Create a CI/CD pipeline to deploy your app to AWS Fargate

**SPL-TF-200-DOCICD-1 - Version 1.0.13**

© 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 demonstrates how to build a fully managed continuous integration and continuous delivery (CI/CD) pipeline for applications that run on Amazon Elastic Container Service (Amazon ECS). You use AWS CodePipeline to model, orchestrate, and visualize a three-stage pipeline that deploys a containerized application using a blue/green deployment strategy. This strategy launches a new version of the application that runs alongside the old version and then slowly shifts traffic to the new version. Blue/green deployments are commonly employed when performing in-place application upgrades. Administrators can use these to validate new code while simultaneously running the old application version. If errors are detected in the new code, the deployment can be rolled back quickly and reliably.

When complete, your pipeline automatically builds a new container image whenever new code is pushed to your source repository and then uses AWS CodeDeploy and Amazon ECS to manage its deployment and shift traffic to it.

OBJECTIVES

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

* Configure CodeCommit as a source control repository for an application.
* Create a CodeBuild project that uses a buildspec file to build a new Docker image and save it to Amazon ECR using an auditable and secure methodology.
* Create appspec.yaml and taskdef.json files that contain dynamic fields for use in blue/green deployments.
* Perform an in-place application upgrade using a blue/green deployment strategy configured in CodeDeploy and Amazon ECS.

PREREQUISITES

This lab requires the following:

* Access to a computer with Windows, macOS X, or Linux (Ubuntu, SuSE, or Red Hat)
* A modern internet browser such as Google Chrome or Mozilla Firefox

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.
* **Learn more:** Where to find more information.
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **Caution:** Information of special interest or importance (not important enough to cause problems with equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **Security:** An opportunity to incorporate security best practices.
* **Refresh:** A time when you might need to refresh a web browser page or list to show new information.
* **Task complete:** A conclusion or summary point in the lab.

**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.

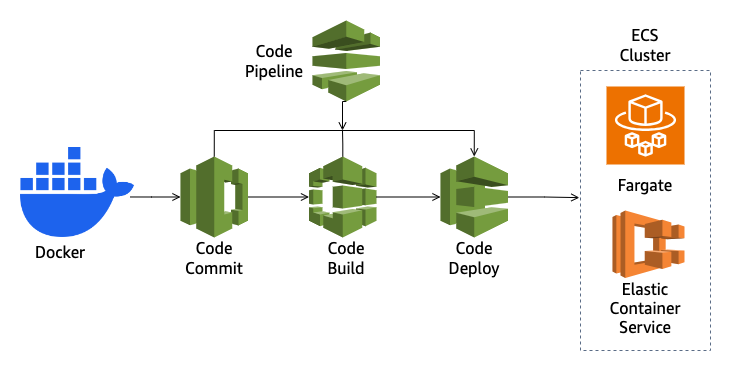
**Error: Choosing Start Lab has no effect**

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.

LAB ENVIRONMENT

The following diagram shows the basic architecture of the lab environment:



*Image description: The preceding diagram depicts the data flow of deploying a CI/CD pipeline using AWS CodePipeline. First a dockerfile is pushed to a repository stored in AWS CodeCommit. AWS CodeBuild is then used to build a container image, and then deployed to Amazon Elastic Container Service (ECS) cluster using AWS CodeDeploy. AWS Fargate is used as the launch type in a Green/Blue deployment in ECS.*

The following list details the major resources already provisioned for your use:

* An application running within an *Amazon ECS Cluster* using *AWS Fargate* as the launch type.
* A *Virtual Private Cloud (VPC)*, *Application Load Balancer*, and task definitions intended for the already created application to run.
* A *Cloud9* environment.

SERVICES USED IN THIS LAB

**Amazon Elastic Container Registry (Amazon ECR)**

Amazon ECR is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM. This is so that specified users or Amazon EC2 instances can access your container repositories and images. You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts. Amazon Elastic Container Registry Public supports public image repositories with resource-based permissions using AWS IAM so that specific users can access your public repositories to push images. Your images are publicly available to pull, either anonymously or using an Amazon ECR public authentication token.

**Amazon Elastic Container Service (Amazon ECS)**

Amazon ECS is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster. Your containers are defined in a task definition that you use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run and maintain a specified number of tasks simultaneously in a cluster. You can run your tasks and services on a serverless infrastructure that is managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.

Amazon ECS enables you to launch and stop your container-based applications by using simple API calls. You can also retrieve the state of your cluster from a centralized service and have access to many familiar Amazon EC2 features.

Amazon ECS is a regional service that simplifies running containers in a highly available manner across multiple Availability Zones within a Region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can create task definitions that define which container images run across your clusters. Your task definitions are used to run tasks or create services. Container images are stored in and pulled from container registries, for example, the Amazon Elastic Container Registry.

**AWS Cloud9**

AWS Cloud9 is a free, cloud-based integrated development environment (IDE) that lets you write, run, and debug your code using just a browser. The IDE includes a code editor, debugger, and terminal. AWS Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, and more, so you don’t need to install files or configure your development machine to start new projects. Because AWS Cloud9 IDE is cloud-based, you can work on your projects from your office, home, or anywhere using an internet-connected machine.

**AWS CodeBuild**

AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale your own build servers. It provides prepackaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more. You can also customize build environments in CodeBuild to use your own build tools. CodeBuild scales automatically to meet peak build requests.

**AWS CodeCommit**

AWS CodeCommit is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud. CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. CodeCommit eliminates the need for you to manage your own source control system or worry about scaling its infrastructure. You can use CodeCommit to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with your existing Git-based tools.

**AWS CodeDeploy**

AWS CodeDeploy is a fully managed deployment service that automates software deployments to various compute services, such as Amazon Elastic Compute Cloud (EC2), Amazon Elastic Container Service (ECS), AWS Lambda, and your on-premises servers. CodeDeploy can deploy application content that runs on a server and is stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories. CodeDeploy can also deploy a serverless Lambda function. You do not need to make changes to your existing code before you can use CodeDeploy.

**AWS CodePipeline**

AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. CodePipeline defines your release process workflow, and describes how a new code change progresses through your release process. A pipeline comprises a series of stages (e.g., build, test, and deploy), which act as logical divisions in your workflow. Each stage is made up of a sequence of actions, which are tasks such as building code or deploying to test environments. AWS CodePipeline provides you with a graphical user interface to create, configure, and manage your pipeline and its various stages and actions, allowing you to easily visualize and model your release process workflow.

**Task 1: Explore the application and review the Fargate configuration**

In this task, you connect to a basic application that displays news about *Amazon Web Services (AWS)*. The application has been deployed on *Amazon ECS* using the *AWS Fargate* launch type. After exploring the application, you review how the *Amazon ECS* cluster, service, and task definition have been configured.

Start by reviewing the application functionality.

1. Copy the **LoadBalancerURL** value that is listed to the left of these instructions. Paste the URL into a new web browser tab and press **Enter**.

**Note:**

* + As you can see, the application pulls in a list of recent announcements related to *AWS* services. Each item in the list includes a brief summary in addition to a headline that links to a blog post that provides additional details.
  + The website background is **blue**. In the subsequent tasks, you update the application source code and then build a CI/CD pipeline to deploy the change. If successful, the newly deployed application features a **green** background.

1. Take a moment to scroll through the list and find an announcement that interests you. Select a headline to open the page with information about it.

Now that you’ve seen what the application does, let’s review its configuration in *Amazon ECS*.

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

Elastic Container Service

.

**Note:** The **Clusters** page shows that one cluster, in addition to a single service and three tasks, has been deployed in your account.

**Consider:** Understanding tasks and services is critical for anyone who wants to manage or deploy applications on *Amazon ECS*. A task consists of one or more containers and their configuration that are scheduled by *Amazon ECS*. You can create tasks independently or as part of a service. When instantiated independently, a task launches containers, which run until the task is ended or the containers exit. Upon termination, no further containers are launched. By contrast, services ensure that a specified number of tasks are running at all times. Services provide administrators with granular control over task placement and scaling and can be automatically associated with a load balancer.

1. Choose the **FargateCluster** link.
2. Choose the **Services** tab and confirm that the **FargateService** is *Active* and displays *3/3 Tasks running*.

**Note:** *3/3 Tasks running* indicates that three replicas of the application are running concurrently in your *Amazon ECS* cluster.

1. Choose the **FargateService** link.

The Status panel shows that the *FargateService* is using an *Application Load Balancer* called *FargateALB* to distribute traffic across three targets. These targets are all in a *Healthy* state and are mapped to your *Amazon ECS* tasks.

1. At the top of the screen, choose the **Deployments** tab.
2. Review the information in the *Active deployment details* section. Notice that the **Deployment controller** is set to **CODE\_DEPLOY**.

**Note:** This is a requirement for blue/green deployments launched on **Fargate**. With the **CODE\_DEPLOY** deployment type, **CodeDeploy** can create a new green deployment, which runs in parallel with your existing blue deployment.

1. To return to the previous screen, in the breadcrumbs at the top of the page, choose **FargateCluster**.
2. Choose the **Tasks** tab to view the tasks that are running inside of the cluster.

**Note:** Notice the three tasks, each which lists the same *task definition*. The *task definition* identifies one or more container images to run in the task in addition to various parameters, including the following:

* + How much CPU and memory to use with each task or each container within a task.
  + The launch type to use, which determines the infrastructure that your tasks are hosted on.
  + The Docker networking mode to use for the containers in your task.
  + The *AWS Identity and Access Management (IAM)* role that your tasks use.

1. In the lower section of the screen, choose **one of the listed tasks**.

You are brought to the *Task overview* section under the *Configuration* tab.

1. Review the information in the *Configuration* panel, paying particular attention to the **Launch type** and **Network mode**. This task is launched using the *Fargate* serverless compute engine. However, because the *awsvpc* network mode has been selected, the task has been assigned its own elastic network interface. This provides the task with the same networking properties that it would have if it were launched on an *Amazon Elastic Compute Cloud (Amazon EC2)* instance.
2. Scroll to the bottom of the page and in the **Container details for application** page, locate the **Image URI** used by the container running in the task. The URI points to an image housed in your private *Amazon ECR* repository.

In the next steps, you navigate to your *Amazon ECR* repository and turn on *image tag immutability*.

1. In the left navigation pane, choose **Repositories**.
2. On the **Introducing the new Amazon ECR console experience** page, choose **Private repositories**.
3. On the *Private repositories* page, under *Repositories*, select the private repository with a name similar to **xxxxxxxxx-application**.
4. With the private repository selected, choose the **Actions** drop-down menu.
5. In the **Actions** drop-down menu, under **Edit**, choose **Repository**.
6. On the *Edit repository* page, in the *General settings* section:
   * Enable the **Tag immutability** toggle icon.

**Security:** Turning on tag immutability improves your security posture and mitigates against image spoofing. It does this by preventing image tags from being overwritten by subsequent image pushes using the same tag.

1. Choose **Save**.

**Task complete:** You have successfully configured AWS News application in Amazon ECS. It is time to start building your fully managed CI/CD pipeline.

**Task 2: Configure CodeCommit as a source control repository**

In this task, you connect to an *AWS Cloud9* environment that has been pre-populated with the application source code. You then create configuration and specification files that define your pipeline and a new task definition. Lastly, you edit the source code to change the application’s background color and save everything to a *CodeCommit* repository.

1. Copy the **Cloud9Environment** value that is listed to the left of these instructions. Paste the URL into a new web browser tab and press **Enter** to open the **AWS Cloud9 console**.

You are redirected to an AWS Cloud9 environment.

**Learn more:** AWS Cloud9 is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code within your browser. It comes prepackaged with many tools that are commonly used in application development, including Docker, Python, and the AWS Command Line Interface (AWS CLI).

1. You do not need the **Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch *Cloud9*, so choose the  next to each tab to close them.
2. **Command:** In the **AWS Cloud9** terminal, run the below command to view the files that you need to push to **AWS CodeCommit**:

ls ~/environment

**Expected output:**

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

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

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

Dockerfile index.js package.json routes static

**Note:** The following list describes files and directories in the repository that you cloned:

* + *Dockerfile*: A file that lists the commands used to create the application Docker image.
  + *index.js*: A file that renders the application code and handles routing.
  + *package.json*: A file that lists metadata, dependencies, and scripts required by Node during the application build phase.
  + *routes*: A directory that contains a JavaScript file that makes a request to an API and fetches a list of *AWS* announcements.
  + *static*: A static directory that contains various files that define how content should be organized and displayed on screen.

1. **Command:** Take a closer look at the **Dockerfile** in this directory. To view the contents of the **Dockerfile**, enter the below command:

cat Dockerfile

**Expected output:**

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

# SPDX-License-Identifier: MIT-0

FROM public.ecr.aws/d5z8k9g9/node18-alpine3.15:latest

# Create app directory

WORKDIR /usr/src/app

# App dependencies

COPY package.json ./

# Download the dependencies listed in the package.json file and create the node\_modules directory

RUN npm install

# Copy the application source code

COPY index.js .

COPY routes ./routes

COPY static ./static

# Listen on port 80

EXPOSE 80

# Start the application

CMD [ "node", "index.js" ]

**Note:** As you can see, this is a fairly basic Dockerfile that builds a Node.js application and exposes port 80.

However, to launch this application on *Amazon ECS* and build a pipeline that automates its deployment, you need three additional files:

* + *buildspec.yaml*: *CodeBuild* uses the commands and parameters in the buildspec file to build a Docker image.
  + *appspec.yaml*: *CodeDeploy* uses the appspec file to select a task definition.
  + *taskdef.json*: Recall that all three tasks that are currently running in your *Amazon ECS* service reference the same task definition. After updating the application source code and building a new container, you need a second task definition that points to it. The taskdef.json file is used to create this new task definition that points to your updated application image.

In the following steps, you create these files.

1. **Command:** To create a buildspec file, enter the below command:
2. cat << 'EOF' > ~/environment/buildspec.yaml
3. version: 0.2
4. phases:
5. pre\_build:
6. commands:
7. - echo Logging in to Amazon ECR...
8. - aws --version
9. - ACCOUNT\_ID=$(echo $CODEBUILD\_BUILD\_ARN | cut -f5 -d ':') && echo "The Account ID is $ACCOUNT\_ID"
10. - echo "The AWS Region is $AWS\_DEFAULT\_REGION"
11. - REPOSITORY\_URI=$ACCOUNT\_ID.dkr.ecr.$AWS\_DEFAULT\_REGION.amazonaws.com/$ACCOUNT\_ID-application
12. - echo "The Repository URI is $REPOSITORY\_URI"
13. - aws ecr get-login-password --region $AWS\_DEFAULT\_REGION | docker login --username AWS --password-stdin $REPOSITORY\_URI
14. - COMMIT\_HASH=$(echo $CODEBUILD\_RESOLVED\_SOURCE\_VERSION | cut -c 1-7)
15. - IMAGE\_TAG=$COMMIT\_HASH
16. build:
17. on-failure: ABORT
18. commands:
19. - echo Build started on `date`
20. - echo Building the Docker image...
21. - docker build -t $REPOSITORY\_URI:$IMAGE\_TAG .
22. - docker tag $REPOSITORY\_URI:$IMAGE\_TAG $REPOSITORY\_URI:$IMAGE\_TAG
23. post\_build:
24. commands:
25. - echo Build completed on `date`
26. - echo Pushing the Docker image...
27. - docker push $REPOSITORY\_URI:$IMAGE\_TAG
28. - echo Writing image definitions file...
29. - printf '[{"name":”myimage","imageUri":"%s"}]' $REPOSITORY\_URI:$IMAGE\_TAG > imagedefinitions.json
30. - printf '{"ImageURI":"%s"}' $REPOSITORY\_URI:$IMAGE\_TAG > imageDetail.json
31. artifacts:
32. files:
33. - imagedefinitions.json
34. - imageDetail.json
35. - appspec.yaml
36. - taskdef.json

EOF

**Expected output:**

*None, unless there is an error*

**Note:** The commands in the buildspec file are organized into phases: install, pre\_build, build, and post\_build. The artifacts section at the bottom of the file points to the location and configuration of the build output. Administrators do not need to include all phases in a buildspec file; however, the phase names should not be changed.

**Consider:** Take a moment to review the structure of the buildspec file and note the following:

* + The version is set to 0.2.
  + The pre\_build phase is an optional phase that is used to run commands before building the application code. In this example, the pre\_build phase is used to set variables that are used throughout the build process and authenticate into *Amazon ECR*. In some cases, for example *$AWS\_DEFAULT\_REGION*, the file references pre-configured *CodeBuild* environment variables. In other cases, variables are set using shell commands.
  + Each newly built image is tagged with the corresponding *commit ID* from *CodeCommit*.
  + The commands included in the build phase are run sequentially. In this case, the *ABORT* command has been included to end the build if any of the commands fail.
  + The post\_build phase pushes the Docker image produced in the build phase to *Amazon ECR*.
  + Upon completion, the build produces artifacts called imageDetail.json and imagedefinitions.json, both of which are saved to the environment root directory. These files are used in the deploy phase of your pipeline and indicate which image to deploy to *Amazon ECS*.
  + The artifacts section specifies that the *appspec.yaml* and *taskdef.json* files uploaded to your *CodeCommit* repository be included as build outputs. Without these files, your deployment fails.

**Note:** Because each new image is tagged with the corresponding commit ID from *CodeCommit* and *image tag immutability* has been turned on in *Amazon ECR*, you now have a fully auditable chain of custody for each image.

1. **Command:** Next, to create the **appspec.yaml** file, enter the following command:
2. cat << EOF > ~/environment/appspec.yaml
3. version: 0.0
4. Resources:
5. - TargetService:
6. Type: AWS::ECS::Service
7. Properties:
8. TaskDefinition: <TASK\_DEFINITION>
9. LoadBalancerInfo:
10. ContainerName: "application"
11. ContainerPort: 80

EOF

**Expected output:**

*None, unless there is an error*

**Note:** This file does not point to a specific task definition. Instead, the TaskDefinition label is populated with a placeholder. *CodeBuild* replaces this with your new task definition at build time. You can optionally include additional information, such as *AWS Lambda* functions that correspond to lifecycle events during your *Amazon ECS* deployment, in appspec files.

1. **Command:** Before you create your *taskdef.json* file, you need to set some environment variables. To create the variables, enter the following command:
2. TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
3. AWS\_REGION=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
4. FAMILY=$(aws ecs list-task-definition-families --status ACTIVE --output text | awk '{print $NF}')
5. ACCOUNT\_ID=$(aws sts get-caller-identity --query Account --output text)

printf "You are using $AWS\_REGION region\nYour task definition family is $FAMILY\nYour account ID is $ACCOUNT\_ID\n"

**Expected output:**

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

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

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

You are using us-west-2 region

Your task definition family is xz8x1SNhwGYbcEHTLZtVKU-2-TaskDefinition-v3QkGbcSenuq

Your account ID is 111111111111

1. **Command:** Lastly, to create the *taskdef.json* file, enter the following command :
2. cat << EOF > ~/environment/taskdef.json
3. {
4. "containerDefinitions": [
5. {
6. "name": "application",
7. "image": "<IMAGE\_NAME>",
8. "portMappings": [
9. {
10. "containerPort": 80,
11. "hostPort": 80,
12. "protocol": "tcp"
13. }
14. ],
15. "essential": true,
16. "logConfiguration": {
17. "logDriver": "awslogs",
18. "options": {
19. "awslogs-group": "cicd-logs",
20. "awslogs-region": "$AWS\_REGION",
21. "awslogs-stream-prefix": "ecs"
22. },
23. },
24. }
25. ],
26. "family": "$FAMILY",
27. "taskRoleArn": "arn:aws:iam::$ACCOUNT\_ID:role/ecsTaskExecutionRole",
28. "executionRoleArn": "arn:aws:iam::$ACCOUNT\_ID:role/ecsTaskExecutionRole",
29. "networkMode": "awsvpc",
30. "status": "ACTIVE",
31. "compatibilities": [
32. "EC2",
33. "FARGATE"
34. ],
35. "requiresCompatibilities": [
36. "FARGATE"
37. ],
38. "cpu": "256",
39. "memory": "512",
40. "tags": [
41. {
42. "key": "Name",
43. "value": "GreenTaskDefinition"
44. }
45. ]
46. }

EOF

**Expected output:**

*None, unless there is an error*

**Note:** This task definition is identical to the one that is currently running in your cluster, with one important difference—the image label is using “<IMAGE\_NAME>” as a placeholder. When the pipeline is run, *CodeDeploy* updates this value with the correct image URI at deployment time. Also, the file references an *Amazon CloudWatch log group*, which must be included; otherwise, your tasks fail.

**Consider:** Recall that when you opened the *AWS News* application, the website background was blue. You received feedback from colleagues that the website is too dark, and they’ve asked you to change the background. You now modify the application code to make the background green. In subsequent tasks, you build a blue/green deployment that uses this modified application code in the green environment.

1. **Command:** To change the application background color to green, enter the following command:

sed -i 's/282F3D/1D8102/g' ~/environment/static/css/app.css

**Expected output:**

*None, unless there is an error*

Now that you’ve updated the application code and created files that define your pipeline, you create a new *CodeCommit* repository and push all the files into it. Your pipeline uses this repository for source control.

1. **Command:** To create a new *CodeCommit* repository and save its Secure Shell (SSH) connection URL to an environment variable, enter the following command:
2. export SRC\_REPO\_URL=$( \
3. aws codecommit create-repository \
4. --repository-name pipeline-source-code \
5. --repository-description "Repository for AWS News application source code" \
6. | jq -r '.[].cloneUrlSsh'
7. )

echo "Repo successfully created. Use $SRC\_REPO\_URL to clone the repository"

**Expected output:**

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

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

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

Repo successfully created. Use ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/pipeline-source-code to clone the repository

1. **Command:** To update the Git configuration enter the following command:

git config --global init.defaultBranch main

**Expected output:**

*None, unless there is an error*

1. **Command:** To initialize the *environment* directory as a local Git repository, enter the following command:
2. cd ~/environment

git init

**Expected output:**

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

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

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

Initialized empty Git repository in /home/ec2-user/environment/.git/

1. **Command:** To stage your application files, commit the updated files to the local repo, and push the application code to the *main* branch in your *AWS CodeCommit* repo, enter the following commands. When asked if you would like to connect to **git-codecommit.[AWS\_REGION].amazonaws.com**, enter

yes

:

git add .

git commit -m "initial commit"

git push --set-upstream $SRC\_REPO\_URL main

**Expected output:**

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

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

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

AWSLabsUser-vKJpDE7K1YuFQ9H2h2Riiv:~/environment (main) $ git add .

AWSLabsUser-vKJpDE7K1YuFQ9H2h2Riiv:~/environment (main) $ git commit -m "initial commit"

[main (root-commit) 306312c] initial commit

24 files changed, 291660 insertions(+)

create mode 100755 .DS\_Store

create mode 100644 .c9/.nakignore

create mode 100644 .c9/Cloud9-Lab-IDE/meta.json

create mode 100644 .c9/amazonwebservices.aws-toolkit-vscode/cloudformation.schema.json

create mode 100644 .c9/amazonwebservices.aws-toolkit-vscode/sam.schema.json

create mode 100644 .c9/launch.json

create mode 100644 .c9/out\_of\_memory

create mode 100644 .c9/project.settings

create mode 100644 .c9/tasks.json

create mode 100755 Dockerfile

create mode 100644 appspec.yaml

create mode 100644 buildspec.yaml

create mode 100755 index.js

create mode 100755 package.json

create mode 100755 routes/news.js

create mode 100755 static/.DS\_Store

create mode 100755 static/css/.DS\_Store

create mode 100755 static/css/amazon-ember-regular.ttf

create mode 100755 static/css/app.css

create mode 100755 static/img/aws-training-logo.png

create mode 100755 static/img/favicon.ico

create mode 100755 static/index.html

create mode 100755 static/js/news.js

create mode 100644 taskdef.json

AWSLabsUser-vKJpDE7K1YuFQ9H2h2Riiv:~/environment (main) $ git push --set-upstream $SRC\_REPO\_URL main

The authenticity of host 'git-codecommit.us-west-2.amazonaws.com (52.119.168.71)' can't be established.

RSA key fingerprint is SHA256:0pJx9SQpkbPUAHwy58UVIq0IHcyo1fwCpOOuVgcAWPo.

RSA key fingerprint is MD5:a8:68:53:e3:99:ac:6e:d7:04:7e:f7:92:95:77:a9:77.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'git-codecommit.us-west-2.amazonaws.com,52.119.168.71' (RSA) to the list of known hosts.

Enumerating objects: 34, done.

Counting objects: 100% (34/34), done.

Delta compression using up to 2 threads

Compressing objects: 100% (29/29), done.

Writing objects: 100% (34/34), 254.96 KiB | 1.16 MiB/s, done.

Total 34 (delta 4), reused 0 (delta 0), pack-reused 0

To ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/pipeline-source-code

\* [new branch] main -> main

branch 'main' set up to track 'ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/pipeline-source-code/main'.

**Task complete:** You have successfully created a CodeCommit repository and uploaded the application code to it. In the following tasks, you use this repository as the source for your CI/CD pipeline.

**Task 3: Create a CodeDeploy application and deployment group**

In this task, you prepare for your blue/green deployment by creating a *CodeDeploy* application and deployment. An application is a name that uniquely identifies the code that you want to deploy. *CodeDeploy* uses the application to ensure that the correct combination of revision, deployment configuration, instances, and Auto Scaling groups are referenced when the pipeline is invoked.

Applications work hand-in-hand with deployment groups. *CodeDeploy* uses deployment groups to specify the *Amazon ECS* service, load balancer, and target groups for your revised application code. They also include configuration details, such as how and when traffic should be rerouted to the new tasks that your pipeline creates.

1. Return to the browser tab connected to the **AWS Management Console**.

**Note:** If you inadvertently closed that tab, open a new browser tab and navigate to the [AWS Management Console](https://console.aws.amazon.com/). Verify that the AWS Region displayed at the top of the screen matches the **AWSRegion** value in the panel to the left of these instructions.

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

CodeDeploy

.

1. In the left navigation pane, in the *CodeDeploy* section, under *Deploy . CodeDeploy*, choose **Applications**.
2. Choose **Create application**.
3. On the *Create application* page, in the *Application configuration* section:
   * For **Application name**, enter

awsnews-application

.

* + For **Compute platform**, select **Amazon ECS** from the drop-down menu.

1. Choose **Create application**.

Now that your application has been created, it needs a deployment group.

1. In the *Deployment groups* section, choose **Create deployment group**.
2. On the *Create deployment group* page:
   * For the **Deployment group name**, enter

awsnews-deployment-group

* + For **Service role**, choose **CodeDeployServiceRole**.
  + In the **Environment configuration** section:
    - For **Choose an ECS cluster name**, select **FargateCluster** from the drop-down menu.
    - For **Choose an ECS service name**, select **FargateService** from the drop-down menu.
  + In the **Load balancers** section:
    - For **Choose a load balancer**, select **FargateALB** from the drop-down menu.
    - For **Production listener port**, select **HTTP: 80** from the drop-down menu.
    - Test listener port is not needed.
    - For **Target group 1 name**, select **BlueTargetGroup** from the drop-down menu.
    - For **Target group 2 name**, select **GreenTargetGroup** from the drop-down menu.
  + For **Deployment settings**, **Traffic rerouting**, ensure that **Reroute traffic immediately** is selected.
  + For **Original revision termination**, set **Days** to

0

; **Hours** to

0

; and **Minutes** to

5

.

1. Choose **Create deployment group**.

**Note:** Your deployment group adds a second target group, called **GreenTargetGroup**, to your service. Both target groups run in parallel for 5 minutes, so you can test the new target group and roll back if necessary. At the end of the 5 minutes, the original target group gets terminated.

**Task complete:** You have successfully created a CodeDeploy application and deployment group.

**Task 4: Build your CI/CD pipeline**

In this task, you use *CodePipeline* to configure a pipeline that automates the release process for your application. When complete, the pipeline invokes whenever *CodeCommit* detects changes. At this point, *CodeBuild* builds a new image, pushes it to *Amazon ECR*, and then *CodeDeploy* deploys the new image to your *Amazon ECS* cluster.

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

CodePipeline

.

1. Choose **Create pipeline**.

**Note:** If you see an introductory page instead of the dashboard, choose **Get Started Now**.

1. On the *Choose pipeline settings* page, in the *Pipeline settings* section:
   * For **Pipeline name**, enter

AwsNewsPipeline

.

* + For **Service role**, select **Existing service role**.
  + For **Role ARN** field, select **CodePipelineServiceRole** from the drop-down menu.

1. Expand **Advanced Settings**, and then:
   * For **Artifact store**, select **Custom location**.
   * For **Bucket**, select a name similar to **codepipelineartifactbucket-XXXXXXXXXXXX** from the drop-down menu.
2. Choose **Next**.

**Consider:** In this lab, you configure *CodePipeline* to be invoked whenever new code is pushed to your *CodeCommit* repository. However, *CodePipeline* is highly versatile and can be invoked by various objects and events, including the following:

* + *Branch*: *CodePipeline* monitors a branch in a *CodeCommit*, *GitHub*, or *BitBucket* repository and is invoked whenever new code is committed to it.
  + *Amazon S3 object or folder*: *CodePipeline* monitors an object or folder in an S3 bucket and is invoked whenever changes are detected.
  + *Amazon ECR*: *CodePipeline* monitors an image repository and is invoked whenever a new image is pushed to it.

1. On the *Add source stage* page, in the *Source* section:
   * For **Source provider**, select **AWS CodeCommit** from the drop-down menu.

**Note:** The menu expands so that you can configure the repository that you created as the source for your pipeline.

* + For the **Repository name**, select **pipeline-source-code** from the drop-down menu.
  + For the **Branch name**, select **main** from the drop-down menu.

Leave the rest of the values at the default settings.

1. Choose **Next**.

Now that you’ve configured *CodeCommit* as your pipeline source, it is time to configure the build stage. You can configure *CodePipeline* to use either *CodeBuild* or *Jenkins* as its build provider. In this lab, you use *CodeBuild*. The build project includes information about how the build should be run, including where to find the source code, which build environment to use, which build commands to run, and where to store the build output.

1. On the *Add build stage* page, in the *Build - optional* section:
   * For **Build provider**, select **AWS CodeBuild** from the drop-down menu.
   * For **Region**, select the option that matches the **AWSRegion** value that is listed to the left of these instructions.
   * For **Project name**, choose **Create project**.

**Note:** A pop-up window appears on screen.

1. On the *Create build project* page, in the *Project configuration* section:
   * For **Project name**, enter

awsnews-build-project

.

1. In the **Environment** section:
   * For **Provisioning model**, ensure that **On-demand** is selected.
   * For **Environment image**, ensure that **Managed image** is selected.
   * For **Compute**, ensure that **EC2** is selected.
   * For **Operating system**, ensure that **Amazon Linux** is selected.
   * For **Runtimes**, ensure that **Standard** is selected.
   * For **Image**, ensure that **aws/codebuild/amazonlinux2-x86\_64-standard:5.0** is selected.
   * For **Image version**, ensure that **Always use the latest image for this runtime version** is selected.
   * For **Service role**, select **Existing service role**.
   * For **Role ARN**, select **CodeBuildServiceRole** from the drop-down menu.
   * Expand **Additional configuration**, and then:
     + For **Privileged**, select **Enable this flag if you want to build Docker images or want your builds to get elevated privileges**.
2. In the *Buildspec* section:
   * For **Build specifications**, select **Use a buildspec file**.
   * For **Buildspec name - *optional***, enter

buildspec.yaml

.

1. In the **Logs** section:
   * For **Group name - *optional***, enter

cicd-logs

.

* + For **Stream name prefix - *optional***, enter

codebuild

.

1. Choose **Continue to CodePipeline**.
2. On the **Leave site?** pop-up box, choose **Leave**.

**Note:** You are brought to the **Add build stage** page on the **Create new pipeline** console.

**Successfully created awsnews-build-project in CodeBuild.**

1. Choose **Next**.
2. On the *Add Deploy stage* page, in the *Deploy - optional* section:
   * For **Deploy provider**, select **Amazon ECS (Blue/Green)** from the drop-down menu.

**Note:** The Amazon ECS (Blue/Green) option is near the bottom of the dropdown menu. If it is not visible, you might need to scroll further down the list.

* + For **Region**, select the option that matches the **AWSRegion** value that is listed to the left of these instructions.
  + For **AWS CodeDeploy application name**, select **awsnews-application** from the drop-down menu.
  + For **AWS CodeDeploy deployment group**, select **awsnews-deployment-group** from the drop-down menu.
  + For **Amazon ECS task definition**, choose **BuildArtifact** from the dropdown menu. Keep the text field next to it empty.
  + For **AWS CodeDeploy AppSpec file**, choose **BuildArtifact** from the dropdown menu. Keep the text field next to it empty.
  + For **Input artifact with image details**, choose **BuildArtifact** from the dropdown menu.
  + For **Placeholder text in the task definition**, enter

IMAGE\_NAME

1. Choose **Next**.

**Learn more:** The *Review* page displays the details of the pipeline that you just configured. It uses the application code in *CodeCommit* to build an image and pushes it to *Amazon ECR*. The appspec.yaml and taskdef.json files are then used to create an isolated service environment for your green deployment. *CodeDeploy* then reroutes load balancer traffic from the containers that run the blue environment to the new set of containers that run the green environment. After traffic has been successfully rerouted to the new containers, the existing containers are terminated.

It is now time to test your pipeline.

1. Choose **Create pipeline**.

**Task complete:** You have successfully configured a CI/CD pipeline and initiated your first blue/green deployment. In the next task, observe its progress.

**Task 5: Observe a blue/green deployment**

In this task, you observe as your pipeline runs through its stages and then watch as *CodeDeploy* reroutes traffic to the new version of your application.

**Note:** Your first pipeline run is in process and should pass through the source stage quickly because it only needs to clone your repository, but the build and deploy stages take considerably longer.

1. When the pipeline passes to the build phase, to view more information about the build process, in the *Build* panel, under  **In progress**, choose **View details**.

**Note:** The Details link might not appear until 15–30 seconds after the pipeline enters the build phase.

1. On the **Action execution details** tab, choose **Logs** tab to view the tail logs.

**Note:** It takes 1–2 minutes for *CodeBuild* to provision a new build environment. When the provisioning has completed, logs appear in the Build logs popup window.

1. Wait for the build to complete and once the **Status** changes to **Succeeded**, choose **Done** on the pop-up window.
2. In the left navigation pane, in the *CodePipeline* section, under *Deploy . CodeDeploy*, choose **Deployments**.
3. The *Deployment history* panel should show the one deployment **In progress**. To view additional details about its status, choose its **Deployment Id** link.

**Note:** The deployment passes through four steps:

* + *Step 1*: Amazon ECS uses the taskdef.json file to build a new task definition.
  + *Step 2*: CodeDeploy reroutes traffic to the new task set.
  + *Step 3*: The original tasks remain live for 5 minutes after traffic has been rerouted. During this time, you can perform testing and invoke a rollback if you detect errors.
  + *Step 4*: The original task set is terminated.

1. Wait for the deployment to reach **Step 3** and then return to the browser tab that contains the **AWS News** application and  **Refresh** the page. Confirm that the website background is **green**, which indicates that the deployment was successful.
2. Because you have confirmed that the deployment was successful, you do not need to wait the entire 5 minutes before terminating the original task set. Choose **Terminate original task set**.
3. Scroll to the *Deployment lifecycle events* panel at the bottom of the page and confirm that the status of all events is **Succeeded**.

**Task complete:** You have successfully built a fully managed pipeline that performs a blue/green deployment for the AWS News application.

**Conclusion**

You have successfully done the following:

* Configured CodeCommit as a source control repository for an application.
* Created a CodeBuild project that uses a buildspec file to build a new Docker image and save it to Amazon ECR using an auditable and secure methodology.
* Created appspec.yaml and taskdef.json files that contain dynamic fields for use in blue/green deployments.
* Performed an in-place application upgrade using a blue/green deployment strategy configured in CodeDeploy and Amazon ECS.

**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.

**Additional resources**

* [What is AWS CodePipeline?](https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html).
* [Blue/Green Deployments on AWS](https://docs.aws.amazon.com/whitepapers/latest/blue-green-deployments/welcome.html?did=wp_card&trk=wp_card).
* [Practicing Continuous Integration and Continuous Delivery on AWS](https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/welcome.html?did=wp_card&trk=wp_card).

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).