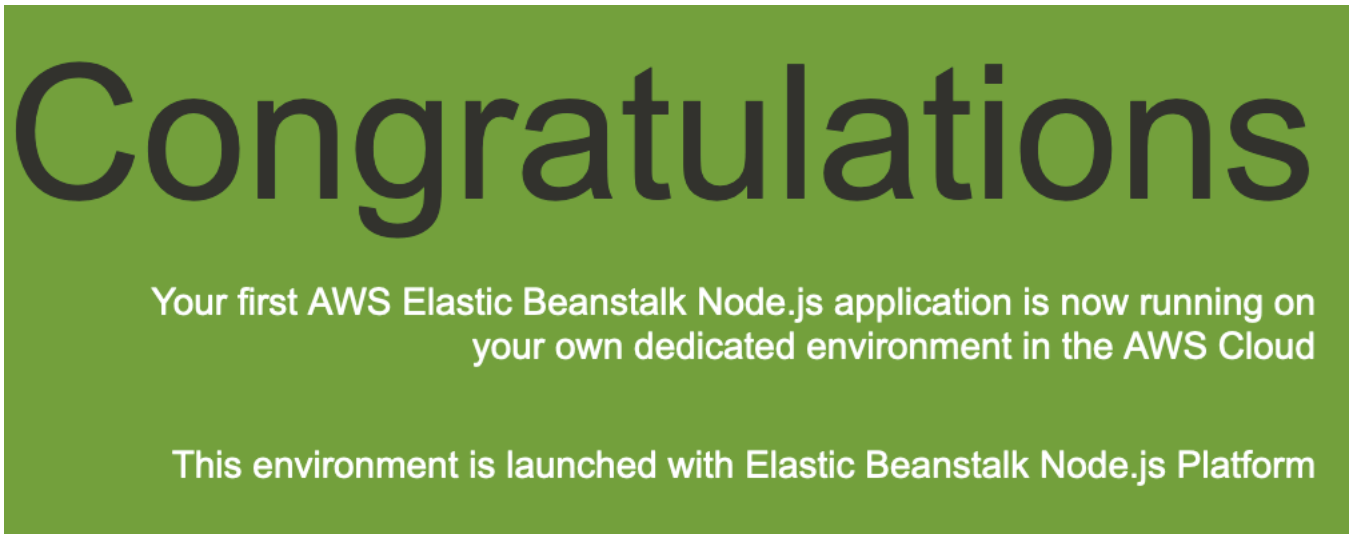
1. Select your app

   To test your sample web app, select the link under the name of your environment.



2. View the webpage

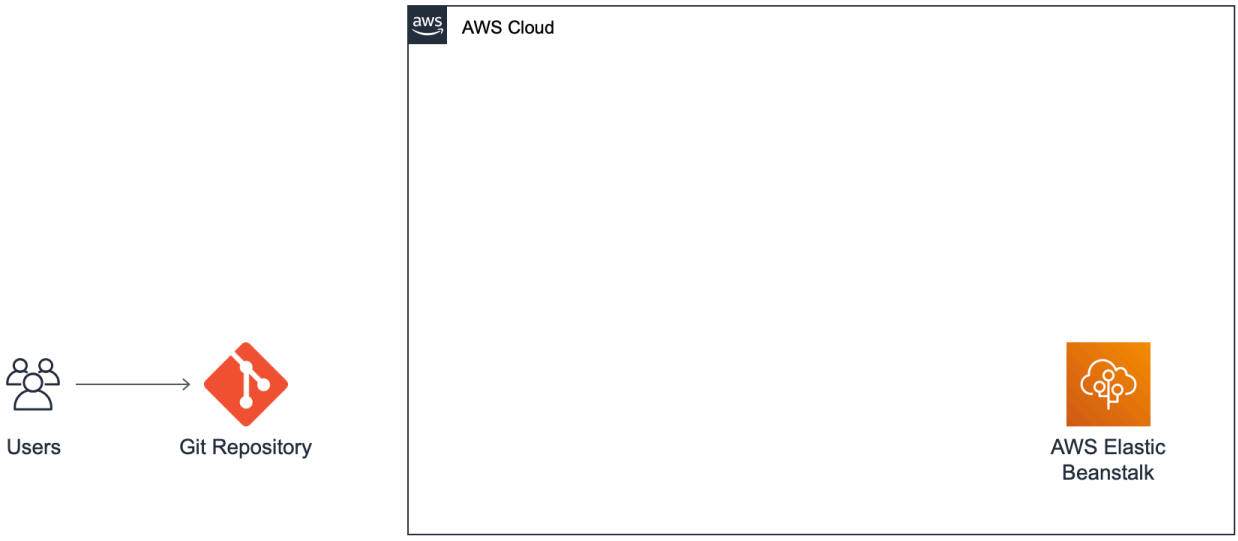
   Once the test has completed, a new browser tab should open with a webpage congratulating you!



Application architecture

Now that we are done with this module, our architecture will look like this:

We have created an AWS Elastic Beanstalk environment and sample application. We will be using this environment and our continuous delivery pipeline to deploy the Hello World! web app we created in the previous module.


AWS Cloud
Users
Git Repository
AWS Elastic
Beanstalk

Module 3: Create Build Project

Time to complete	5 minutes
Services used	AWS CodeBuild

Overview

In this module, you will use AWS CodeBuild to build the source code previously stored in your GitHub repository. AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy.

What you will accomplish

In this module, you will:

- Create a build project with AWS CodeBuild
- Set up GitHub as the source provider for a build project
- Run a build on AWS CodeBuild

Key concepts

Build process—Process that converts source code files into an executable software artifact. It may include the following steps: compiling source code, running tests, and packaging software for deployment.

Continuous integration—Software development practice of regularly pushing changes to a hosted repository, after which automated builds and tests are run.

Build environment—Represents a combination of the operating system, programming language runtime, and tools that CodeBuild uses to run a build.

Buildspec—Collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build.

Build Project—Includes information about how to run a build, including where to get the source code, which build environment to use, which build commands to run, and where to store the build output.

OAuth—Open protocol for secure authorization. OAuth enables you to connect your GitHub account to third-party applications, including AWS CodeBuild.

Implementation

Step 1: Configure the AWS CodeBuild project

1. Create a project

   In a new browser tab, open the AWS CodeBuild console.

   Choose the orange **Create project** button.

2. Configure the project

   In the **Project name** field, enter**Build-DevOpsGettingStarted.**

   Select **GitHub** from the **Source provider** dropdown menu.

   Confirm that the **Connect using OAuth** radio button is selected.

   Choose the white **Connect to GitHub** button. A new browser tab will open asking you to give AWS CodeBuild access to your GitHub repo.

   Choose the green **Authorize aws-codesuite** button.

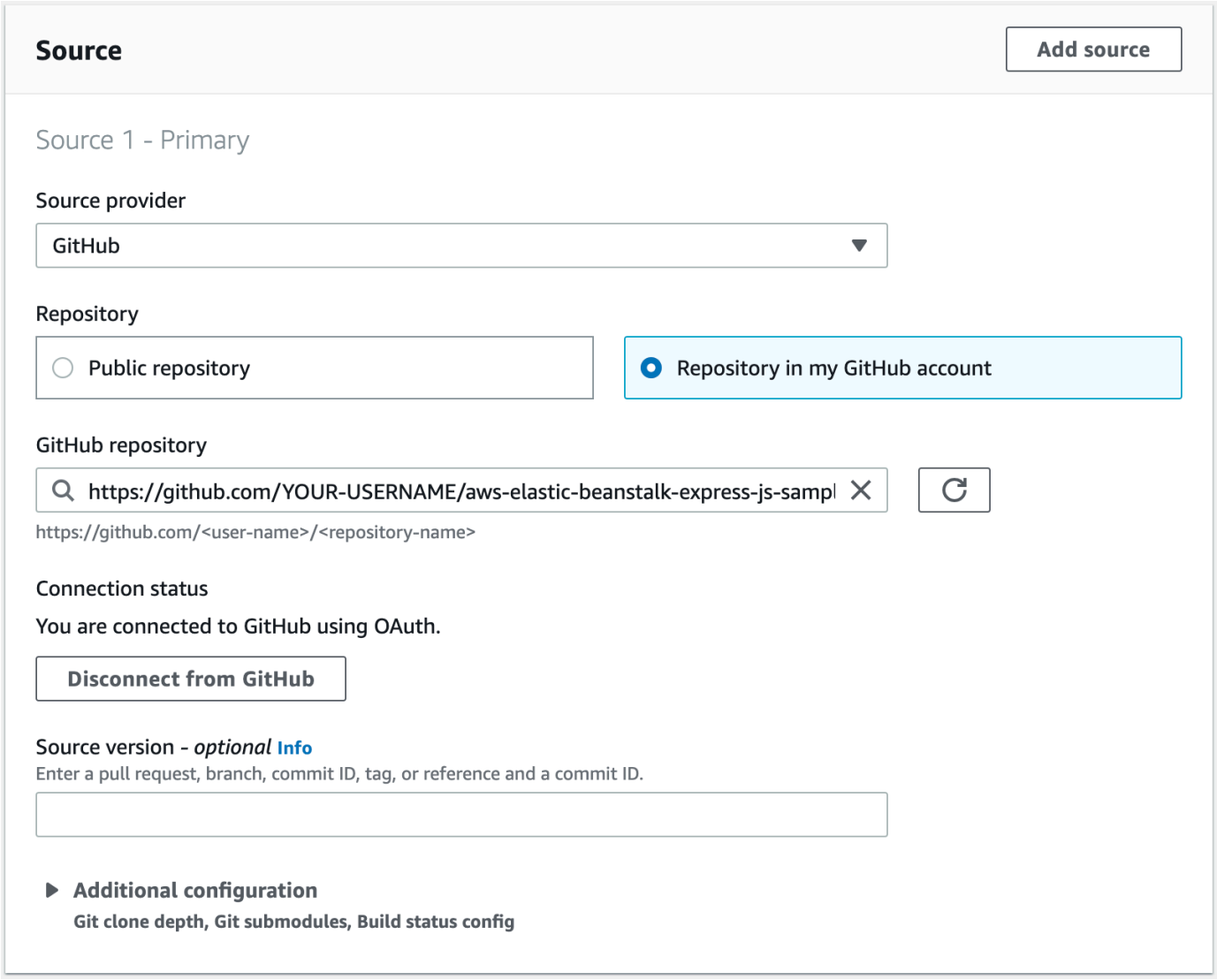
   Enter your GitHub password.

   Choose the orange **Confirm** button.

   Select **Repository in my GitHub account.**

   Enter **aws-elastic-beanstalk-express-js-sample** in the search field.

   Select the repo you forked in Module 1. After selecting your repo, your screen should look like the screenshot.



3. Continue configuring the project

   Confirm that **Managed Image** is selected.

   Select **Amazon Linux 2** from the **Operating system** dropdown menu.

   Select **Standard** from the **Runtime(s)** dropdown menu.

   Select **aws/codebuild/amazonlinux2-x86_64-standard:3.0** from the **Image** dropdown menu.

   Confirm that **Always use the latest image for this runtime version** is selected for **Image version.**

   Confirm that **Linux** is selected for **Environment type.**

Confirm that **New service role** is selected.

Step 2: Create a Buildspec file for the project

1. Open the build command editor

   Select **Insert build commands.**

   Choose **Switch to editor.**

2. Update the Buildspec file

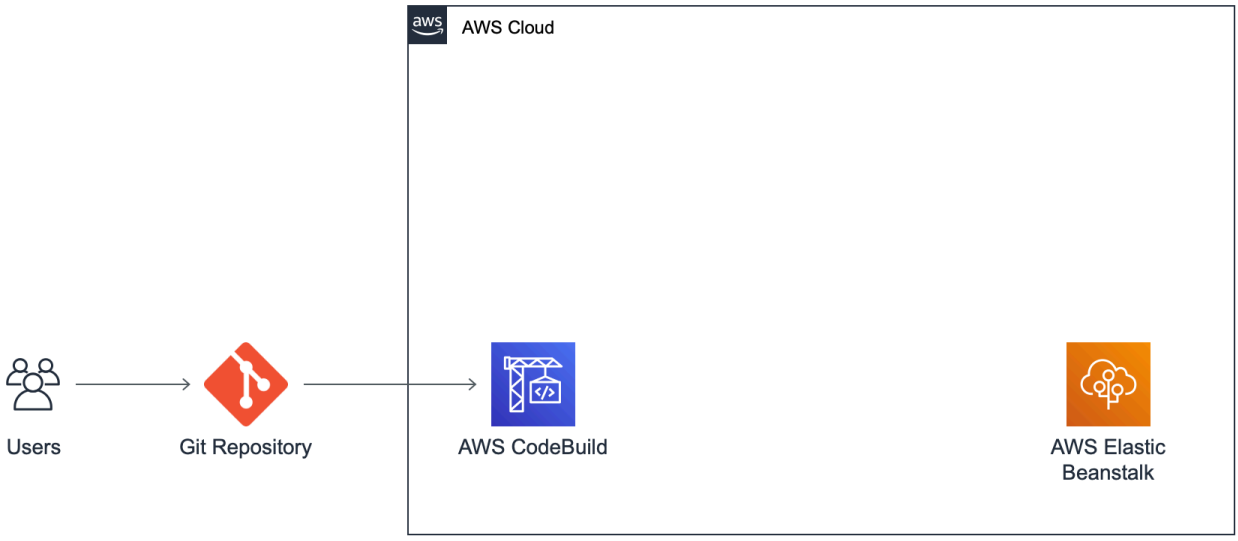
   Replace the Buildspec in the editor with the code below:

```
version: 0.2
phases:
    build:
        commands:
            - npm i --save
artifacts:
    files:
        - '**/*'
```

Application architecture

Here's what our architecture looks like now.

We have created a build project on AWS CodeBuild to run the build process of the Hello World! web app from our GitHub repository. We will be using this build project as the build step in our continuous delivery pipeline, which we will create in the next module.


AWS Cloud
Users
Git Repository
AWS CodeBuild
AWS Elastic Beanstalk

Module 4: Create Delivery Pipeline

Time to complete	10 minutes
Services used	AWS CodePipeline

Overview

In this module, you will use AWS CodePipeline to set up a continuous delivery pipeline with source, build, and deploy stages. The pipeline will detect changes in the code stored in your GitHub repository, build the source code using AWS CodeBuild, and then deploy your application to AWS Elastic Beanstalk.

What you will accomplish

In this module, you will:

- Set up a continuous delivery pipeline on AWS CodePipeline
- Configure a source stage using your GitHub repo
- Configure a build stage using AWS CodeBuild
- Configure a deploy stage using your AWS Elastic Beanstalk application
- Deploy the application hosted on GitHub to Elastic Beanstalk through a pipeline

Key concepts

Continuous delivery—Software development practice that allows developers to release software more quickly by automating the build, test, and deploy processes.

Pipeline—Workflow model that describes how software changes go through the release process. Each pipeline is made up of a series of stages.

Stage—Logical division of a pipeline, where actions are performed. A stage might be a build stage, where the source code is built and tests are run. It can also be a deployment stage, where code is deployed to runtime environments.

Action—Set of tasks performed in a stage of the pipeline. For example, a source action can start a pipeline when source code is updated, and a deploy action can deploy code to a compute service like AWS Elastic Beanstalk.

Implementation

Step 1: Create a new pipeline

1. In a browser window, open the AWS CodePipeline console.
2. Choose the orange **Create pipeline** button. A new screen will open up so you can set up the pipeline.
3. In the **Pipeline name** field, enter **Pipeline-DevOpsGettingStarted.**
4. Confirm that **New service role** is selected.
5. Choose the orange **Next** button.

Step 2: Configure the source stage

1. Select **GitHub version 1** from the **Source provider** dropdown menu.
2. Choose the white **Connect to GitHub** button. A new browser tab will open asking you to give AWS CodePipeline access to your GitHub repo.
3. Choose the green **Authorize aws-codesuite** button. Next, you will see a green box with the message **You have successfully configured the action with the provider.**
4. From the **Repository** dropdown, select the repo you created in Module 1.
5. Select **main** from the **branch** dropdown menu.
6. Confirm that **GitHub webhooks** is selected.
7. Choose the orange **Next** button.

Step 3: Configure the build stage

1. From the **Build provider** dropdown menu, select **AWS CodeBuild.**
2. Under **Region** confirm that the **US West (Oregon)** Region is selected.
3. Select **Build-DevOpsGettingStarted** under **Project name.**
4. Choose the orange **Next** button.

Step 4: Configure the deploy stage

1. Select **AWS Elastic Beanstalk** from the **Deploy provider** dropdown menu.
2. Under **Region,** confirm that the **US West (Oregon)** Region is selected.
3. Select the field under **Application name** and confirm you can see the app **DevOpsGettingStarted** created in Module 2.
4. Select **DevOpsGettingStarted-env** from the **Environment name** textbox.
5. Choose the orange **Next** button. You will now see a page where you can review the pipeline configuration.
6. Choose the orange **Create pipeline** button.

Step 5: Watch first pipeline execution

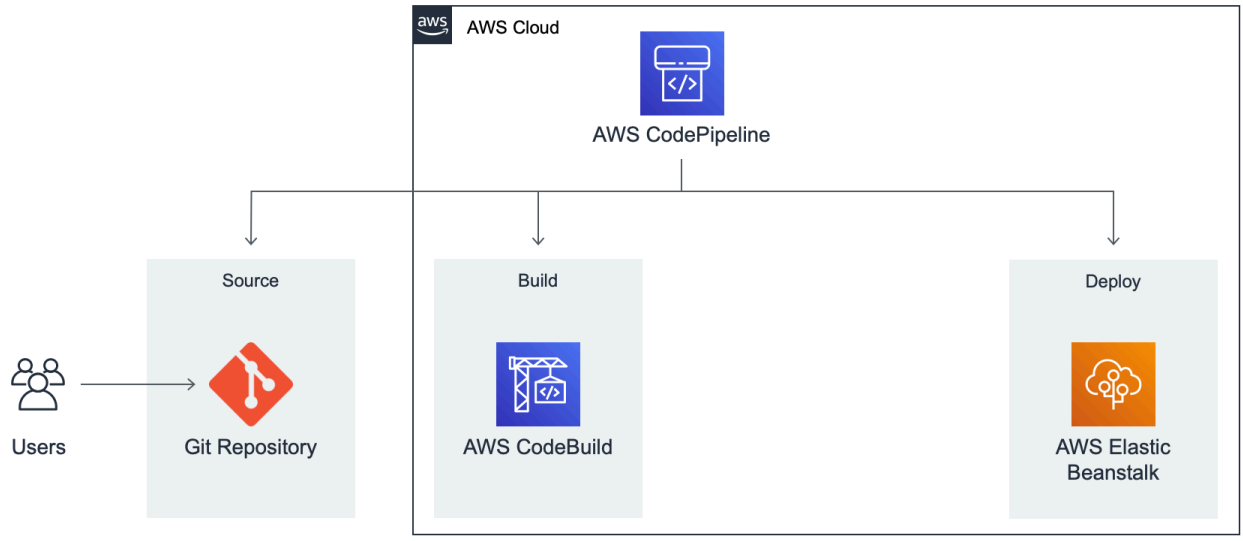
While watching the pipeline execution, you will see a page with a green bar at the top. This page shows all the steps defined for the pipeline and, after a few minutes, each will change from blue to green.

1. Once the **Deploy** stage has switched to green and it says **Succeeded,** choose **AWS Elastic Beanstalk.** A new tab listing your AWS Elastic Beanstalk environments will open.
2. Select the URL in the **Devopsgettingstarted-env** row. You should see a webpage with a white background and the text you included in your GitHub commit in Module 1.

Application architecture

Here's what our architecture looks like now:

We have created a continuous delivery pipeline on AWS CodePipeline with three stages: source, build, and deploy. The source code from the GitHub repo created in Module 1: Set Up Git Repo is part of the source stage. That source code is then built by AWS CodeBuild in the build stage. Finally, the built code is deployed to the AWS Elastic Beanstalk environment created in Module 3: Create Build Project.


AWS Cloud
AWS CodePipeline
Source
Build
Deploy
Users
Git Repository
AWS CodeBuild
AWS Elastic Beanstalk

Module 5: Finalize Pipeline and Test

AWS experience	Beginner
Time to complete	5 minutes
Services used	AWS CodePipeline

Overview

In this module, you will use AWS CodePipeline to add a review stage to your continuous delivery pipeline.

As part of this process, you can add an approval action to a stage at the point where you want the pipeline execution to stop so someone can manually approve or reject the action. Manual approvals are useful to have someone else review a change before deployment. If the action is approved, the pipeline execution resumes. If the action is rejected—or if no one approves or rejects the action within seven days—the result is the same as the action failing, and the pipeline execution does not continue.

What you will accomplish

In this module, you will:

- Add a review stage to your pipeline
- Manually approve a change before it is deployed

Key concepts

Approval action—Type of pipeline action that stops the pipeline execution until someone approves or rejects it.

Pipeline execution—Set of changes, such as a merged commit, released by a pipeline. Pipeline executions traverse the pipeline stages in order. Each pipeline stage can only process one execution at a time. To do this, a stage is locked while it processes an execution.

Failed execution—If an execution fails, it stops and does not completely traverse the pipeline. The pipeline status changes to **Failed** and the stage that was processing the execution is unlocked. A failed execution can be retried or replaced by a more recent execution.

Implementation

Step 1: Create review stage in pipeline

1. Open the AWS CodePipeline console.
2. You should see the pipeline we created in Module 4, which was called **Pipeline-DevOpsGettingStarted.** Select this pipeline.
3. Choose the white **Edit** button near the top of the page.
4. Choose the white **Add stage** button between the **Build** and **Deploy** stages.
5. In the **Stage name** field, enter **Review.**
6. Choose the orange **Add stage** button.
7. In the **Review** stage, choose the white **Add action group** button.
8. Under **Action name,** enter **Manual_Review.**
9. From the **Action provider** dropdown, select **Manual approval.**
10. Confirm that the optional fields have been left blank.
11. Choose the orange **Done** button.
12. Choose the orange **Save** button at the top of the page.
13. Choose the orange **Save** button to confirm the changes. You will now see your pipeline with four stages: **Source, Build, Review,** and **Deploy.**

Step 2: Push a new commit to your repo

1. In your favorite code editor, open the **app.js** file from Module 1.
2. Change the message in Line 5.
3. Save the file.
4. Open your preferred Git client.
5. Navigate to the folder created in Module 1.
6. Commit the change with the following commands:

```
git add app.js
git commit -m "Full pipeline test"
```

7. Push the local changes to the remote repo hosted on GitHub with the following command:

```
git push
```

Step 3: Monitor the pipeline and manually approve the change

1. Navigate to the AWS CodePipeline console.
2. Select the pipeline named **Pipeline-DevOpsGettingStarted.** You should see the **Source** and **Build** stages switch from blue to green.
3. When the **Review** stage switches to blue, choose the white **Review** button.
4. Write an approval comment in the **Comments** textbox.
5. Choose the orange **Approve** button.
6. Wait for the **Review** and **Deploy** stages to switch to green.
7. Select the **AWS Elastic Beanstalk** link in the **Deploy** stage. A new tab listing your Elastic Beanstalk environments will open.
8. Select the URL in the **Devopsgettingstarted-env** row. You should see a webpage with a white background and the text you had in your most recent GitHub commit.

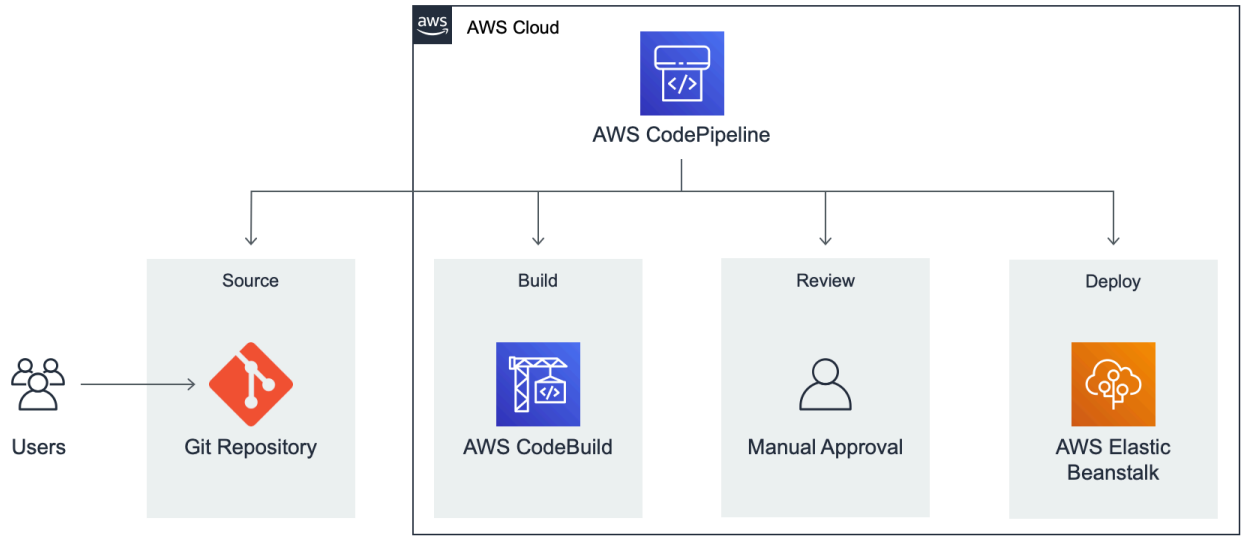
Congratulations!

You successfully built a continuous delivery pipeline on AWS! As a great next step, dive deeper into specific AWS technologies and take your application to the next level.

Application architecture

With all modules now completed, here is the architecture of what you built:

We have used AWS CodePipeline to add a review stage with manual approval to our continuous delivery pipeline. Now, our code changes will have to be reviewed and approved before they are deployed to AWS Elastic Beanstalk.



Clean up resources

(Optional) Delete AWS Elastic Beanstalk application

1. In a new browser window, open the AWS Elastic Beanstalk Console.
2. In the left navigation menu, click on **Applications.** You should see the **DevOpsGettingStarted** application listed under **All applications**.
3. Select the radio button next to **DevOpsGettingStarted.**
4. Click the **Actions** at the top of the page.
5. In the dropdown menu, select **Delete application**.
6. Type **DevOpsGettingStarted** in the text box to confirm deletion.
7. Click **Delete**.

(Optional) Delete pipeline in AWS CodePipeline

1. In a new browser window, open the AWS CodePipeline Console.
2. Select the **radio button** next to **Pipeline-DevOpsGettingStarted.**

3. Click **Delete pipeline** at the top of the page.
4. Type **delete** in the text box to confirm deletion.
5. Click **Delete**.

(Optional) Delete pipeline resources from Amazon S3 bucket

1. In a new browser window, open the Amazon S3 Console.
2. You should see a bucket named **codepipeline-us-west-2** followed by your AWS account number. Click on this bucket. Inside this bucket, you should see a folder named **Pipeline-DevOpsGettingStarted**.
3. Select the checkbox next to the **Pipeline-DevOpsGettingStarted** folder.
4. Click **Actions** from the dropdown menu.
5. In the dropdown menu, select **Delete**.
6. Click **Delete**.

(Optional) Delete build project in AWS CodeBuild

1. In a new browser window, open the AWS CodeBuild Console.
2. In the left navigation, click **Build projects** under **Build**. You should see the **Build-DevOpsGettingStarted** build project listed under **Build project**.
3. Select the radio button next to **Build-DevOpsGettingStarted.**
4. Click **Delete build project** at the top of the page.
5. Type **delete** in the text box to confirm deletion.
6. Click **Delete**.