**Getting Started with DevOps on AWS**

Getting Started with DevOps on AWS

2 hours10 Credits



**SPL-TF-100-DOGTST-1 - Version 1.0.2**

© 2021 Amazon Web Services, Inc. and 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.

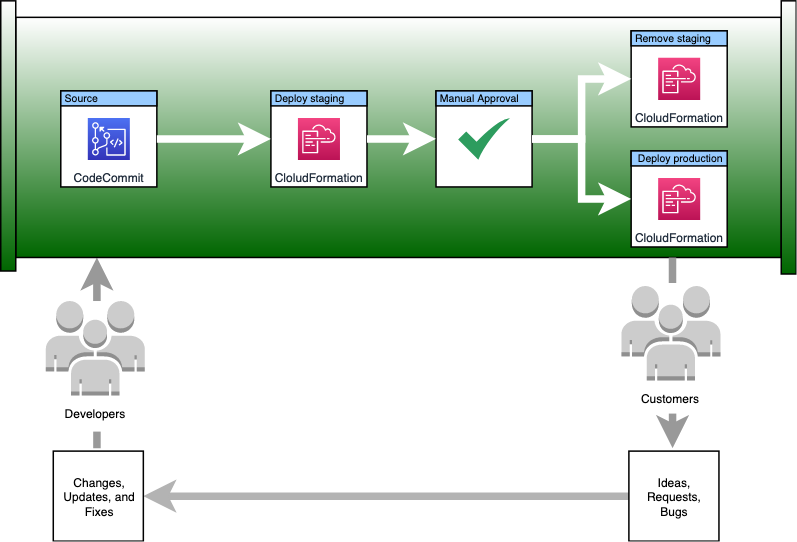
Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

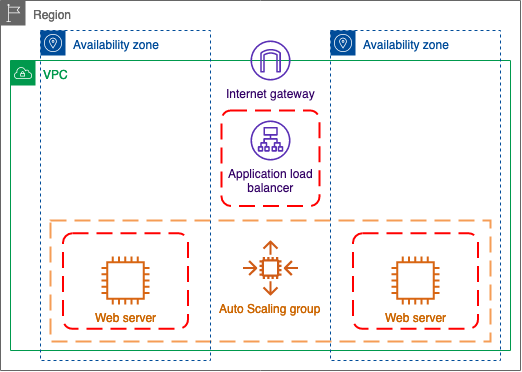
DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

In this lab, you will learn how to use AWS DevOps tools to create, configure, and control a Continuous Integration and Continuous Delivery (CI/CD) pipeline. You create a web application with infrastructure as code. This application will be automatically staged in one AWS region and, after your manual approval, deployed to a second region while removing the staging environment.

This lab creates a CI/CD pipeline like the diagram below.



The pipeline deploys a web application to a staging environment to verify it deploys successfully and then deploys the application to the production environment. The web application includes its own network environment, application load balancer, and auto scaling group. This is illustrated in the following diagram:



The AWS services used in this lab are:

* **AWS Cloud9** is used as an IDE to develop and make any code changes.
* **AWS CodeCommit** which acts as the source control repository.
* **AWS CodePipeline** which will orchestrate the phases of the CI/CD pipeline. It automatically starts the pipeline when source changes are submitted to the repository.
* **AWS CloudFormation** treats infrastructure as code, and using a parameterized template, it consistently provisions the web application's environments, in two distinct regions. The template describes resources such as the Amazon EC2 instances, security groups, and required network resources to host the web application in multiple regions.
* **Amazon Elastic Compute Cloud (EC2)** hosts the webpage. Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud.

**Learning objectives**

* Define and provision your infrastructure as a code using AWS CloudFormation.
* Create a code repository and establish continuous integration using AWS CodeCommit.
* Use the Cloud9 IDE to make code changes and integrate to a central repository.
* Build a release pipeline to automate continuous integration and continuous delivery (CI/CD) orchestration using AWS CodePipeline.
* Edit the CI/CD pipeline to add stages and actions, both sequential and parallel.
* Manage immutable deployments in a pipeline using runtime stage parameters.
* Control CI/CD process using manual approvals.

**Technical knowledge prerequisites**

This is an introductory lab. No previous experience is required. Familiarity with basic navigation of the AWS Management Console and editing scripts is recommended but not required.

**Start Lab**

1. At the top of your screen, launch your lab by choosing **Start Lab**

This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

 If you are prompted for a token, use the one distributed to you (or credits you have purchased).

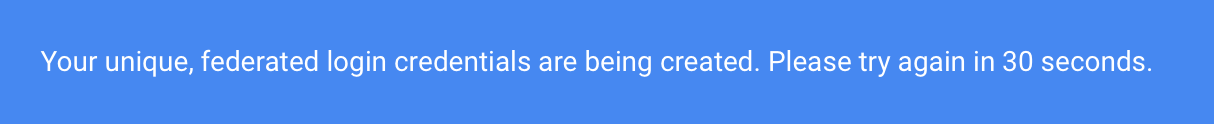
1. Open your lab by choosing **Open Console**

This automatically logs you in to the AWS Management Console.

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

Common Login Errors

**Error: Federated login credentials**



If you see this message:

* Close the browser tab to return to your initial lab window
* Wait a few seconds
* Choose **Open Console** again

You should now be able to access the AWS Management Console.

**Error: You must first log out**



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

* Choose **click here**
* Close your browser tab to return to your initial lab window
* Choose **Open Console** again

**Starting environment**

At the start of this lab, the following resources have been provided for you.

* AWS Cloud9 development environment
* AWS Identity and Access Management (IAM) Roles

**Task 1: Create the infrastructure code for the environments**

Developers, which also include operations staff in a DevOps environment, can be more efficient when they can immediately experience the affects of their code changes. Organizations become more agile when system improvements and fixes can be approved and pushed to production quickly and safely.

In this task, you automate the deployment of your staging environment. Since you're using infrastructure as code, your staging resources are identical to the production resources that you automate later. This helps to make sure any testing that you perform in staging will match what will happen when the new version is deployed to production.

Since the staging environment can be easily removed when it's no longer in use, you don't have to worry about the costs of having long running resources that aren't absolutely necessary.

Task 1.1: Create the AWS CodeCommit repository

A version control system is a system that records changes to a file or set of files over time so that you can recall specific versions later. One of the most popular version control systems today is Git. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. With Git, each developer or engineer, has a local version of the code to work on. You commit your changes locally with a comment summarizing the changes. You can easily move between versions (commits) if you need to roll back changes.

To share your code with other developers or to use with automation systems, you need to push a copy of your local repository to a shared repository. When you use a shared repository all of your changes are merged with anyone else that may have made changes to the code. Git has robust support for conflict resolution when it's needed. You use AWS CodeCommit for a shared repository in this lab.

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 familiar to users of Git-based repositories, but even those unfamiliar should find the transition to CodeCommit relatively simple. CodeCommit provides a console for the easy creation of repositories and the listing of existing repositories and branches. In a few simple steps, users can find information about a repository and clone it to their computer, creating a local repo where they can make changes and then push them to the CodeCommit repository. Users can work from the command line on their local machines or use a GUI-based editor.

1. On the navigation bar, choose the **Services** button, and choose **CodeCommit**.
2. Choose **Create repository**
3. For **Repository name**, enter 
4. Choose **Create**

Your repository has been created and is ready for use. Before you leave this screen copy the address that you use to connect to it.

1. Choose the **Clone URL** button, and choose **Clone HTTPS**.

This copied the repository url to your clipboard. This is the url that you tell the local git client to use when creating your local repository. Paste it into a local text editor to be used in a few steps.

Task 1.2: Download the initial version of your infrastructure code and add it to your repository

You now need to clone the repository that you just created and create your infrastructure code in it. Instead of using your local environment that may not have the tools you need installed, use an AWS Cloud9 development environment to create your code.

AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. 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. Since your Cloud9 IDE is cloud-based, you can work on your projects from your office, home, or anywhere using an internet-connected machine.

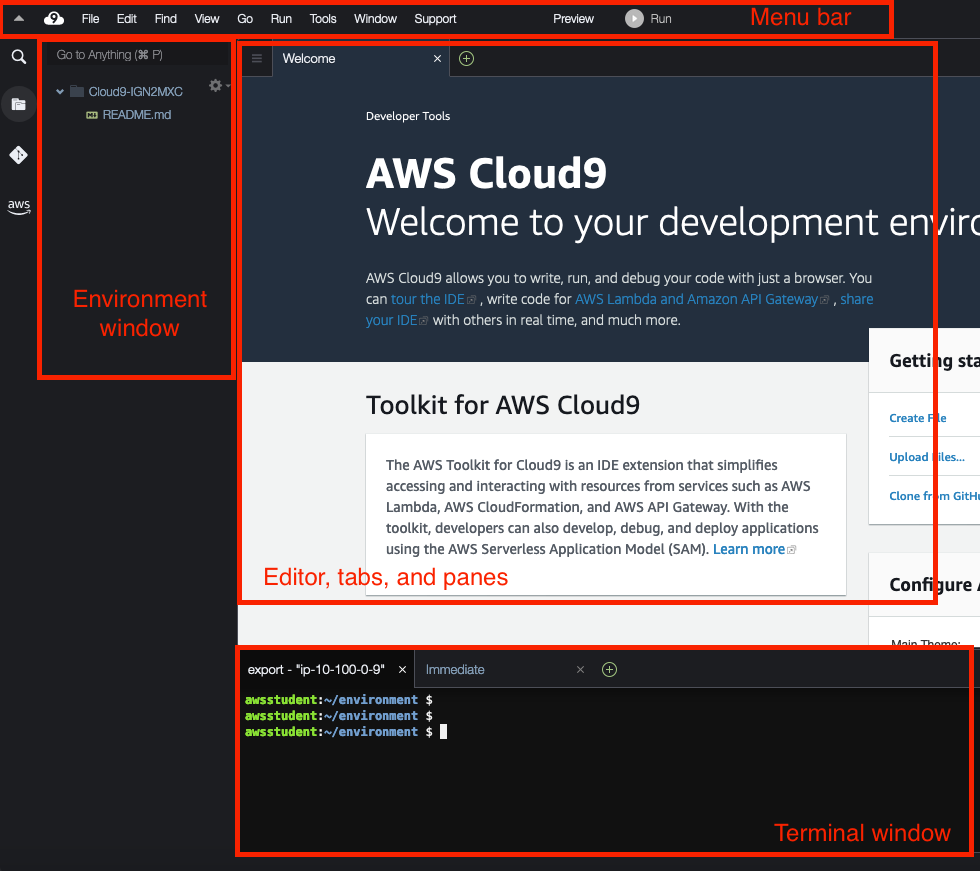
For this lab, a Cloud9 development environment has already been provisioned for you. Go ahead and connect to it.

1. On the navigation bar, choose the **Services** button, and choose **Cloud9**.
2. Choose **Open IDE**

This will open the Cloud9 IDE in a new web browser tab.

**The Cloud9 IDE**

The AWS Cloud9 integrated development environment (IDE) has several sections that you use throughout this lab. The color of your IDE and the screenshot below may be different but they should contain the same content. When you launch an IDE for the first time with your IAM account, the default theme is randomized but you can change it if you need to.



**Menu bar**

The menu bar, at the top edge of the IDE, contains common commands for working with files and code and changing IDE settings.

**Environment window**

The Environment window shows a list of your folders and files in the environment.

**Editor, tabs, and panes**

The editor is where you can do things such as write code, run a terminal session, and change IDE settings. Each instance of an open file, terminal session, and so on is represented by a tab.

**Terminal window**

You can run one or more terminal sessions in the IDE. This allows you to run commands on the EC2 instance that was created for your environment. By default, the IAM permissions granted to the terminal window mirror the permissions granted to your AWS user account.

1. Replace ***[Clone\_URL]*** with the **HTTPs URL** you copied in the last task and run the following code in the **Terminal** window to clone your repository.

git clone [Clone\_URL]

content\_copy

1. Run the following command to change to the cloned directory.

cd web-application

content\_copy

1. Run the following command to create a new **dev** branch in the local repository.

git checkout -b dev

content\_copy

AWS CloudFormation is a service that helps you model and set up your Amazon Web Services resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you work with templates and stacks. You create templates to describe your AWS resources and their properties. Whenever you create a stack, AWS CloudFormation provisions the resources that are described in your template.

An AWS CloudFormation template is a JSON or YAML formatted text file. You can save these files with any extension, such as .json, .yaml, .template, or .txt. AWS CloudFormation uses these templates as blueprints for building your AWS resources.

1. Download the **application.template** file to your Cloud9 environment with the following command.

curl https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/SPL-TF-100-DOGTST-1/v1/application.template > application.template

content\_copy

This Cloudformation template will create your entire web application in the 269 lines of code that it contains. As your pipeline runs for the first time, you will explore the contents of this file.

1. Run the following code to stage all changes in the folder to be committed

git add .

content\_copy

1. Commit the changed files to the local git repository with the next command.

git commit -m "initial commit with application template"

content\_copy

1. Finally, run the command below to push your local commit to the remote CodeCommit repository while creating the remote dev branch.

git push --set-upstream origin dev

content\_copy

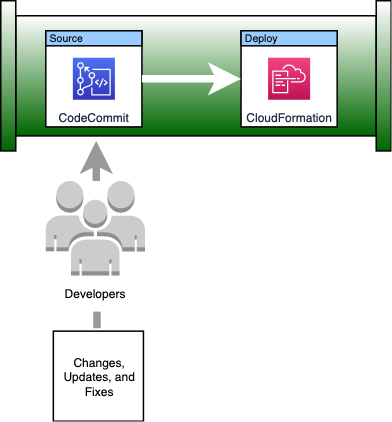
**Task 2: Deploy the application to a staging region using CodePipeline**

Now that you have your CloudFormation template ready, you could use that template and manually deploy it with the Cloudformation service. This would probably not be the most efficient option though. Preferably, you want to automatically deploy the template to a staging environment every time you push changes to the repository.

AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously.

CodePipeline allows you to build an automated workflow to enable your deployments to run successfully every time.

The staging pipeline will look like the diagram below.



**In the "Your environments" web browser tab.**

1. On the navigation bar, choose the **Services** button, and choose **CodePipeline**.
2. Choose **Create pipeline**
3. For **Pipeline name**, enter 
4. For **Service role**, choose **Existing service role**.
5. For **Role ARN**, select **codePipelineRole**.

 Important: This is an IAM role that was created for you for this lab. This role grants CodePipeline the IAM permissions it needs to connect to your CodeCommit repository and manage CloudFormation stacks.

1. Choose **Next**
2. For **Source provider**, select **AWS CodeCommit**.
3. For **Repository name**, select **web-application**.
4. For **Branch name**, select **dev**.
5. Choose **Next**

A build stage is useful if you need to compile your code, download dependencies, or perhaps run static code analysis. One potential use for this in your pipeline would be to verify that your CloudFormation template doesn't break any company security rules before being deployed. You could have a script that verifies no ports will be opened to the internet other than what is needed to access the load balancer. That's out of scope for this lab, but know that this is where you could place those types of actions.

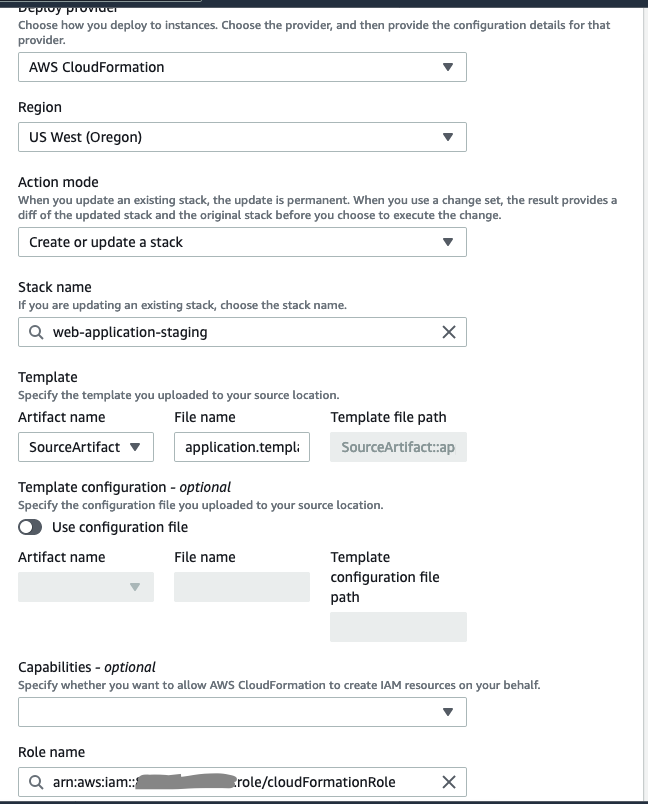
1. Choose **Skip build stage**
2. Choose **Skip**
3. For **Deploy provider**, choose **AWS CloudFormation**.
4. For **Region**, confirm that the name corresponds to the **PrimaryRegion** output to the left of the instructions.

 Note: If you need a list of Region to Region names, you can view them [here](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html#Concepts.RegionsAndAvailabilityZones.Regions)

1. For **Action mode**, select **Create or update a stack**.
2. For **Stack name**, enter 
3. For **Artifact name**, select **SourceArtifact**.

 Note: The SourceArtifact is just a copy of your code repository.

1. For **File name**, enter 
2. For **Role name**, select **cloudFormationRole**.



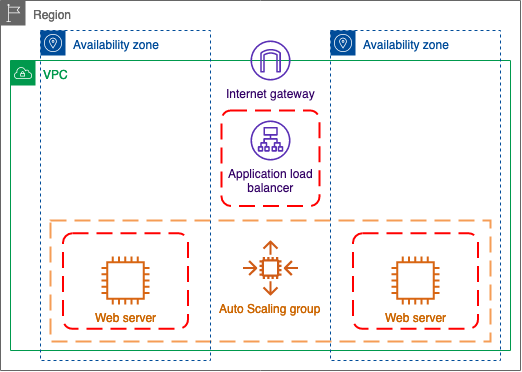
1. Choose **Next**
2. Choose **Create pipeline**

Now that you have created the pipeline, it will automatically run for the first time. In the source stage, the pipeline clones your CodeCommit repository and uploads it to an S3 bucket that holds artifacts for each run of the pipeline. Once that completes, the pipeline will move to the Deploy stage, that uses the values you provided to launch your CloudFormation stack.

While you wait for the pipeline to complete, continue to the next subtask to explore the template you are deploying.

Task 2.1: Explore the web application code

Just as a reminder, here is the diagram of your web application.



Open the **application.template** file in Cloud9.

Starting on **line 4** is the **Parameters** section. This is where you can create variables to use in your template. They can be specified with default values, as they are here, or the values can be passed to CloudFormation when you launch the stack.

On **line 18** is the start of the **Resources** section. This is where all of the AWS elements that you need to deploy are defined. Comments have been added to the template to explain what each group of resources are.

Starting with **line 19** the network resources that your application needs for connectivity are defined.

**Line 90**, begins the section where the two Amazon EC2 security groups are added. These are essentially the firewalls that describe what can access your resources and on which ports.

Beginning at **line 120**, the Application Load Balancer, Target Group and HTTP Listener to direct traffic between the two are included.

**Line 150** to the end of the **Resources** section has the Auto Scaling Group and Launch Template declarations. This will determine how your servers are built and how many of them will be in service. You'll make changes to the Launch Template later and it will be broken down more at that time.

Finally, on **line 266** you have the **Outputs** section. This is where you can make dynamically created values available for use by other entities. The only output specified in this template is the Application Load Balancer DNS name so that it can be used to view the web application.

Once the pipeline completes, take a look at what this template creates.

Task 2.2: Test your staging website

On the **"CodePipeline - AWS Developer Tools"** web browser tab, follow the steps below to view the stack in the AWS CloudFormation console.

1. On the **Deploy** action, choose Details
2. Confirm that the **web-application-staging** stack is selected.
3. Choose the **Resources** tab.

This tab shows you a list of every AWS resource that CloudFormation created for you in this stack.

1. Choose the **Outputs** tab.
2. Choose the **Value** of the **ApplicationURL** to open the website.

You should see a simple web page with the instance type, instance id, and region on it.

This website is hosted on 2 web servers with traffic balanced by an application load balancer. If you refresh the page a few times you should see the Instance Id change based on which web server handled your request.

You can close the web browser tab with your staging web application in it.

**Task 3: Extend continuous delivery to the production region**

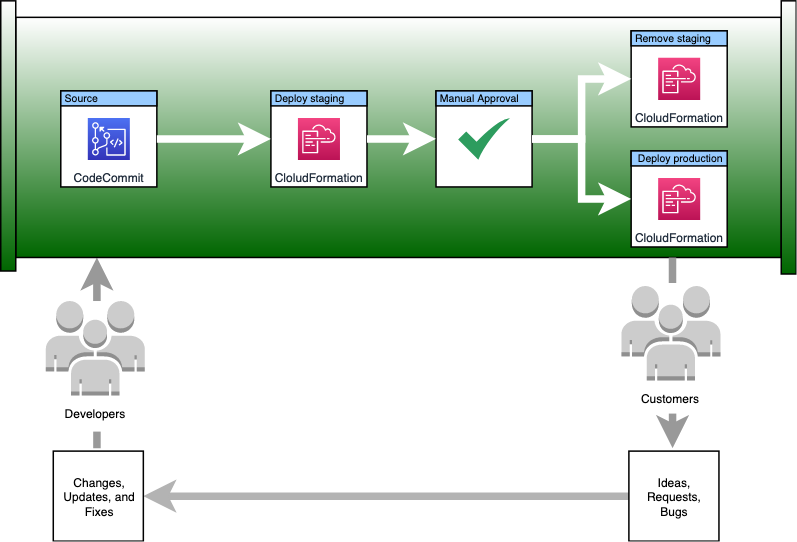
Deploying the stack to staging was successful. Now, enhance your pipeline by adding additional stages to it.

Add a manual approval stage to act as a gate to the next steps. If approved, the next stage will delete the staging environment and deploy a stack to production in parallel. If, however, the approval stage is rejected the pipeline will stop. You can address the reason for the rejection and push your changes again to request another approval. The CloudFormation stack for the staging environment will be updated to reflect your changes.

A lot of times, you have slight differences between your staging environment and the production environment. If you're not doing performance or load testing, why would you use production sized instances? This would add un-needed costs to your bill. To demonstrate the flexibility of CodePipeline, you'll deploy the production environment with slightly larger instance types in a completely separate AWS region.

As mentioned above, the staging resources in this deployment are meant to be temporary. When the stack has been approved, they should be removed from your AWS environment. This will save you money. You also have confidence that your staging deployment matches the last deployed version with only the committed changes that have been made in your code.

The pipeline you end up with will resemble the diagram below.

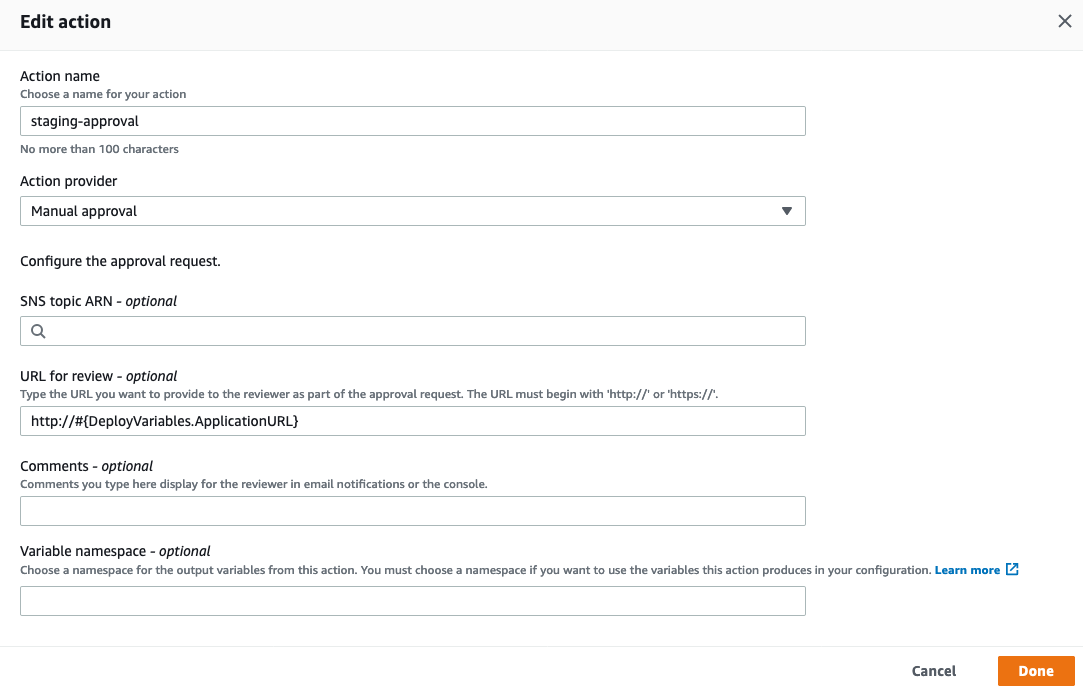


Task 3.1: Add a new stage to ask for approval before deploying to production

**On the "CodePipeline - AWS Developer Tools" web browser tab.**

1. Choose **Edit**
2. After the **Deploy** stage, choose **Add stage**
3. For **Stage name**, enter 
4. Choose **Add stage**
5. On the **Edit: approval-stage** card, choose **Add action group**
6. For **Action name**, enter 
7. For **Action provider**, choose **Manual approval**.
8. For **URL for review**, enter 

 Note: What you did was use the variable namespace that is exported from your deploy stage. The variable namespace for cloudformation makes the stack outputs available for you to use anywhere in your pipeline after the deploy stage. To use multiple namespaces in your pipeline, give each deploy stage a unique namespace name. Ultimately, in your pipeline, this allows you to provide a friendly URL to help with the approval decision.



1. Choose **Done**

Task 3.2: Add a new stage to deploy to the production region

1. On the **Edit: approval-stage** card, choose **Done**
2. After the **approval-stage** stage, choose **Add stage**
3. For **Stage name**, enter 
4. Choose **Add stage**
5. On the **Edit: deploy-to-production** card, choose **Add action group**
6. For **Action name**, enter 
7. For **Action provider**, under **Deploy**, select **AWS CloudFormation**.
8. For **Region**, select the region that corresponds to the **SecondaryRegion** output to the left of the instructions.

 Note: If you need a list of Region to Region names, you can view them [here](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html#Concepts.RegionsAndAvailabilityZones.Regions)

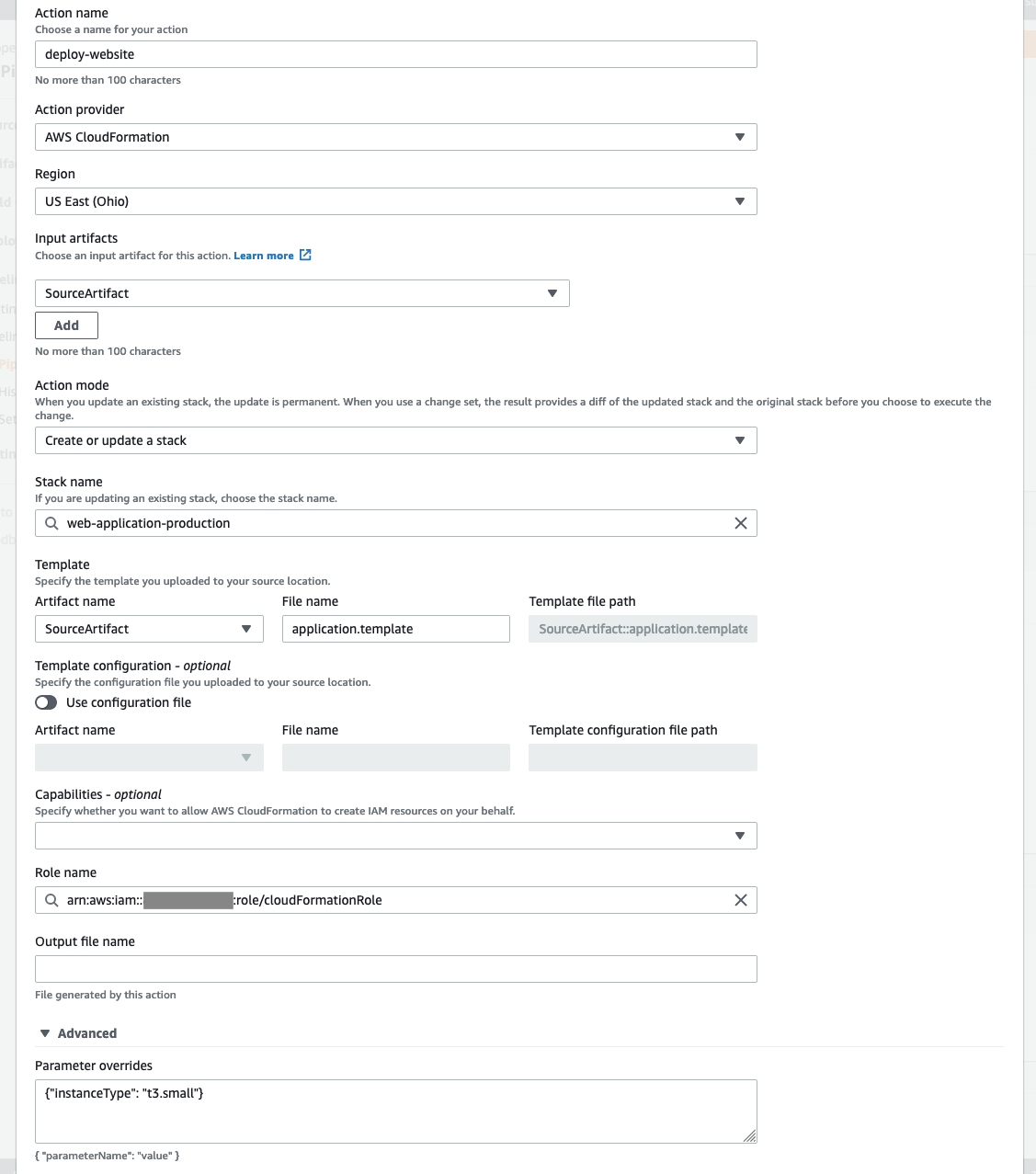
1. For **Input artifacts**, select **SourceArtifact**.
2. For **Action mode**, select **Create or update a stack**.
3. For **Stack name**, enter 
4. For **Artifact name**, select **SourceArtifact**.
5. For **File name**, enter 
6. For **Role name**, select **cloudFormationRole**.

A lot of times you may need to modify the parameters of a cloudformation stack when you deploy it to multiple environments. As an example, you need to change the production instance type to use t3.small instead of the default t3.micro.

There are two main ways to do this. One would be to create json file in your repository that has parameters defined. This file would be specified in the **Template configuration** section. In your pipeline you could have a parameters file for the staging deployment and a separate file for the production deployment.

Another method is to use **Parameter overrides** on the pipeline action itself. Since you only need to modify one setting on this pipeline, use this approach.

1. Expand **Advanced**.
2. For **Parameter overrides**, enter 



1. Choose **Done**

You could stop here. This will deploy the web application to your secondary region for production. But as stated above, you really don't need or want the staging web application after it has been approved.

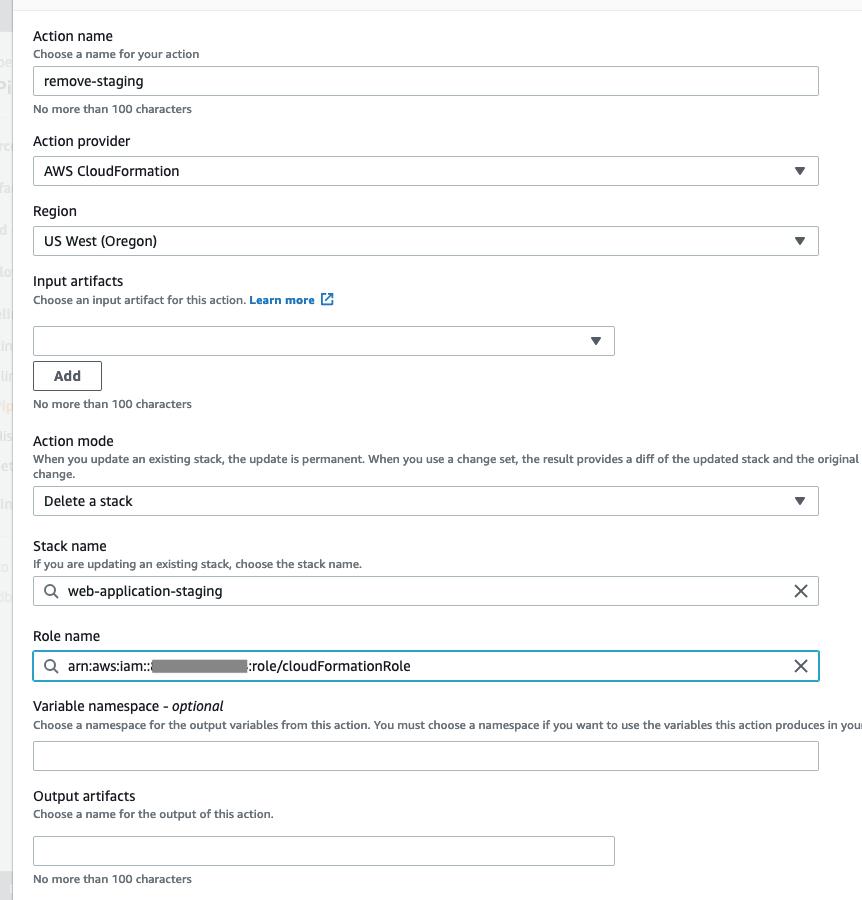
Task 3.3: Remove the staging environment while the production environment is being deployed

Add another action to delete the staging stack when your changes are deployed to production.

1. To add a parallel stage action, beside **deploy-website**, choose **Add action**.
2. For **Action name**, enter 
3. For **Action provider**, under **Deploy**, select **AWS CloudFormation**.
4. For **Region**, confirm that the name corresponds to the **PrimaryRegion** output to the left of the instructions.

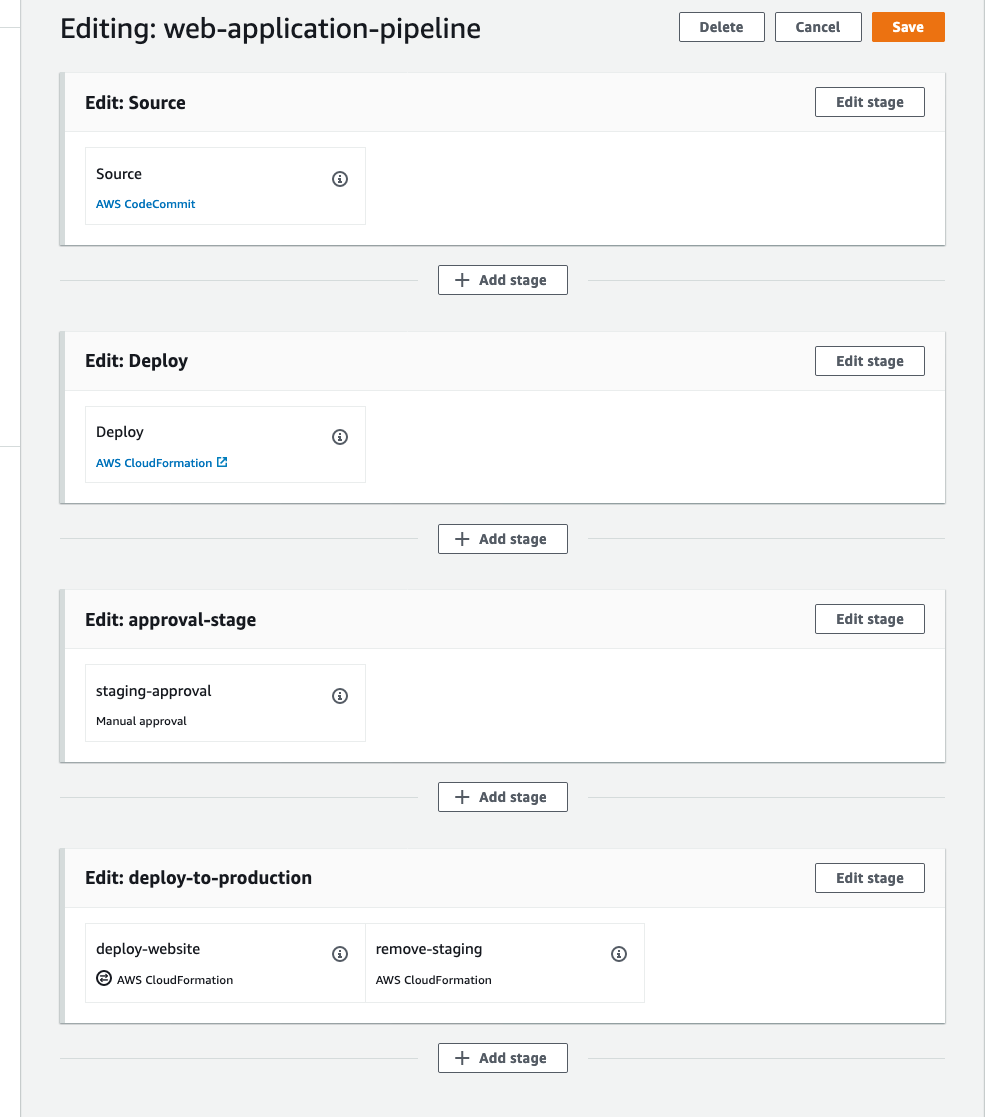
 Note: If you need a list of Region to Region names, you can view them [here](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html#Concepts.RegionsAndAvailabilityZones.Regions)

1. For **Action mode**, select **Delete a stack**.
2. For **Stack name**, enter 
3. For **Role name**, select **cloudFormationRole**.



1. Choose **Done**

Before saving, confirm that you pipeline matches the screenshot below.



1. At the top of the page, choose **Save**
2. Choose **Save**

**Task 4: Modify the application and test the entire CI/CD process**

**In the "Cloud9..." web browser tab.**

1. In the **Environment** window, expand the **web-application** directory and open the **application.template**.

Now that your pipeline is configured, you need to test a full deployment.

To test the pipeline, change the color scheme of the **index.html** page. This page is created on the web servers with CloudFormation initialization scripts. These scripts are defined on the launch template.

The cfn-init helper script reads template metadata from the **AWS::CloudFormation::Init** key and acts accordingly to:

* Install packages
* Write files to disk
* Enable/disable and start/stop services
* Run commands

In your template, the metadata for the cfn-init script was specified on the launch template resource beginning on **line 190**. The metadata installs httpd (a web server), ensures that it's running, and creates the index.html page.

Each time a new instance is launched by the autoscaling group, the EC2 UserData (scripts that are passed to the instance which only run at first launch) will use cfn-init to read this metadata and deploy this configuration. The UserData starts at **line 239** of your template and the line that calls cfn-init is at **line 251**.

Unfortunately, changing the metadata isn't recognized as a change that would require the launch template resource to be replaced. You'll get around that by adding a line to the change log section of the UserData for the launch template which will cause a new version to be created.

**Replacing your web servers without impacting service**

Your autoscaling group has an Update policy and a Creation policy specified for it beginning on **line 156**.

What you've specified with the Update policy is that any change, including the launch template version, will require a replacement of the autoscaling group.

The Creation policy will make sure that the instances launched by the replacement autoscaling group are healthy before the original one is removed. With the Creation policy you specified that it isn't actually healthy unless it receives 2 success signals within 5 minutes.

Where do those success signals come from? Cfn-signal, which runs as the last line of the UserData in your launch template on **line 264**. The exit code of cfn-init is sent to the autoscaling group, if cfn-init exited successfully, that would be a success signal. If both instances send success signals, then the replacement autoscaling group is marked as being successfully created and the original group can be removed.

This type of update should result in no downtime for your application when you approve updates to be deployed to production.

The **index.html** file content starts on **line 203**. Using the search functionality of the Cloud9 IDE replace all instances of the existing color with the new color.

1. On the **Menu** bar, select **Find**, and choose **Find**.
2. On the find bar, below the code editor, in the **Find** box, enter 
3. In the **Replace With** box, enter 
4. Choose **Replace All** (this button may be a different color)

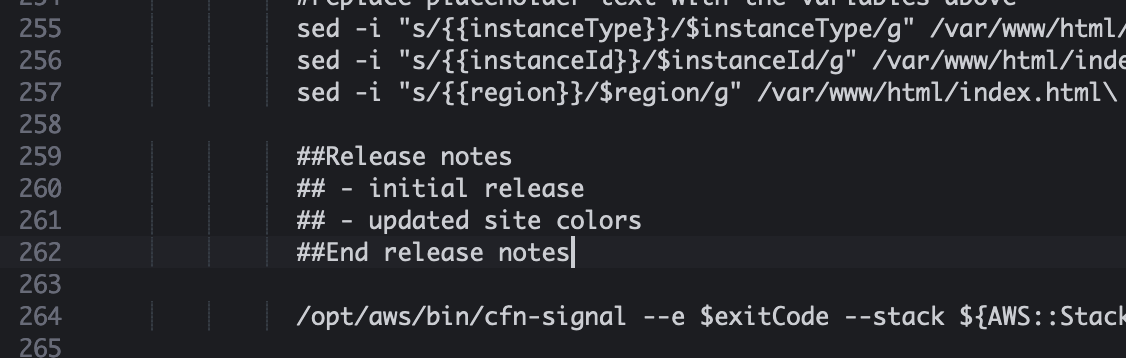
As described above, since a change in metadata alone will not cause the launch template resource to be replaced, Add a comment to the UserData property of the launch template resource.

1. On **line 261** of the **application.template**, add the following line.

## - updated site colors

content\_copy

 Critical: This line should match the indention of **line 260**. The UserData section should look like the following screenshot after you are done.



1. On the **Menu** bar, choose **File**, and choose **Save**.
2. To confirm that git sees your changes, run the following command in the **Terminal** window.

git status

content\_copy

You should see that the application.template has changes that need to be staged and committed.

1. In the **Terminal** window, enter the following commands to commit and push your changes to the CodeCommit repository.

# confirm that you are in the correct directory in the terminal

cd ~/environment/web-application

# stage and commit the changes locally with a meaningful message

git commit -am "changed the color scheme of the page"

# push the changes to CodeCommit

git push

content\_copy

**In the "CodePipeline - AWS Developer Tools" web browser tab.**

You should see the the pipeline running, if not refresh the page. The staging environment should take 2-3 minutes for the autoscaling group to be updated with the new launch template.

Wait for the **staging-approval** action to say  waiting for approval.

1. Choose **Review**
2. Choose the link displayed in the **URL for review** field to check your staging website.

This should have opened the staging website in a new web browser tab with the new color scheme. Leave the that tab open so that you can confirm it was decommissioned after your production website has been deployed.

1. On the **CodePipeline - AWS Developer Tools** web browser tab, choose **Approve**
2. On the **staging-approval** stage, choose Details.

This section shows, information about the approval including any comments that the approver may have entered. CodePipeline saves the results or each run, including approval information. You can view past approvals in the **History** section of the pipeline in the left navigation pane.

Once the **deploy-to-production** stage has Succeeded (2-3 minutes), reload the web browser tab for your staging website. You should receive a message stating that the site cannot be found. This confirms that your staging website is no longer running and you wouldn't be charged for those resources. You can close that web browser tab now.

1. On the **deploy-to-production** stage, choose the link for the **Pipeline execution ID**.
2. Under Actions, choose **deploy-website**.
3. On the **Output variables** card, Copy the **Value** of the **ApplicationURL** and open it in a new web browser tab.

You should see a web page that looks the one in staging. The differences are that the **Instance Type** now says **t3.small** and the **Region** should match the **SecondaryRegion** to the left of these instructions.

If you remember from your CloudFormation template, the **Instance Type** is specified by the parameter that you used a manual override for in the pipeline. All of these values are loaded from the EC2 instance metadata service.

**Conclusion**

This lab demonstrated how to automate CloudFormation with CodePipeline. Even though this lab deployed a simple website, you should be able to use this same pattern to automate the deployment of any AWS resources. You saw how each part of the pipeline can be used to add more functionality to your automated workflows.

 Congratulations! You now have successfully:

* Defined and provisioned your infrastructure as a code using AWS CloudFormation.
* Created a code repository and established continuous integration using AWS Developer Tools.
* Used the Cloud9 IDE to make code changes and integrate to a central repository.
* Built a release pipeline to automate continuous integration and continuous delivery (CI/CD) orchestration using AWS CodePipeline.
* Edited the CI/CD pipeline to add stages and actions, both sequential and parallel.
* Managed immutable deployments in a pipeline using runtime stage parameters.
* Controlled CI/CD process using manual approvals.

**End Lab**

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

1. Return to the AWS Management Console.
2. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.
3. Choose **End Lab**
4. Choose **OK**
5. (Optional):

* Select the applicable number of stars
* Type a comment
* Choose **Submit**
  + 1 star = Very dissatisfied
  + 2 stars = Dissatisfied
  + 3 stars = Neutral
  + 4 stars = Satisfied
  + 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

**Additional Resources**

* For more information about **DevOps and AWS**, see [DevOps on AWS](https://aws.amazon.com/devops/?nc2=h_ql_sol_use_dops).
* For more information about **AWS CloudFormation**, see [CloudFormation Documentation](https://aws.amazon.com/cloudformation/?nc2=h_ql_prod_mg_cf)
* For more information about **AWS CodePipeline**, see [CodePipeline Documentation](https://aws.amazon.com/codepipeline/?c=dv&sec=srv" \t "_blank)
* For more information about **AWS Cloud9**, see [Cloud9 Documentation](https://aws.amazon.com/cloud9/?c=dv&sec=srv)

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

Ready for more?

Here's another lab we think you'll like.