Multi-Region Infrastructure Deployment

Implementation Guide

Multi-Region Infrastructure Deployment: Implementation Guide

Table of Contents

Minimize latency when accessing objects in different geographic regions with the Multi-Region Infrastructure Deployment solution

April 2020 (last update (p. 20): July 2020)

This implementation guide discusses architectural considerations and configuration steps for deploying the Multi-Region Infrastructure Deployment solution in the Amazon Web Services (AWS) Cloud. It includes links to an AWS CloudFormation template that launches and configures the AWS services required to deploy this solution using AWS best practices for security and availability.

The guide is intended for IT infrastructure architects, administrators, and DevOps professionals who have practical experience architecting in the AWS Cloud.

Overview

The Amazon Web Services (AWS) Cloud makes it easy for developers to code, build, and deploy software. AWS has a continuous integration (CI) orchestration service called AWS CodePipeline and multiple deployment options to help developers quickly deploy and manage their software. Using AWS CodePipeline makes it easy to continuously build and test your AWS CloudFormation templates whenever you change them.

The Multi-Region Infrastructure Deployment solution supports controlled updates to infrastructure for applications that are deployed across primary and secondary Regions. This makes it easier to set up multi-region architectures and maintain consistency of workloads. This solution automatically detects the source code changes in the GitHub repository, deploys and validates a `stage` AWS CloudFormation stack, and deploys AWS CloudFormation stacks into a primary and secondary Region. This solution automatically provisions and configures AWS CodePipeline to automate the continuous integration/continuous delivery (CI/CD) pipeline for CloudFormation templates in the AWS Cloud.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of the date of publication, the cost for running this solution with default settings in the US East (N. Virginia) Region is approximately **$3.00 per month**. This estimate will vary based on the number of times a commit is pushed to the GitHub repository, AWS Lambda functions invoked, and Amazon Simple Storage Service (Amazon S3) storage fees. Prices are subject to change. For full details, see the pricing webpage for each AWS service you will be using in this solution.

Architecture overview

Deploying this solution builds the following environment in the AWS Cloud.

The solution's AWS CloudFormation template deploys a continuous integration/continuous delivery (CI/CD) pipeline in the primary Region using AWS CodePipeline.

AWS CodePipeline automatically detects changes in the GitHub repository and pulls the current source code from GitHub, encrypts and stores the code in the Amazon Simple Storage Service (Amazon S3) `artifact` bucket, and pushes it through a series of validation steps and deployment stages.

When you deploy the AWS CloudFormation template, you have the option to either retain or delete the `stage` AWS CloudFormation stack. The following sections detail both deployment options.

Option 1: Delete the stage CloudFormation stack

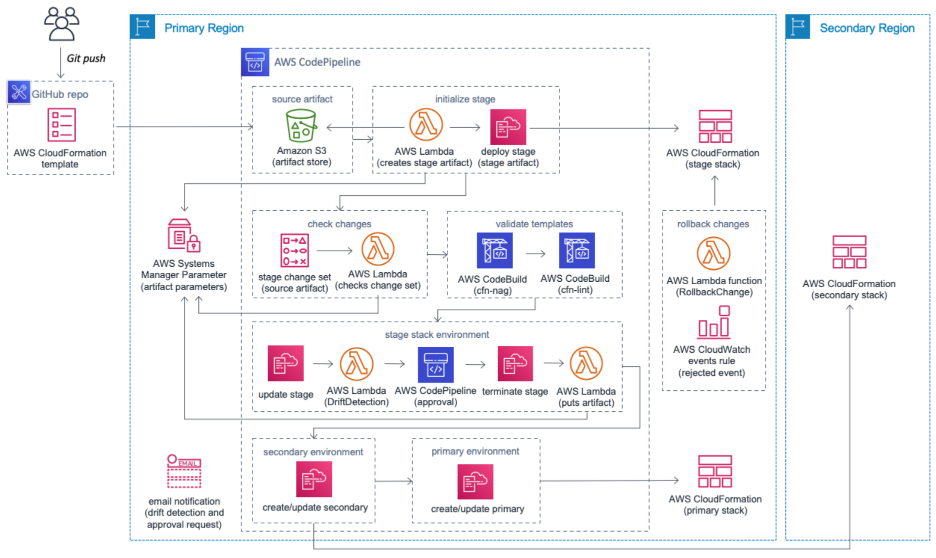
Primary Region
Secondary Region
Git push
GitHub repo
AWS CloudFormation template
AWS CodePipeline
source artifact
Amazon S3 (artifact store)
initialize stage
AWS Lambda (creates stage artifact)
deploy stage (stage artifact)
AWS CloudFormation (stage stack)
AWS Systems Manager Parameter (artifact parameters)
check changes
stage change set (source artifact)
AWS Lambda (checks change set)
validate templates
AWS CodeBuild (cfn-nag)
AWS CodeBuild (cfn-lint)
rollback changes
AWS Lambda function (RollbackChange)
AWS CloudWatch events rule (rejected event)
AWS CloudFormation (secondary stack)
stage stack environment
update stage
AWS Lambda (DriftDetection)
AWS CodePipeline (approval)
terminate stage
AWS Lambda (puts artifact)
email notification (drift detection and approval request)
secondary environment
create/update secondary
primary environment
create/update primary
AWS CloudFormation (primary stack)

Figure 1: Multi-Region Infrastructure Deployment architecture when the stage stack is deleted

The `stage` stack can be used in two ways: either as a temporary environment to support change validation or as a permanent environment to reflect the production codebase. The following workflow occurs when you choose to delete the `stage` stack after manual approval.

When a change is pushed to the GitHub repository, an AWS Lambda function checks the AWS Systems Manager parameter to determine if there is an existing AWS CodePipeline artifact for the `stage` AWS CloudFormation stack. If the artifact exists, the `stage` AWS CloudFormation stack uses the artifact to launch the initial `stage` AWS CloudFormation stack. If the artifact does not exist, the `stage` AWS CloudFormation stack uses the current source code to launch the initial `stage` AWS CloudFormation stack. Then, an AWS CloudFormation change set is created and an AWS Lambda function is invoked to check if the changes will result in a change to your production infrastructure. When initially deployed, there is no change, so the AWS Lambda function is skipped. The AWS CloudFormation template from your GitHub repository is then validated with security and style checks using cfn-nag and cfn-lint. Once validated, the `stage` AWS CloudFormation stack updates to the latest AWS CloudFormation template and an AWS Lambda function detects drifts on the primary and secondary stacks. Then, the template must be manually approved in the AWS CodePipeline console. Once approved, the `stage` AWS CloudFormation stack terminates and the `primary` AWS CloudFormation stack in the primary Region and the `secondary` AWS CloudFormation stack in the secondary Region deploy.

If the AWS CloudFormation template is rejected, an Amazon CloudWatch event triggers and an AWS Lambda function terminates the `stage` AWS CloudFormation stack.

Option 2: Retain the stage CloudFormation stack

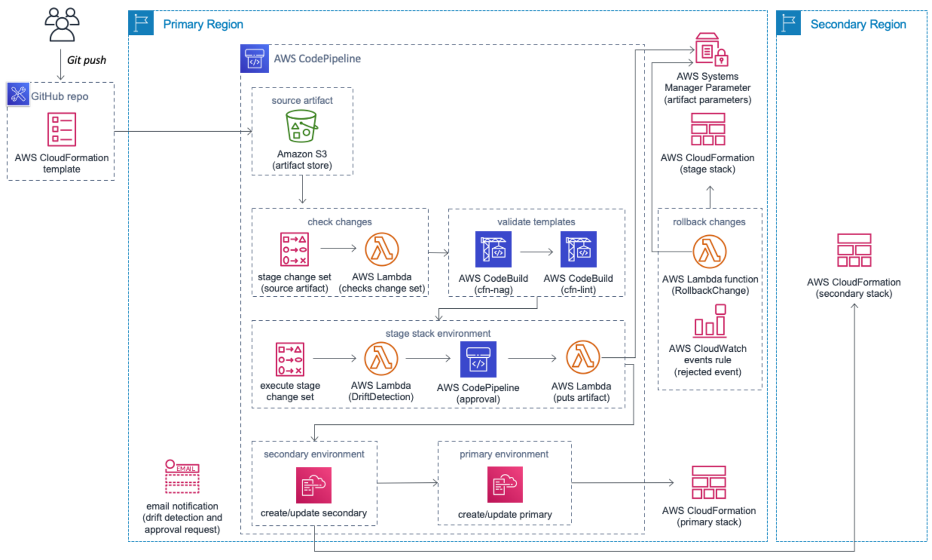
Git push
GitHub repo
AWS CloudFormation template
Primary Region
AWS CodePipeline
source artifact
Amazon S3 (artifact store)
check changes
stage change set (source artifact)
AWS Lambda (checks change set)
validate templates
AWS CodeBuild (cfn-nag)
AWS CodeBuild (cfn-lint)
stage stack environment
execute stage change set
AWS Lambda (DriftDetection)
AWS CodePipeline (approval)
AWS Lambda (puts artifact)
secondary environment
create/update secondary
primary environment
create/update primary
email notification (drift detection and approval request)
AWS Systems Manager Parameter (artifact parameters)
AWS CloudFormation (stage stack)
rollback changes
AWS Lambda function (RollbackChange)
AWS CloudWatch events rule (rejected event)
AWS CloudFormation (primary stack)
Secondary Region
AWS CloudFormation (secondary stack)

Figure 2: Multi-Region Infrastructure Deployment architecture when the stage stack is retained

The following workflow occurs when you choose to retain the `stage` AWS CloudFormation stack after manual approval.

When a change is pushed to the GitHub repository, an AWS CloudFormation change set is created and an AWS Lambda function checks if the changes will result in a change to the infrastructure. The AWS CloudFormation template from your GitHub repository is then validated with security and style checks using cfn-nag and cfn-lint. Once validated, the template deploys to a `stage` AWS CloudFormation stack in the primary Region, and an AWS Lambda function detects drifts to the `primary` and the `secondary` AWS CloudFormation stacks. The template must be manually approved in the AWS CodePipeline console. Once approved, the the `primary` AWS CloudFormation stack in the primary Region and the `secondary` AWS CloudFormation stack in the secondary Region deploy.

If the AWS CloudFormation template is rejected, an Amazon CloudWatch event triggers, and an AWS Lambda function reverts the `stage` AWS CloudFormation stack to the previous infrastructure. If there is no stage artifact information in the AWS Systems Manager parameters, the `stage` AWS CloudFormation stack terminates, since there are no `primary` nor `secondary` AWS CloudFormation stacks.

Solution components

AWS CloudFormation change sets

The Multi-Region Infrastructure Deployment solution uses AWS CloudFormation change sets to enable you to preview the `stage` stack in the primary AWS Region and review how the proposed changes might impact your running resources.

CI/CD pipeline

This solution deploys a continuous integration/continuous delivery (CI/CD) pipeline in the primary Region. The pipeline retrieves the CloudFormation template using the values provided in the solution's template parameters **GitHub Repo**, **GitHub Branch**, and **Template Path**.

GitHub repository

This solution uses the customer provided GitHub repository to store the CloudFormation template that defines the infrastructure for your application. The pipeline uses a GitHub Webhook to automatically detect changes to the `source` GitHub repository, which will trigger the pipeline to run again.

CloudFormation custom resource

This solution uses an AWS CloudFormation custom resource to create an Amazon Simple Storage Service (Amazon S3) bucket in the secondary Region. This enables artifacts to be deployed to the primary and secondary Region.

CloudFormation template filter

When a change is pushed to the GitHub repository the pipeline is automatically triggered. An AWS Lambda function filters the commits so that only changes to the customer's CloudFormation template are pushed through the pipeline. Without this filter, any non-CloudFormation template changes in a commit would cause the AWS CodePipeline to attempt to deploy unchanged AWS CloudFormation templates over existing infrastructure.

CloudFormation change set validation

This solution uses AWS CodeBuild projects to run cfn-lint and cfn-nag scans on the AWS CloudFormation template that is to be deployed. If any errors are found, the release is stopped and the error will be available in the AWS CodeBuild console.

CodePipeline deployment stages

An AWS CodePipeline deployment stage creates an AWS CloudFormation change set using the template that is pushed through the pipeline, then, a an AWS CloudFormation `execute` change set deploys the template changes into the primary or secondary Region.

The CodePipeline consists of three deployment stages:

- **Stage:** This environment is used to perform application integration tests to further validate the changes being deployed before moving to later stages of the pipeline. Manual approval is required before this solution can progress to later deployment stages.
- **Primary:** This environment represents your production infrastructure in the primary AWS Region.
- **Secondary:** This environment represents your production infrastructure in the secondary Region.

CloudFormation rollback change

When a manual approval request to deploy infrastructure changes to the `primary` and `secondary` AWS CloudFormation stacks is rejected, this solution automatically rolls back the changes on the `stage` AWS CloudFormation stack.

CloudFormation drift detection

Changes to AWS CloudFormation resources that are made outside of the AWS service can complicate stack update or deletion operations, resulting in drift. When any drifts occur with the AWS CloudFormation stack in either the primary or secondary regions, an AWS Lambda function sends an Amazon Simple Notification Service (Amazon SNS) notification. For information about drift detection, see Detecting unmanaged configuration changes to stacks and resources in the *AWS CloudFormation User Guide*.

Lambda functions

This solution uses the following AWS Lambda functions: `StageArtifactCreator`, `StageArtifactPutter`, and `RollbackChange`. The `StageArtifactCreator` Lambda function creates the AWS CodePipeline artifact for the `stage` AWS CloudFormation stack in an Amazon Simple Storage Service (Amazon S3) bucket. The `stage` AWS CloudFormation stack uses the artifact to deploy itself. The `StageArtifactPutter` Lambda function creates or updates AWS Systems Manager parameters, which include Amazon S3 paths and the `stage` AWS CloudFormation stack parameters. The `StageArtifactCreator` Lambda function uses the parameters to create the `stage` AWS CloudFormation stack so that it is the same as the `primary` AWS CloudFormation stack. The `RollbackChange` Lambda function uses the parameter to roll back the changes in the `stage` AWS CloudFormation stack.

Design considerations

CloudFormation execution role

When AWS CodePipeline executes an AWS CloudFormation change set commands, it assumes the `CloudFormationExecutionRole` that has the necessary permissions to provision and manage the solution’s AWS CloudFormation template.

When deploying the solution, the **CloudFormation Execution Policy** template parameter must specify an ARN for an AWS Identity and Access Management (IAM) policy with the necessary permissions. We recommend that you create a new IAM policy that contains the minimally-scoped set of permissions necessary to manage the deployed template.

CloudFormation template dependencies

If the template hosted in GitHub contains dependencies on nested stacks or AWS Lambda functions, the deployment package must be in the same AWS Region as the primary and secondary Regions. Therefore, nested stacks templates and AWS Lambda function source code need to be in the same Region because AWS CloudFormation can only refer to regional Amazon Simple Storage Service (Amazon S3) buckets.

GitHub source repository

AWS CodePipeline connects to your existing GitHub `source` repository. To integrate with GitHub, CodePipeline uses an OAuth application or a personal access token for your pipeline. CodePipeline creates a GitHub webhook that starts your pipeline when a change occurs in the repository.

AWS CloudFormation template

This solution uses AWS CloudFormation to automate the deployment of the Multi-Region Infrastructure Deployment solution in the AWS Cloud. It includes the following AWS CloudFormation template, which you can download before deployment:

View Template **multi-region-infrastructure-deployment.template:** Use this template to launch the Multi-Region Infrastructure Deployment solution and all associated components. The default configuration deploys AWS CodePipeline, AWS CodeBuild, AWS Lambda, Amazon Simple Storage Service (Amazon S3), and AWS Identity and Access Management (IAM), but you can also customize the template based on your specific needs.

Automated deployment

Before you launch the automated deployment, please review the architecture, configuration, and other considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy Multi-Region Infrastructure Deployment into your account.

Time to deploy: Approximately five minutes

Prerequisite

This solution uses AWS Secrets Manager to securely store the GitHub credentials. Before launching this solution, create a secret and store it in AWS Secrets Manager.

1. Sign in to the AWS Secrets Manager console.
2. Choose **Store a new secret**.
3. On the **Store a new secret** page, select Other **type of secrets**, and enter the following key and value pairs:

Key	Value	Description
github-username	*`<Your GitHub username>`*	The name of the GitHub user or company that owns the GitHub repository.
github-access-token	*`<Your GitHub access token>`*	The GitHub authentication token that allows AWS CodePipeline to perform operations on your GitHub repository.

4. Choose **Next**.
5. In **Step 2 Name and description**, enter a **Secret name** and choose **Next**.
6. In **Step 3 Configure automatic rotation**, choose **Next**.
7. In **Step 4 Review**, choose **Store**.

After storing a new secret for this solution, proceed to **Launch the stack**.

Launch the Stack

Note
You are responsible for the cost of the AWS services used while running this solution. See the Cost (p. 2) section for more details. For full details, see the pricing webpage for each AWS service you will be using in this solution.

1. Sign in to the AWS Management Console and click the button below to launch the `multi-region-infrastructure-deployment` AWS CloudFormation template.

Launch Solution

You can also download the template as a starting point for your own implementation.

2. The template launches in the US East (N. Virginia) Region by default. To launch this solution in a different AWS Region, use the Region selector in the console navigation bar.
3. On the **Create stack** page, verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack.
5. Under **Parameters**, review the parameters for the template, and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Notification Email Address	<Optional input>	Email address to receive notifications for change and drift detection.
Solution Secret Name	*`<Requires input>`*	The name for the AWS Secrets Manager secret.
GitHub Repo	*`<Requires input>`*	The name of the repository where source changes are detected.
GitHub Branch	*`<Requires input>`*	The name of the branch where source changes are detected.
Secondary Region	*`<Requires input>`*	The secondary AWS Region that this solution deploys AWS CloudFormation stacks to.
Template Path	*`<Requires input>`*	The root GitHub repo path to the CloudFormation template that this solution deploys.
CloudFormation Execution Policy	*`<Requires input>`*	The IAM policy that has the necessary permissions to manage the AWS CloudFormation template specified in the **TemplatePath** parameter.
Stage Parameters	<Optional input>	AWS CloudFormation template parameters passed to the `stage` stack. **Note** Enter the parameters as a JSON Object with a key for each parameter in the `stage` stack and the appropriate value.

Parameter	Default	Description
		Leave this parameter blank if the template in GitHub doesn't accept parameters.
Secondary Parameters	<Optional input>	AWS CloudFormation template parameters passed to the secondary stack. **Note** Enter the parameters as a JSON Object with a key for each parameter in the `secondary` stack and the appropriate value. Leave this blank if the template in GitHub doesn't accept parameters.
Primary Parameters	<Optional input>	AWS CloudFormation template parameters passed to the primary stack. **Note** Enter the parameters as a JSON Object with a key for each parameter in the `primary` stack and the appropriate value. Leave this blank if the template in GitHub doesn't accept parameters.
Delete Stage Stack	`Yes`	Specifies whether or not to terminate the stage environment. If set to `Yes`, the stage environment terminates after manual approval to deploy the infrastructure changes to the primary and the secondary environments. If set to `No`, then this solution will keep the stage environment after the manual approval.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should see a status of **CREATE_COMPLETE** in approximately five minutes.

Note

In addition to the primary AWS Lambda functions, this solution includes the `custom-resource` Lambda function, which runs only during initial configuration or when resources are updated or deleted. When running this solution, the `custom-resource` function is inactive. However, do not delete this function as it is necessary to manage associated resources.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model can reduce your operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the services operate. For more information about security on AWS, visit the AWS Security Center.

IAM Roles

AWS Identity and Access Management (IAM) roles enable customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grants the AWS Lambda function access to the other AWS services used in this solution.

AWS Systems Manager parameters

This solution uses AWS Systems Manager parameters to store the AWS CodePipeline stage environment artifacts. All parameters are secure strings.

AWS Secrets Manager secret

This solution requires you to store your GitHub username and GitHub access token in AWS Secrets Manager. This solution retrieves the GitHub credential to pull the source code from your GitHub repository.

Note
Since this solution cannot rotate the GitHub access token automatically, the customer is responsible for rotating the GitHub access token.

Additional Resources

- AWS Lambda
- Amazon Simple Storage Service
- Amazon CloudWatch
- AWS CodePipeline
- AWS Secrets Manager
- AWS CodeBuild
- AWS Identity and Access Management
- AWS CloudFormation
- AWS Systems Manager

Appendix A: Disabling cfn-nag and cfn-lint

This solution runs cfn-nag and cfn-lint to scan the AWS CloudFormation template that is to be deployed. If errors are found, the release is stopped. For information about cfn-nag, see the cfn_nag GitHub repository. For information about cfn-lint, see the cfn-python-lint GitHub repository.

You have the option to disable cfn-nag and cfn-lint while deploying new changes to AWS CloudFormation stacks by using the following procedure.

1. Sign in to the AWS CodeBuild console.
2. On the **Build projects** page, select `CfnLintBuildProject-<hash>` or `CfnNagBuildProject-<hash>`.
3. Choose **Edit** and then choose **Environment**.
4. Expand **Additional configuration** and change the `RUNNING_CFN_LINT` or `RUNNING_CFN_NAG` value to **No**.
5. Choose **Update environment**.

Appendix B: Uninstall the solution

You can uninstall the Multi-Region Infrastructure Deployment solution using the AWS Management Console or the AWS Command Line Interface (AWS CLI). However, the AWS Systems Manager parameters and AWS Secrets Manager secret, Amazon Simple Storage Service (Amazon S3) buckets, and AWS CloudFormation stacks created by this solution must be manually deleted.

Detaching managed policies from CodeBuildServiceRole

AWS CodeBuild creates managed policies in `CodeBuildServiceRole` when you enable or disable cfn-nag or cfn-lint manually (see Appendix A (p. 15)). These managed policies will block your ability to delete the solution. As a result, before you can delete the solution, you must use the following procedure to first detach the managed policies in `CodeBuildServiceRole`.

1. Sign in to the AWS Identity and Access Management (IAM) console.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, search for *`<stack-name>`*`CodeBuildServiceRole` to filter the list of roles. You need to detach all the `CodeBuildServiceRole` managed policies.
4. Select a *`<stack-name>`*`-CodeBuildServiceRole` managed policy, and on the **Summary** page, choose the **Detach** icon (**x**).
5. Return to the Roles page and continue to select a *`<stack-name>`*`-CodeBuildServiceRole` managed policy until you have detached all of them.

Deleting AWS CloudFormation stacks created by this solution

Before you can delete the solution's AWS CloudFormation stack, you must delete the `stage`, `primary`, and `secondary` stacks first. The `stage`, `primary`, and `secondary` stacks are created by the solution's IAM role. If you delete this solution before deleting these stacks, you will receive an invalid role error. If you accidentally deleted the solution before removing the `stage`, `primary`, and `secondary` stacks, see How do I delete an AWS CloudFormation stack that's stuck in the DELETE_FAILED status? in the AWS Premium Support Knowledge Center.

1. Sign in to the AWS CloudFormation console
2. On the **Stacks** page, select the name of your stage environment (or stack).
3. Choose **Delete**.

Repeat these steps until you have deleted all the environments (stacks). Next, you can delete the solution's AWS CloudFormation stack using either the AWS Management Console or AWS CLI.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.

2. On the **Stacks** page, select the solution stack.
3. Choose **Delete**.

Using AWS CLI

Determine whether AWS CLI is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <cloudformation-stack-name>
```

Deleting the Amazon S3 Buckets

This solution is configured to retain the Amazon S3 buckets if you decide to delete the AWS CloudFormation stack to prevent against accidental data loss. After uninstalling the solution, you can manually delete these buckets if you do not need to retain the data. Use the following procedure to delete the Amazon S3 buckets.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. Locate the *`<stack-name>`* S3 buckets.
4. Select one of the S3 buckets and choose **Delete**.

Repeat the steps until you have deleted all the *`<stack-name>`* S3 buckets.

Alternatively, you can configure the AWS CloudFormation template to delete the Amazon S3 buckets automatically. Prior to deleting the stack, change the deletion behavior in the AWS CloudFormation DeletionPolicy Attribute.

Deleting the AWS Systems Manager parameters

Use the following procedure to delete the AWS Systems Manager parameters.

1. Sign in to the AWS Systems Manager console.
2. Choose **Parameter Store** from the left navigation pane.
3. Select the *`<stack-name>`*`/parameter-artifact` and *`<stack-name>`*`/source-artifact` parameters and choose **Delete**.

Deleting the AWS Secrets Manager secret

Use the following procedure to delete the AWS Secrets Manager parameters.

1. Sign in to the AWS Secrets Manager console.
2. Select the **Secret** you created for this solution.
3. Choose **Actions** and choose **Delete secret**.

Appendix C: Collection of operational metrics

This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When enabled, the following information is collected and sent to AWS:

- **Solution ID:** The AWS solution identifier
- **Version:** The version of the deployed solution
- **Timestamp:** The UTC formatted timestamp of when the event occured
- **Unique ID (UUID):** Randomly generated, unique identifier for each solution deployment
- **Success:** Value of true, if the solution was able to validate a CloudFormation change set and push it to later stages of the pipeline, if unsuccessful the value will be false.

Note that AWS will own the data gathered via this survey. Data collection will be subject to the AWS Privacy Policy. To opt out of this feature, modify the AWS CloudFormation template mapping section as follows:

```
Solution:
      Metrics:
         SendAnonymousData: Yes
```

to

```
Solution:
    Metrics:
       SendAnonymousData: No
```

Source code

You can visit our GitHub repository to download the templates and scripts for this solution, and to share your customizations with others.

Document revisions

Date	Change
April 2020	Initial release
July 2020	Updated the architecture diagram; renamed pre-production environment to stage; added AWS CloudFormation drift detection and an AWS CloudFormation template parameter to delete the stage environment; replaced the AWS CloudFormation template parameter to an AWS Secrets Manager secret for the GitHub credential; for additional information about changes to v1.1, see the changelog file in the GitHub repository

Notices