



CI/CD Configurations for Backend Applications using Cross account

Agenda:

CodeBuild:

AWS code build is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With code build, you don't need to provision, manage, and scale your own build servers. code build scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue. You can get started quickly by using prepackaged build environments, or you can create custom build environments that use your own build tools.

CodePipeline:

AWS code pipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. code pipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. This enables you to rapidly and reliably deliver features and updates. You can easily integrate AWS code pipeline with third-party services such as git hub or with your own custom plugin.

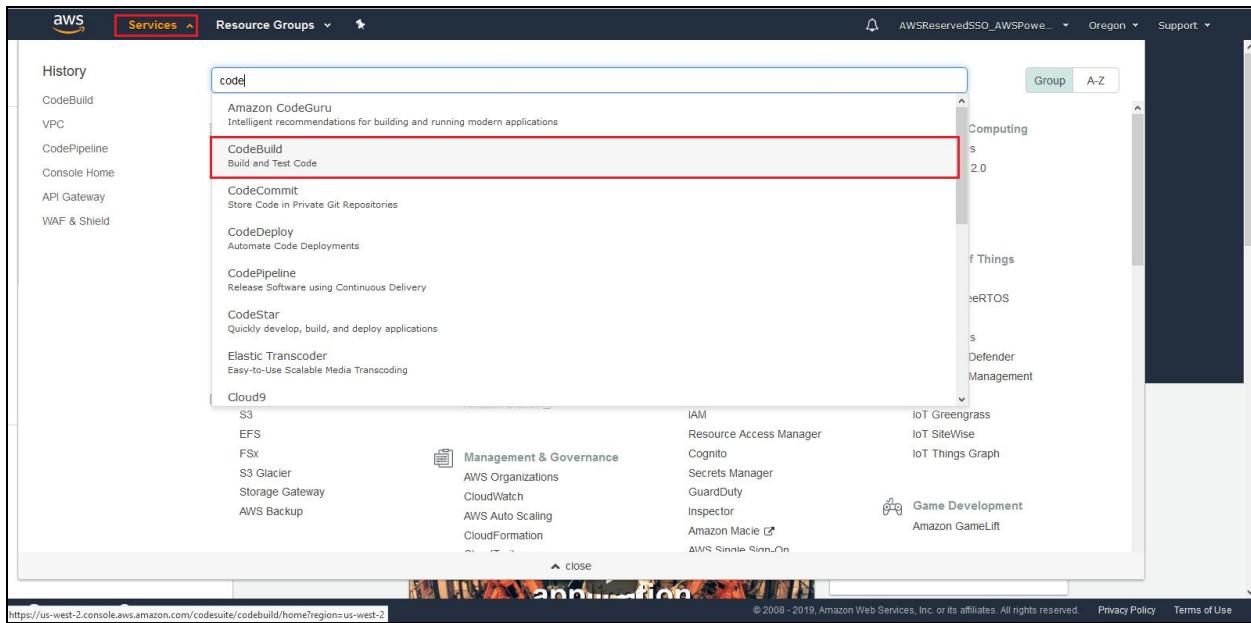
Serverless Deployment:

The serverless framework was designed to provision your AWS lambda functions, events and infrastructure resources safely and quickly.

The serverless framework translates all syntax in **serverless.yml** to a single AWS cloudformation template. By depending on cloudformation for deployments, users of the serverless framework get the safety and reliability of cloudformation.

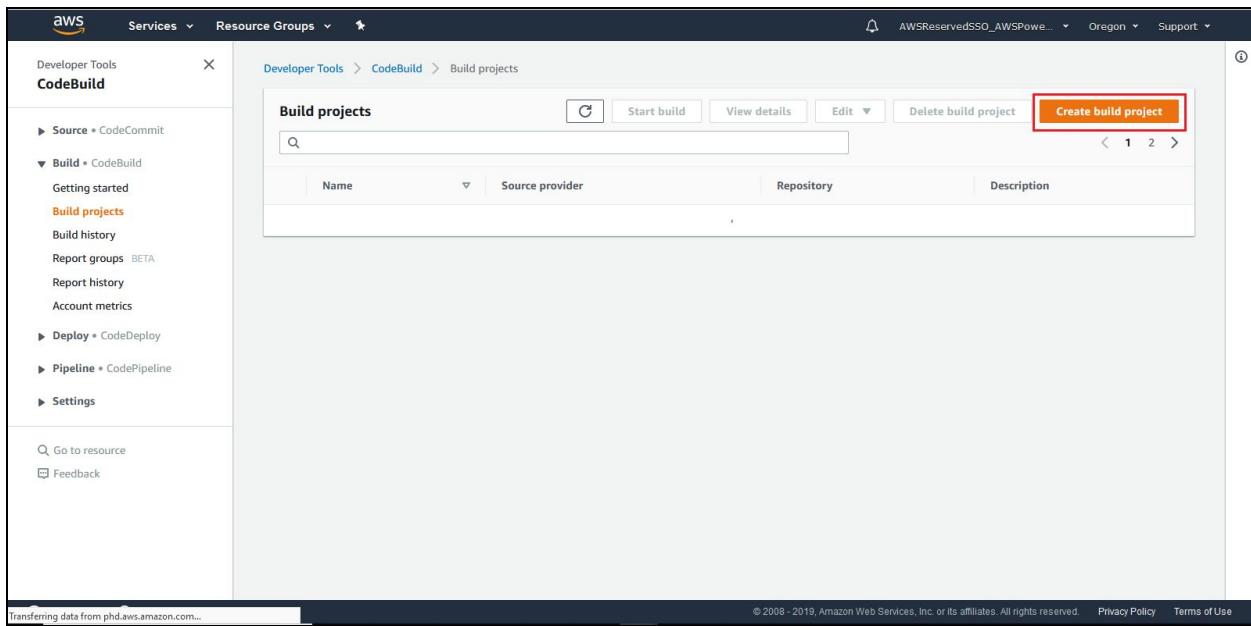
By the end of the document, we learn how to create a CI-CD pipeline for NodeJs and ASP dotnet core applications using cross account.

First we need to login into the AWS console and goto **code build** service.



The screenshot shows the AWS console search interface. A search bar at the top contains the text "code". Below the search bar, a list of services is displayed, with "CodeBuild" being the first item and highlighted with a red box. Other listed services include Amazon CodeGuru, CodeCommit, CodeDeploy, CodePipeline, CodeStar, and Elastic Transcoder. To the right of the search results, there is a sidebar with various AWS service categories like Computing, IoT, and Machine Learning, each with a small icon and some text.

Goto build projects and click on **create build project** to create a new build project for integration.



The screenshot shows the AWS CodeBuild "Build projects" page. On the left, there is a navigation sidebar with options like "Source", "Build", "Deploy", "Pipeline", and "Settings", with "Build" currently selected. The main area is titled "Build projects" and contains a table with columns for "Name", "Source provider", "Repository", and "Description". At the top of this section, there is a search bar and several buttons: "Start build", "View details", "Edit", "Delete build project", and a prominent "Create build project" button, which is also highlighted with a red box.

Enter the name of the project.

The screenshot shows the 'Create build project' page in the AWS Management Console. The 'Project configuration' section is visible, containing fields for 'Project name' (set to 'Americold-BillingApp-Backend-NodeJs-QA'), 'Description - optional' (empty), and 'Build badge - optional' (unchecked). Below these are sections for 'Source' and 'Additional configuration tags'. The 'Source' section includes a 'Source provider' dropdown set to 'AWS CodeCommit' and a 'Repository' input field containing 'Americold.BillingApp.Backend.NodeJs'. Other source provider options like GitHub and Bitbucket are listed below. The bottom of the page shows standard AWS navigation links for Feedback, English (US), and legal notices.

Under the source section, select the source provider as **AWS code commit**, and select the repository and branch as shown below.

The screenshot shows the 'Source' configuration page. It details the selection of 'Source provider' as 'AWS CodeCommit' and 'Repository' as 'Americold.BillingApp.Backend.NodeJs'. Under 'Reference type', 'Branch' is selected. The 'Branch' dropdown shows 'QA' as the chosen branch. A 'Commit ID - optional' field is present but empty. The 'Source version info' section shows a commit hash '28e12283 Squashed commit of the following'. Below this are sections for 'Additional configuration' and 'Git clone depth, Git submodules'.

Under environment section we took the managed image that is provided by AWS, operating system as Ubuntu, runtime as standard, image as standard 2.0 version, image version as always latest, environment type as linux and created new service role for the code build .

Environment

Environment image

- Managed image Use an image managed by AWS CodeBuild
- Custom image Specify a Docker image

Operating system

Ubuntu

The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See [Docker Images Provided by CodeBuild](#) for details.

Runtime(s)

Standard

Image

aws/codebuild/standard:2.0

Image version

Always use the latest image for this runtime version

Environment type

Linux

Privileged

Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Image

aws/codebuild/standard:2.0

Image version

Always use the latest image for this runtime version

Environment type

Linux

Privileged

Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Service role

- New service role Create a service role in your account
- Existing service role Choose an existing service role from your account

Role name

codebuild-Americold-BillingApp-Backend-NodeJs-QA-service-role

Type your service role name

Additional configuration

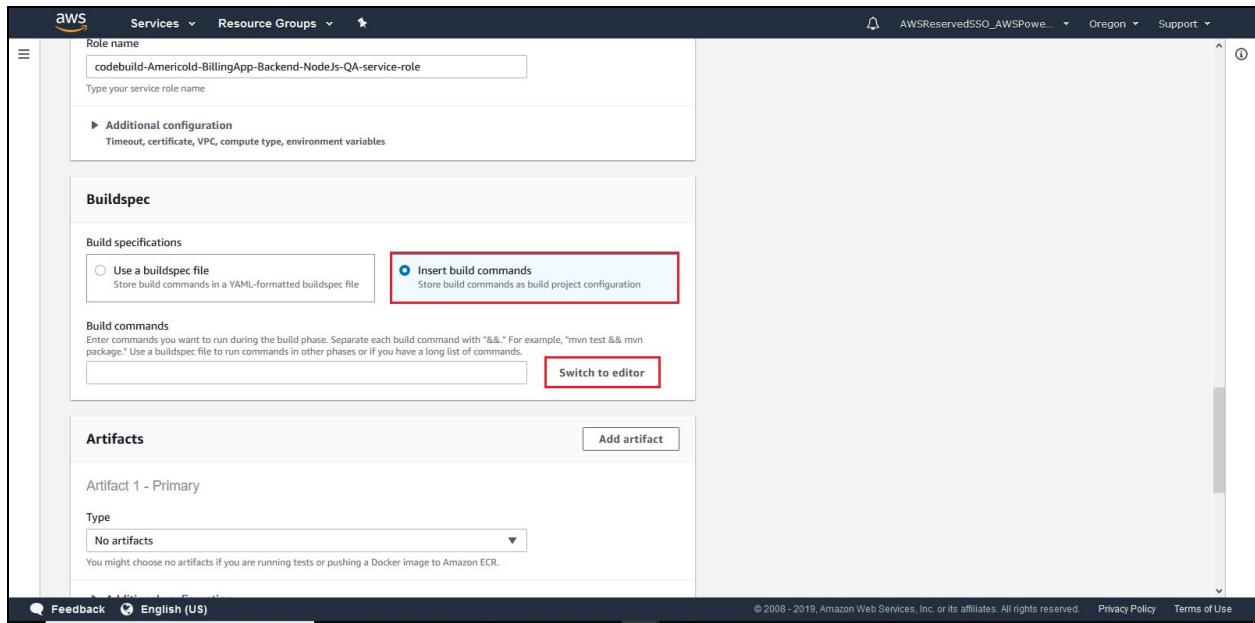
Timeout, certificate, VPC, compute type, environment variables

Buildspec

Build specifications

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Under the build spec section, choose an option as **insert build commands** to edit the **buildspec.yml** file from console. buildspec.yml file has a set of commands for integration and deployment.



Configure the respective stages as per requirement in **buildspec.yml** file as shown below.

```

version: 0.2

env:
  variables:
    - key: "value"
    - key: "value"
    - key: "value"
    - key: "value"
    - key: "value"

git-credential-helper: yes

phases:
  install:
    #if you use the Ubuntu standard image 2.0 or later, you must specify runtime-versions.
    #if you specify runtime-versions and use an image other than Ubuntu standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 10
      # name: version
      # name: version
      # command
      # - command
      # - command
      pre_build:
        commands:
          - apt install jq -y
          - unset AWS_SESSION_TOKEN
          - export AWS_REGION=us-west-2
          - temp_ole=$aws sts assume-role --role-arn "arn:aws:iam::017586350412:role/Americold_CrossAccount_Role" --role-session-name "QA"
          - export AWS_ACCESS_KEY_ID=$(echo $temp_ole | jq .Credentials.AccessKeyId | xargs)
          - export AWS_SECRET_ACCESS_KEY=$(echo $temp_ole | jq .Credentials.SecretAccessKey | xargs)
          - export AWS_SESSION_TOKEN=$(echo $temp_ole | jq .Credentials.SessionToken | xargs)
          # - command
          # - command
      build:
        commands:
          - cd Inventory_Consolidation
          - npm install
          - npm install -g serverless
          # - command
          # - command
      post_build:
        commands:
          - serverless deploy --stage qa
          # - command
          # - command
  artifacts:
    #artifacts:

```

The screenshot shows the AWS CodeBuild console under the 'Build project' section. The left sidebar lists 'Source', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The 'Build' section is expanded, showing 'Build project' with 'Build spec' selected. The main area displays the contents of the buildspec.yml file, which defines a build phase named 'install' with specific commands for setting environment variables, installing dependencies, and deploying using Serverless. The 'Edit' button is located at the top right of the code editor area.

Below is the step by step explanation for the above script:

Install stage:

- **runtime-versions:**

nodejs : 10: It defines the runtime environment and version for the application

Pre_build stage:

- **apt install jq -y:** Install node modules using the following command
- **unset AWS_SESSION_TOKEN:** Clear the AWS session token
- **export AWS_REGION=us-west-2:** Set the AWS region
- **temp_role=\$(aws sts assume-role --role-arn "arn:aws:iam::017586350412:role/Americold_CrossAccount_Role" --role-session-name "QA") :** Create temporary AWS IAM role and store in the variable as shown above
- **export AWS_ACCESS_KEY_ID=\$(echo \$temp_role | jq .Credentials.AccessKeyId | xargs)**
export AWS_SECRET_ACCESS_KEY=\$(echo \$temp_role | jq .Credentials.SecretAccessKey | xargs)
export AWS_SESSION_TOKEN=\$(echo \$temp_role | jq .Credentials.SessionToken | xargs)
Set the AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_SESSION_TOKEN variables with their respective values using jq tool as shown above

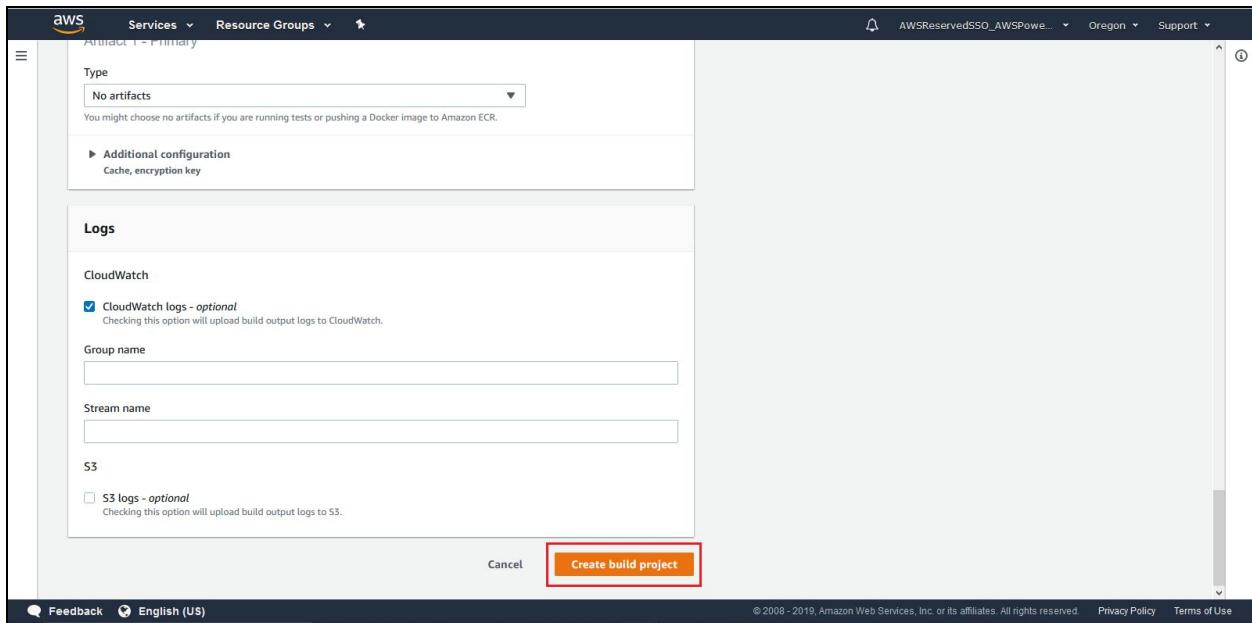
Build Stage:

- **cd Inventory_consolidation:** Go to the folder called Inventory_consolidation to install the necessary dependencies for integration.
- **npm install:** Install node modules using the following command
- **npm install -g serverless:** Install serverless module using below command

Post-Build stage:

- **serverless deploy --stage qa:** To deploy an application using serverless.yml template file we will run the following command

We are enabling cloud watch logs to check the build logs. Once all the configurations were done, click on **create build project**.



The new build project has been created and the dashboard is shown as below.

The screenshot shows the AWS CodeBuild project dashboard for 'Americold-BillingApp-Backend-NodeJs-QA'. The left sidebar navigation includes 'Source', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The main content area displays the 'Configuration' tab, which lists the source provider as 'AWS CodeCommit' and the primary repository as 'Americold.BillingApp.Backend.NodeJs'. Below this is the 'Build history' tab, which shows a single completed build run. The build details include the status 'Success', build number '1', source version 'v1', submitter 'aws-codedeploy', duration '1 minute 2 seconds', and completed time '1 minute ago'.

Goto details and goto **environment** section in build details.

The screenshot shows the AWS CodeBuild project configuration page. The left sidebar has a 'Build project' section selected. The main area shows the 'Configuration' tab, which includes fields for Source provider (AWS CodeCommit), Primary repository (Americold.BillingApp.Backend.NodeJs), Artifacts upload location (disabled), and Build badge (disabled). Below this is the 'Build details' tab, which is highlighted with a red border. The 'Project configuration' tab shows the project name (Americold-BillingApp-Backend-NodeJs-QA) and ARN (arn:aws:codebuild:us-west-2:646652670602:project/Americold-BillingApp-Backend-NodeJs-QA). The 'Source' tab is also visible at the bottom.

Click on service role which is new service role created while configuring code build. It will redirect us to **IAM console**.

The screenshot shows the 'Environment' section of the AWS CodeBuild project configuration. The 'Service role' field is highlighted with a red box and contains the value 'arn:aws:iam::646652670602:role/IAM_CodeBuild'. Other environment settings include the Image (aws/codebuild/standard:2.0), Environment type (Linux), Compute (3 GB memory, 2 vCPUs), and Privileged (False). The 'Buildspec' section at the bottom contains the following YAML code:

```
version: 0.2
env:
  variables:
    # key: "value"
    # key: "value"
    # parameter-store:
      # key: "value"
```

The IAM role dashboard will be as shown below and we need to add the required policies to access the AWS services for application deployment through code build.

The screenshot shows the AWS IAM Role Summary page for the role 'IAM_CodeBuild'. The left sidebar shows the IAM navigation menu with 'Roles' selected. The main area displays the role's details: Role ARN (arn:aws:iam:646632670602:role/IAM_CodeBuild), Role description (Allows CodeBuild to call AWS services on your behalf.), Instance Profile ARNs (/), Path (/), Creation time (2019-10-24 00:24 UTC+0530), Last activity (2020-01-08 17:10 UTC+0530 (Today)), and Maximum CLI/API session duration (1 hour). Below these details is a 'Permissions' tab, which is currently active, showing 'Permissions policies (10 policies applied)'. Under this tab, the 'Attach policies' button is highlighted with a red box. A table lists the attached policies: AWSCodeCommitFullAccess (AWS managed policy), AmazonS3FullAccess (AWS managed policy), and CloudFrontFullAccess (AWS managed policy). There are also entries for CloudWatchLogsFullAccess, AWSCodePipelineFullAccess, Americold_CrossAccount_Policy_UAT, Americold_CrossAccount_Policy_QA, CodeBuildBasePolicy-Americold_BillingApp_Backend-NodeJs-UAT-us-west-2, and CodeBuildBasePolicy-Americold-BillingApp-Backend-NodeJs-UAT-us-west-2, all listed as Managed policy.

Click on **Attach policies** to add the policies for the IAM Role.

This screenshot is identical to the one above, showing the AWS IAM Role Summary page for the role 'IAM_CodeBuild'. The left sidebar shows the IAM navigation menu with 'Roles' selected. The main area displays the role's details and the 'Permissions' tab, which is active. The 'Attach policies' button is highlighted with a red box. The table under 'Permissions policies (10 policies applied)' shows the same list of attached policies as the previous screenshot, including AWSCodeCommitFullAccess, AmazonS3FullAccess, CloudFrontFullAccess, CloudWatchLogsFullAccess, AWSCodePipelineFullAccess, Americold_CrossAccount_Policy_UAT, Americold_CrossAccount_Policy_QA, CodeBuildBasePolicy-Americold_BillingApp_Backend-NodeJs-UAT-us-west-2, and CodeBuildBasePolicy-Americold-BillingApp-Backend-NodeJs-UAT-us-west-2, all categorized as Managed policy.

Search for the required policy and select the check boxes and click on Attach Policies.

The screenshot shows the 'Add permissions to IAM_CodeBuild' dialog. At the top, there's a search bar labeled 'Search' with a red box around it. Below the search bar is a table listing various AWS policies. The table has columns for 'Policy name', 'Type', and 'Used as'. The 'Attach policy' button at the bottom right is also highlighted with a red box.

Policy name	Type	Used as
AdministratorAccess	Job function	Permissions policy (9)
AlexaForBusinessDeviceSetup	AWS managed	None
AlexaForBusinessFullAccess	AWS managed	None
AlexaForBusinessGatewayExecution	AWS managed	None
AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	None
AlexaForBusinessReadOnlyAccess	AWS managed	None
AmazonAPIGatewayAdministrator	AWS managed	Permissions policy (5)
AmazonAPIGatewayInvokeFullAccess	AWS managed	None
AmazonAPIGatewayPushToCloudWatchLogs	AWS managed	Permissions policy (1)
AmazonAppStreamFullAccess	AWS managed	None
AmazonAppStreamReadOnlyAccess	AWS managed	None
AmazonAppStreamServiceAccess	AWS managed	None
AmazonAthenaFullAccess	AWS managed	None
AmazonCloudWatchLogsFullAccess	AWS managed	None

Here we have attached the policies for **Code commit, S3, cloud front, cloud watch logs, code pipeline and Cross account policy**.

The screenshot shows the 'Summary' page for the 'IAM_CodeBuild' role. On the left, there's a navigation menu with 'Roles' selected. The main area displays the role's ARN, description, and creation details. Below this is a 'Permissions' tab showing a list of attached policies. The list is highlighted with a red box. The policies listed include AWSCodeCommitFullAccess, AmazonS3FullAccess, CloudFrontFullAccess, CloudWatchLogsFullAccess, AWSCodePipelineFullAccess, Americold_CrossAccount_Policy_UAT, Americold_CrossAccount_Policy_QA, CodeBuildBasePolicy-Americold_BillingApp_UI-us-west-2, and CodeBuildBasePolicy-Americold-BillingApp-Backend-NodeJs-UAT-us-west-2.

Here we have attached the QA environment account assume role policy to the code build role to provide the necessary permissions to deploy the code into QA environment.

Policy name	Policy type
AWSCodeCommitFullAccess	AWS managed policy
AmazonS3FullAccess	AWS managed policy
CloudFrontFullAccess	AWS managed policy
CloudWatchLogsFullAccess	AWS managed policy
AWSCodePipelineFullAccess	AWS managed policy
Americold_CrossAccount_Policy_UAT	Managed policy
Americold_CrossAccount_Policy_QA	Managed policy

```

1+ {
2+   "Version": "2012-10-17",
3+   "Statement": [
4+     {
5+       "Effect": "Allow",
6+       "Action": "sts:AssumeRole",
7+       "Resource": "arn:aws:iam::017586350412:role/Americold_CrossAccount_Role"
8+     }
9+   ]
}
  
```

Login into the AWS QA environment account and we can verify the account number of QA environment with policy attached in dev environment.

Federated Login: AWSReservedSSO_AWSPowerUserAccess_06ae7a87b362...

Account: 0175-8635-0412

My Account
My Organization
My Service Quotas
My Billing Dashboard
Orders and Invoices
Switch Role
Sign Out

Amazon DynamoDB
Want more scale? Try a serverless NoSQL database service for your modern application. [Get started](#)

Amazon SageMaker Studio
The first visual integrated development environment for machine learning. [Learn more](#)

AWS IQ
Connect with AWS Certified third-party experts for on-demand consultations and project help. [Get started](#)

AWS Security Hub

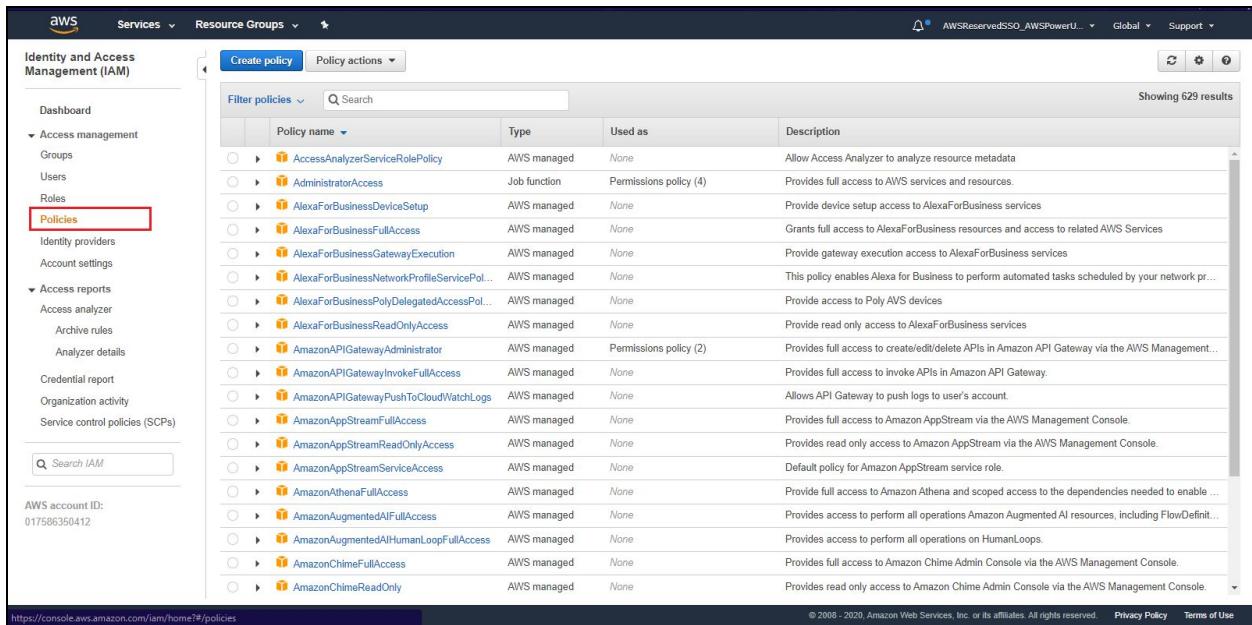
After login into the QA environment goto AWS IAM service

The screenshot shows the AWS Management Console homepage. At the top, there's a search bar with "iam" typed in, and a red box highlights the "IAM" service entry in the search results. Below the search bar, there's a "Recently visited services" section with links to IAM, RDS, EC2, CodeCommit, and VPC. A large "All services" section follows, containing categories like Compute, Blockchain, Security, Identity, & Compliance, and others. To the right, there are sections for "Access resources on the go" (with a link to the AWS Console Mobile App), "Explore AWS" (with links to Amazon DynamoDB, Amazon SageMaker Studio, and AWS IQ), and "AWS Security Hub".

Dashboard of the AWS IAM service will be as shown below

The screenshot shows the AWS IAM service dashboard. On the left, a navigation sidebar lists various IAM management options like Dashboard, Access management, and Access reports. The main content area has a "Welcome to Identity and Access Management" header and a "Feature Spotlight" video player for "Introduction to AWS IAM". Below these, there's a summary of IAM Resources (Users: 3, Groups: 1, Roles: 28, Customer Managed Policies: 9) and a "Security Status" section with four items: "Activate MFA on your root account", "Create individual IAM users", "Use groups to assign permissions", and "Apply an IAM password policy". A progress bar indicates "4 out of 4 complete". On the right, there's a "Additional Information" section with links to IAM best practices, documentation, and other resources.

Goto policies section



The screenshot shows the AWS Identity and Access Management (IAM) Policies page. The left sidebar has a 'Policies' link highlighted with a red box. The main content area displays a table of 629 policies, with columns for Policy name, Type, Used as, and Description. The table includes rows for various AWS services like AccessAnalyzerServiceRolePolicy, AdministratorAccess, AlexaForBusinessDeviceSetup, etc.

Policy name	Type	Used as	Description
AccessAnalyzerServiceRolePolicy	AWS managed	None	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	Job function	Permissions policy (4)	Provides full access to AWS services and resources.
AlexaForBusinessDeviceSetup	AWS managed	None	Provide device setup access to AlexaForBusiness services
AlexaForBusinessFullAccess	AWS managed	None	Grants full access to AlexaForBusiness resources and access to related AWS Services
AlexaForBusinessGatewayExecution	AWS managed	None	Provide gateway execution access to AlexaForBusiness services
AlexaForBusinessNetworkProfileServicePol...	AWS managed	None	This policy enables Alexa for Business to perform automated tasks scheduled by your network pr...
AlexaForBusinessPolyDelegatedAccessPol...	AWS managed	None	Provide access to Poly AVS devices
AlexaForBusinessReadOnlyAccess	AWS managed	None	Provide read only access to AlexaForBusiness services
AmazonAPIGatewayAdministrator	AWS managed	Permissions policy (2)	Provides full access to create/edit/delete APIs in Amazon API Gateway via the AWS Management...
AmazonAPIGatewayInvokeFullAccess	AWS managed	None	Provides full access to invoke APIs in Amazon API Gateway.
AmazonAPIGatewayPushToCloudWatchLogs	AWS managed	None	Allows API Gateway to push logs to user's account.
AmazonAppStreamFullAccess	AWS managed	None	Provides full access to Amazon AppStream via the AWS Management Console.
AmazonAppStreamReadOnlyAccess	AWS managed	None	Provides read only access to Amazon AppStream via the AWS Management Console.
AmazonAppStreamServiceAccess	AWS managed	None	Default policy for Amazon AppStream service role.
AmazonAthenaFullAccess	AWS managed	None	Provide full access to Amazon Athena and scoped access to the dependencies needed to enable ...
AmazonAugmentedAIFullAccess	AWS managed	None	Provides access to perform all operations Amazon Augmented AI resources, including FlowDefinit...
AmazonAugmentedAIHumanLoopFullAccess	AWS managed	None	Provides access to perform all operations on HumanLoops.
AmazonChimeFullAccess	AWS managed	None	Provides full access to Amazon Chime Admin Console via the AWS Management Console.
AmazonChimeReadOnly	AWS managed	None	Provides read only access to Amazon Chime Admin Console via the AWS Management Console.

Here we can see the cross account access policy which is created previously.

The screenshot shows the AWS IAM Policies page. The left sidebar has 'Policies' selected. A search bar at the top right contains the query 'cross'. The results table shows three policies:

	Policy name	Type	Used as	Description
1.	AmazonVPCCrossAccountNetworkInterface...	AWS managed	None	Provides access to create network interfaces and attach them to cross-account resources
2.	Americold_CrossAccount_Policy	Customer managed	Permissions policy (1)	
3.	CloudWatch-CrossAccountAccess	AWS managed	Permissions policy (1)	Allows CloudWatch to assume CloudWatch-CrossAccountSharing roles in remote accounts on beh...

Here we can check the permissions given to the policy which are required for the deployment

The screenshot shows the AWS IAM Policy Summary page for the policy 'Americold_CrossAccount_Policy'. The left sidebar has 'Policies' selected. The main area shows the policy details:

- Policy ARN: am.aws.iam:017586350412:policy/Americold_CrossAccount_Policy
- Description: (empty)
- Permissions tab is selected, showing the policy document:

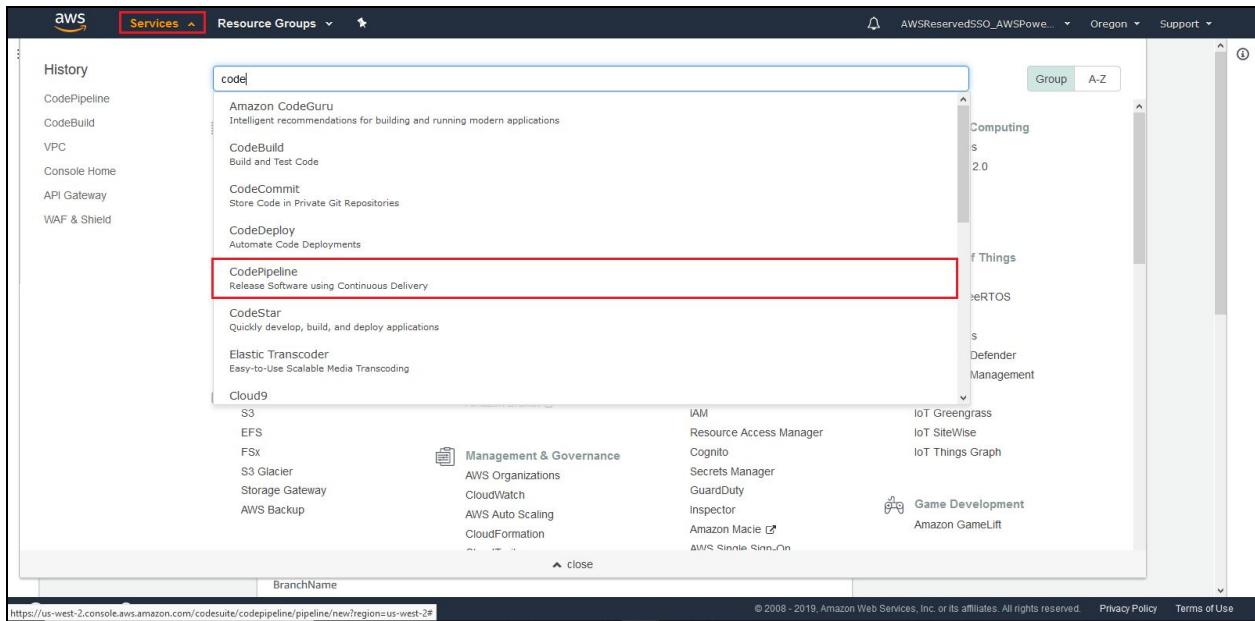
```

1 {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Sid": "VisualEditor0",
6             "Effect": "Allow",
7             "Action": [
8                 "iam:PassRole",
9                 "iam:GenerateCredentialReport",
10                "iam:GenerateServiceSpecificReport",
11                "iam:ListServiceLastAccessedDetailsWithEntities",
12                "iam>ListServerCertificates",
13                "iam:GenerateServiceLastAccessedDetails",
14                "logs:",
15                "iam>ListPoliciesGrantingServiceAccess",
16                "iam:.GetServiceLastAccessedDetails",
17                "iam:ListVirtualMFADevices",
18                "iam:GetOrganizationsAccessReport",
19                "cloudFront:*",
20                "iam:SimulateCustomPolicy",
21                "iam:GetAccountAuthorizationDetails",
22                "iam:GetCredentialReport",
23                "iam>ListPolicies",
24                "iam>ListSAMLProviders",
25                "ss:"]
26            ]
27        }
28    ]
29}

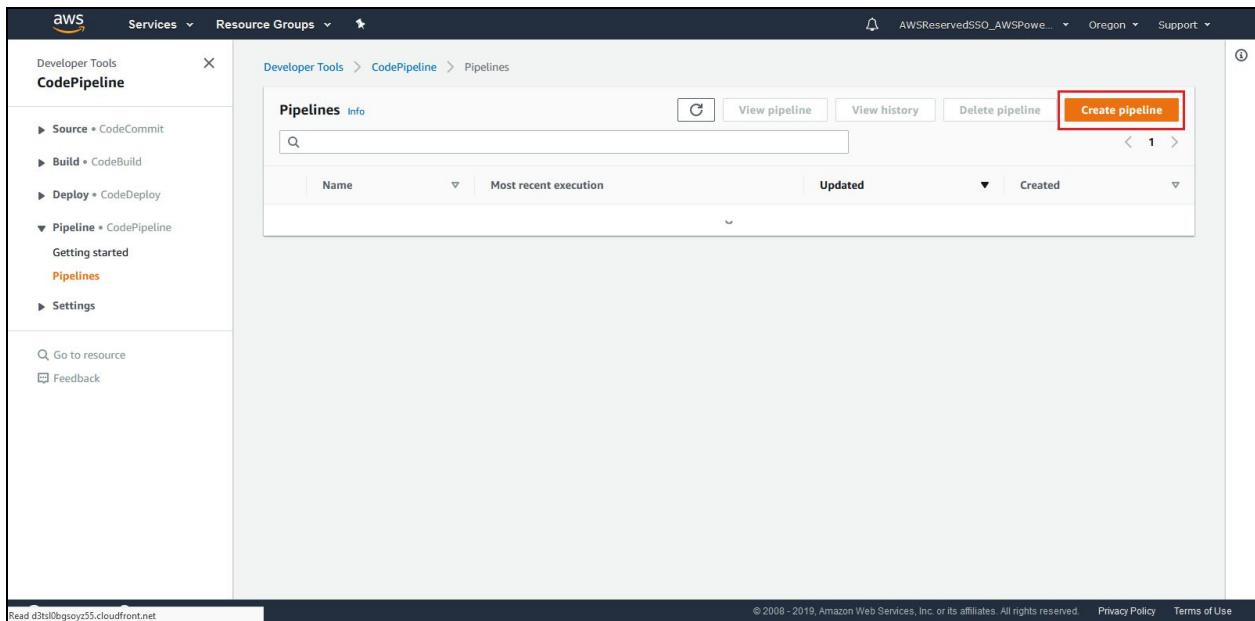
```

We are going to automate the process for integration and deployment using CodePipeline.

Goto **code pipeline service** in AWS console.



Click on **create pipeline** to create new pipeline for automated CI/CD.



Enter the name of the pipeline and choose the service role as new service role and click on **Next** as shown below. Here a new IAM service role will be created for code pipeline.

The screenshot shows the 'Choose pipeline settings' step in the AWS CodePipeline console. The pipeline name is set to 'Americold-BillingApp-Backend-NodeJs-QA'. The 'New service role' option is selected, and the role name is 'AWSCodePipelineServiceRole-us-west-2-Americold-BillingApp-Backe'. The 'Allow AWS CodePipeline to create a service role so it can be used with this new pipeline' checkbox is checked. The 'Next' button is highlighted with a red box.

Under the Add source stage, select the source provider as **AWS code commit**.

The screenshot shows the 'Add source stage' step in the AWS CodePipeline console. The 'Source provider' dropdown is open, showing 'AWS CodeCommit' selected. Other options include 'Amazon ECR', 'Amazon S3', and 'GitHub'. The 'Next' button is highlighted with a red box.

Select the name of the repository and branch name. choose the change detection options as

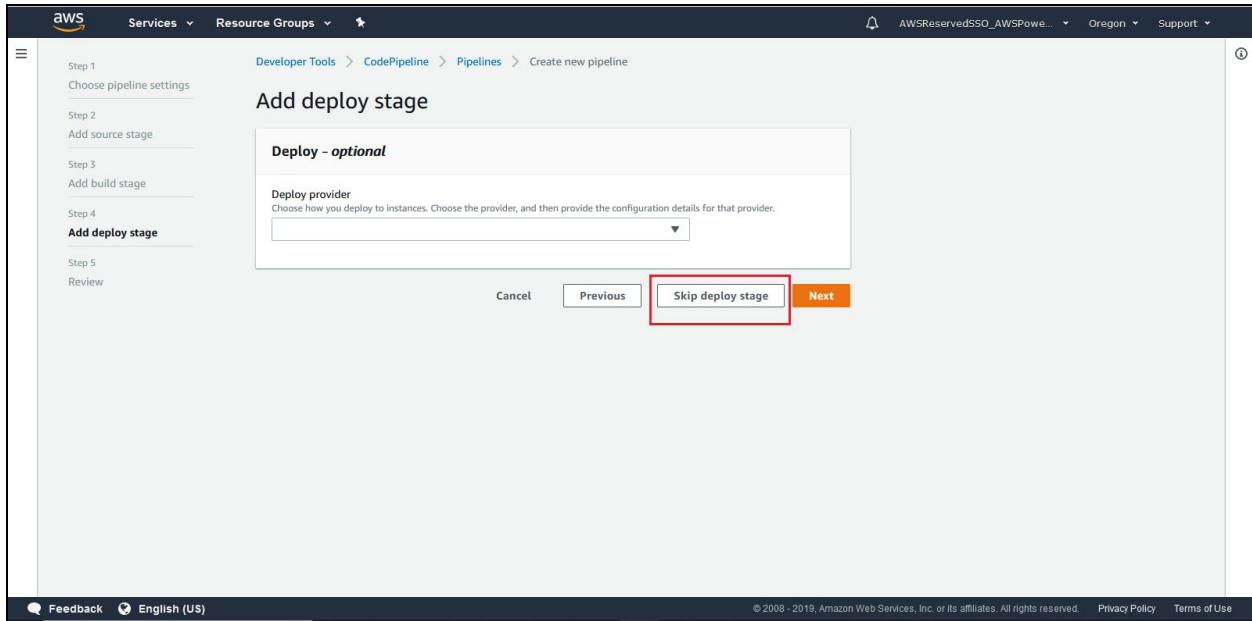
amazon cloud watch events to enable cloudwatch logs for the pipeline and click on **Next** to goto build configuration.

The screenshot shows the 'Add source stage' step in the AWS CodePipeline pipeline creation wizard. The 'Source' provider is set to 'AWS CodeCommit'. The repository name is 'Americold.BillingApp.Backend.NodeJs' and the branch name is 'QA'. Under 'Change detection options', the 'Amazon CloudWatch Events (recommended)' option is selected. The 'Next' button is highlighted with a red box.

Under add build stage, select the build provider as **AWS code build**, region as **US West – (Oregon)**, select the build project name that we created in the above steps and click on **Next** as shown below.

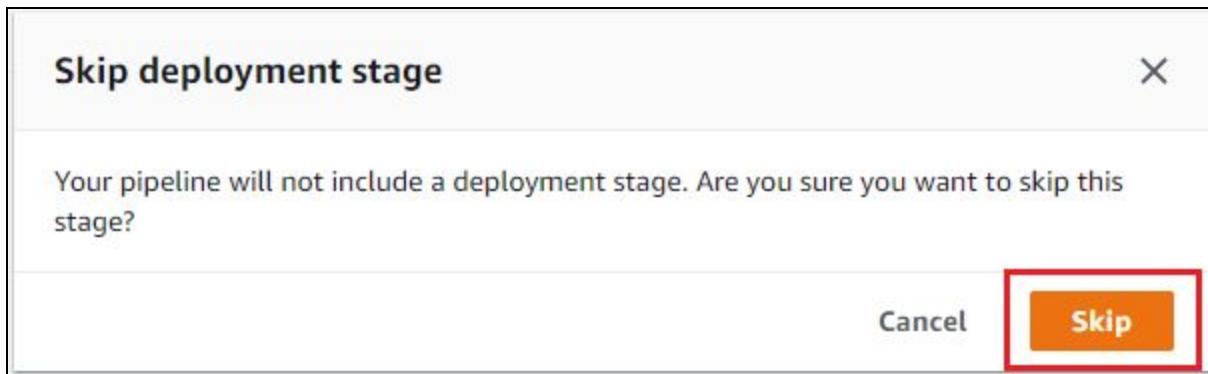
The screenshot shows the 'Add build stage' step in the AWS CodePipeline pipeline creation wizard. The 'Build provider' is set to 'AWS CodeBuild'. The 'Region' is set to 'US West - (Oregon)'. The 'Project name' is 'Americold-BillingApp-Backend-NodeJs-QA'. The 'Next' button is highlighted with a red box.

We are deploying through the commands. So, we can **skip** the deploy stage.

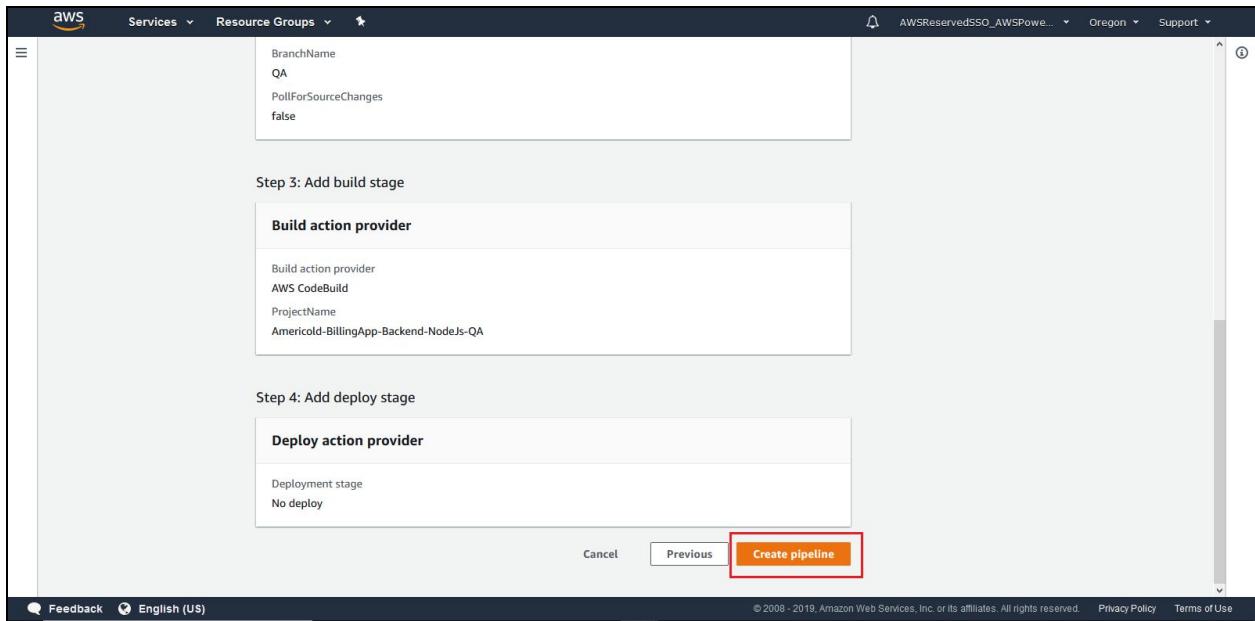


The screenshot shows the AWS CodePipeline 'Create new pipeline' wizard at Step 4: 'Add deploy stage'. On the left, a sidebar lists steps: Step 1 (Choose pipeline settings), Step 2 (Add source stage), Step 3 (Add build stage), Step 4 (Add deploy stage, currently selected), Step 5, and Review. The main area is titled 'Add deploy stage' and contains a section titled 'Deploy - optional' with a 'Deploy provider' dropdown menu. At the bottom, there are 'Cancel', 'Previous', 'Skip deploy stage' (which is highlighted with a red box), and 'Next' buttons.

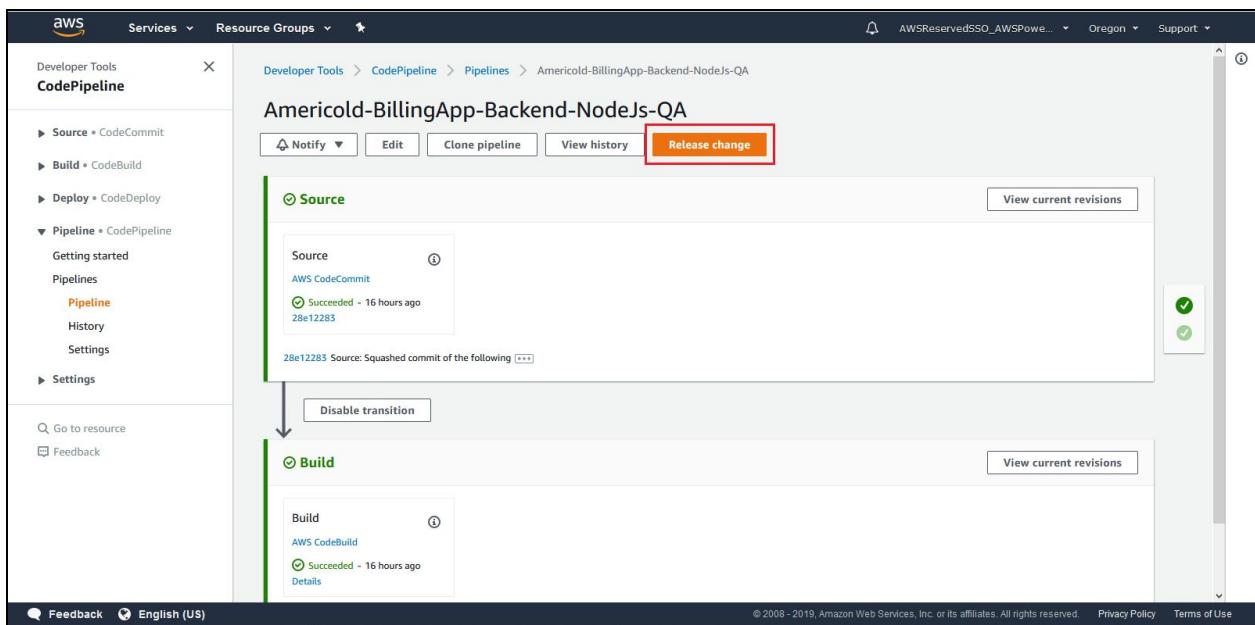
Click on **Skip**.



Review the pipeline configuration and click on **create pipeline**.



We can see the process is automated for CI/CD. We can check the stage logs by clicking on details. Pipeline will trigger automatically for the first time to release changes and later it will be triggered when the new changes are committed to repository.



We can check the **build logs** here.

The screenshot shows the AWS CodeBuild console for a project named "Americold-BillingApp-Backend-NodeJs-QA". The build status is "Succeeded". The build logs tab is selected, showing a list of API requests and responses. The logs indicate a successful deployment process, including pruning old artifacts and setting up monitoring.

Build status:

- Status: Succeeded
- Initiator: codepipeline/Americold-BillingApp-Backend-NodeJs-QA
- Build ARN: arn:aws:codebuild:us-west-2:646632670602:build/Americold-BillingApp-Backend-NodeJs-QA:40e8b079-d804-45d4-9598-2d21c5d6ec95
- Resolved source version: 28e12283bad1b17deb8c7adfb61e35483aa0cb1
- Start time: Dec 16, 2019 6:28 AM
- End time: Dec 16, 2019 6:29 AM
- Build Number: 31

Build logs:

```
117 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/inventoryConsolidation
118 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/facilityData
119 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/customerData
120 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/inventory/transaction
121 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearch
122 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearchApply
123 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearchView
124 DELETE - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/deleteSavedSearch/{Search_Name}
125 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/userInfo
126 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/usermenu
127 functions:
128   Login: americold-billingapp-nodeapis-q>Login
129   layers:
130     None
131 Serverless: Prune: Running post-deployment pruning
132 Serverless: Prune: Querying for deployed function versions
133 Serverless: Prune: americold-billingapp-nodeapis-q>Login has 3 additional versions published and 0 aliases, 1 version selected for deletion
134 Serverless: Prune: Deleting Function americold-billingapp-nodeapis-q>Login v16...
135 Serverless: Prune: Pruning complete.
136 Serverless: Removing old service artifacts from S3...
137 Serverless: Run the "serverless" command to setup monitoring, troubleshooting and testing.
138
139 [Container] 2019/12/16 00:59:44 Phase complete: POST_BUILD State: SUCCEEDED
140 [Container] 2019/12/16 00:59:44 Phase context status code: Message:
141
```

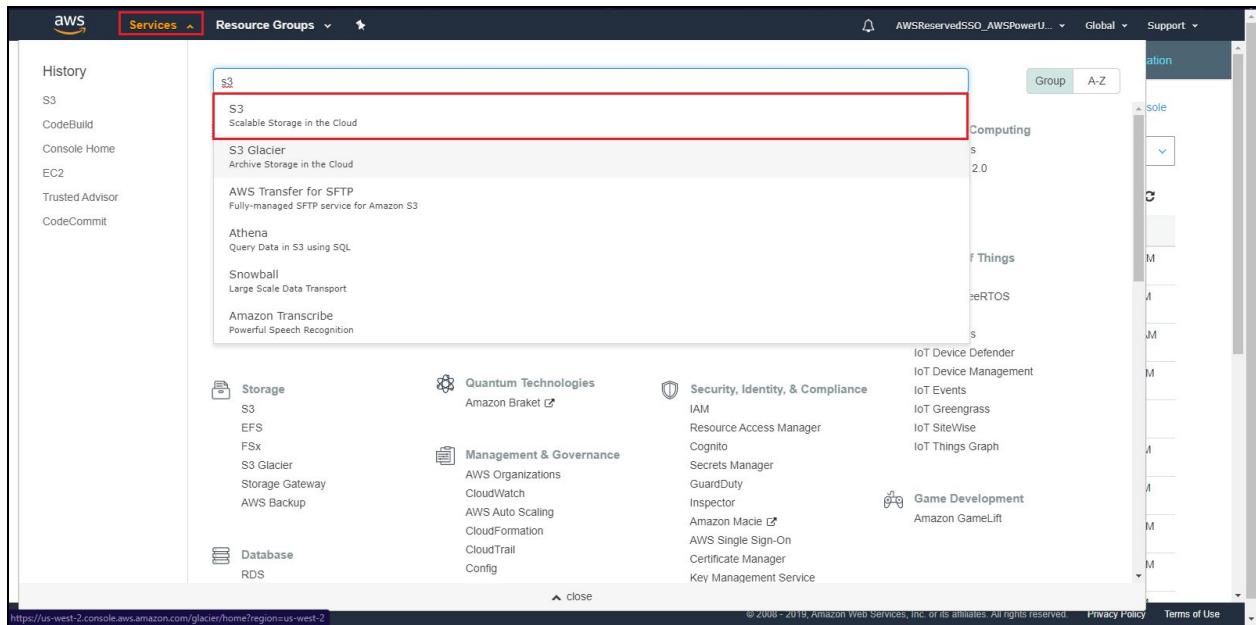
For backend applications, while running the **serverless deploy** command it will automatically create/updates the cloudformation stack using a serverless.yml template file.

Through the cloudformation template file, the artifacts will automatically deploy into **S3 bucket**, updates the lambda function and APIs

The screenshot shows the AWS Lambda function logs for a deployment. The logs detail the execution of the "serverless" command, including pruning old artifacts, updating the CloudFormation stack, and deploying new function versions. The deployment was completed successfully.

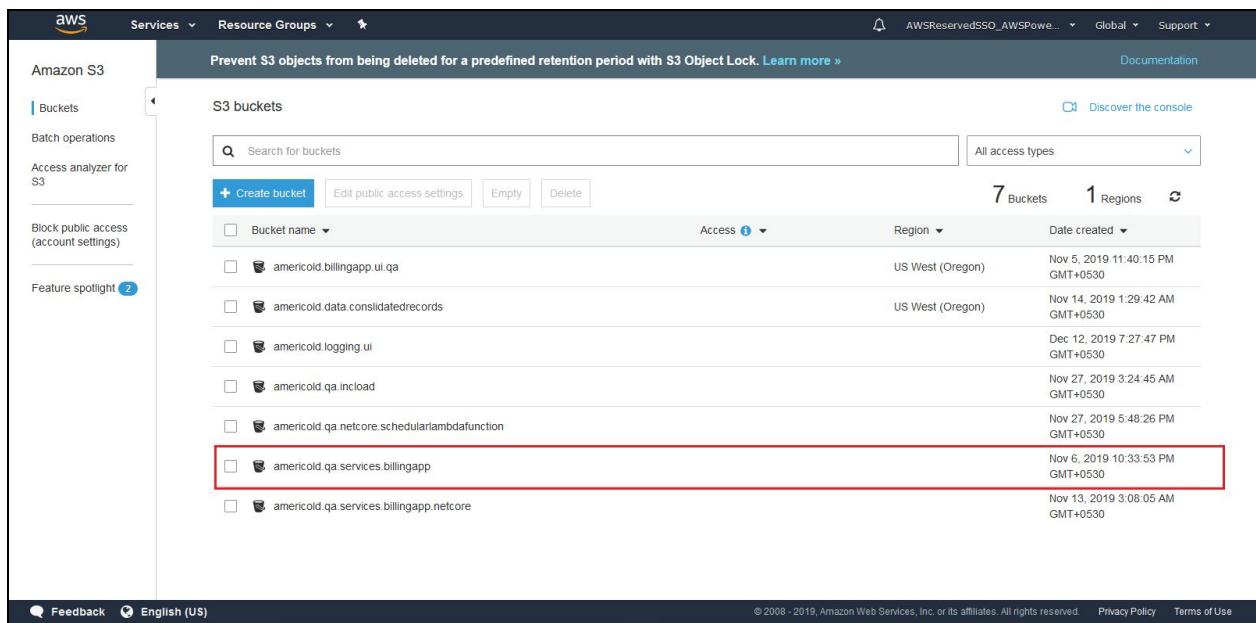
```
117 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/inventoryConsolidation
118 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/facilityData
119 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/customerData
120 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/inventory/transaction
121 POST - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearch
122 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearchApply
123 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/savedSearchView
124 DELETE - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/deleteSavedSearch/{Search_Name}
125 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/userInfo
126 GET - https://thexv56jod.execute-api.us-west-2.amazonaws.com/qb/api/usermenu
127 functions:
128   Login: americold-billingapp-nodeapis-q>Login
129   layers:
130     None
131 Serverless: Prune: Running post-deployment pruning
132 Serverless: Prune: Querying for deployed function versions
133 Serverless: Prune: americold-billingapp-nodeapis-q>Login has 3 additional versions published and 0 aliases, 1 version selected for deletion
134 Serverless: Prune: Deleting Function americold-billingapp-nodeapis-q>Login v16...
135 Serverless: Prune: Pruning complete.
136 Serverless: Removing old service artifacts from S3...
137 Serverless: Run the "serverless" command to setup monitoring, troubleshooting and testing.
138
139 [Container] 2019/12/16 00:59:44 Phase complete: POST_BUILD State: SUCCEEDED
140 [Container] 2019/12/16 00:59:44 Phase context status code: Message:
141
```

Go to S3 service to check whether the artifacts in the S3 bucket are updated or not.



The screenshot shows the AWS Services menu. The 'S3' service is highlighted with a red box. Other services listed include History, CodeBuild, Console Home, EC2, Trusted Advisor, and CodeCommit. Below the main menu, there are several service categories: Storage (S3, EFS, FSx, S3 Glacier, Storage Gateway, AWS Backup), Database (RDS), Quantum Technologies (Amazon Braket), Management & Governance (AWS Organizations, CloudWatch, AWS Auto Scaling, CloudFormation, CloudTrail, Config), Security, Identity, & Compliance (IAM, Cognito, Secrets Manager, GuardDuty, Inspector, Amazon Macie, AWS Single Sign-On, Certificate Manager, Key Management Service), and Game Development (Amazon GameLift). A sidebar on the right lists various AWS services like IoT Device Defender, IoT Device Management, IoT Events, IoT Greengrass, IoT SiteWise, IoT Things Graph, IoT Thing Graph, Game Development, Amazon GameLift, and others. The URL at the bottom is https://us-west-2.console.aws.amazon.com/glacier/home?region=us-west-2.

Following is the **S3 bucket** where we are storing our artifacts for backend Node.js application.



The screenshot shows the AWS S3 buckets list. The 'Buckets' section is selected in the sidebar. The main area displays a table of buckets. One bucket, 'americold.qa.services.billingapp', is highlighted with a red box. The table includes columns for Bucket name, Access, Region, and Date created. The bucket details are as follows:

Bucket name	Access	Region	Date created
americold.billingapp.ul.qa	Public	US West (Oregon)	Nov 5, 2019 11:40:15 PM GMT+0530
americold.data.consolidatedrecords	Public	US West (Oregon)	Nov 14, 2019 1:29:42 AM GMT+0530
americold.logging.ul	Public		Dec 12, 2019 7:27:47 PM GMT+0530
americold.qa.inload	Public		Nov 27, 2019 3:24:45 AM GMT+0530
americold.qa.netcore.schedulerlambdafunction	Public		Nov 27, 2019 5:48:26 PM GMT+0530
americold.qa.services.billingapp	Public		Nov 6, 2019 10:33:53 PM GMT+0530
americold.qa.services.billingapp.netcore	Public		Nov 13, 2019 3:08:05 AM GMT+0530

Here we can check our **latest artifact** in the S3 bucket.

The screenshot shows the AWS S3 console interface. The navigation bar at the top includes 'Services', 'Resource Groups', and 'Amazon S3'. Below the navigation is a breadcrumb trail: 'Amazon S3 > americold.qa.services.billingapp > serverless > americold-billingapp-nodeapis > qa'. A red box highlights the 'qa' folder in the breadcrumb trail. The main area is titled 'americold.qa.services.billingapp' with a 'Overview' tab selected. Below the overview is a search bar and a row of buttons: 'Upload', '+ Create folder', 'Download', and 'Actions'. To the right of these buttons is the region 'US West (Oregon)' and a refresh icon. The main content area displays a table of objects in the 'qa' folder. The columns are 'Name', 'Last modified', 'Size', and 'Storage class'. The first object, '1576044682548-2019-12-11T06.11:22.548Z', is highlighted with a red box. The table has a footer 'Viewing 1 to 5'.

Go to **cloud formation** to check whether the events got triggered or not.

The screenshot shows the AWS CloudFormation service page. The left sidebar lists services: History, CodeBuild, Console Home, EC2, Trusted Advisor, CodeCommit, S3, Storage, FSx, S3 Glacier, Storage Gateway, AWS Backup, Databases. A search bar at the top contains the text 'cloudform'. Below the search bar is a grid of service icons and names. Services listed include EC2, Lambda, S3, ECR, EKS, Lightsail, ECR, ECS, Batch, Blockchain, Amazon Managed Blockchain, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, AWS Image Builder, Storage, Quantum Technologies, Amazon Braket, Management & Governance, AWS Organizations, CloudWatch, AWS Auto Scaling, CloudFormation, CloudTrail, Security, Identity, & Compliance, IAM, Resource Access Manager, Cognito, Secrets Manager, GuardDuty, Inspector, Amazon Macie, AWS Single Sign-On, Computing, WorkSpaces, AppStream 2.0, WorkDocs, WorkLink, Internet Of Things, IoT Core, Amazon FreeRTOS, IoT 1-Click, IoT Analytics, IoT Device Defender, IoT Device Management, IoT Events, IoT Greengrass, IoT SiteWise, IoT Things Graph, Game Development, Amazon GameLift. The bottom of the screen shows the URL 'https://us-west-2.console.aws.amazon.com/cloudformation/home?region=us-west-2' and the standard AWS footer with links for 'Privacy Policy' and 'Terms of Use'.

Here we can check our **cloud formation stack** updated time.

The previous console is being deprecated in favor of the new console
The previous console is being deprecated in favor of this new AWS CloudFormation Console, that we are continually improving based on customer feedback. There will be no new features added or improvements made to the previous console. We will continue to support bug fixes and patches until 03/31/2020 after which the previous console will no longer be accessible.

CloudFormation > Stacks

Stacks (8)

Stack name	Status	Created time	Description
americoldawserverlessapi	UPDATE_COMPLETE	2019-11-20 00:35:35 UTC+0530	An AWS Serverless Application that uses the ASP.NET Co...
americold-billingapp-nodeapis-qa	UPDATE_COMPLETE	2019-11-11 19:25:43 UTC+0530	The AWS CloudFormation template for this Serverless ap...
StackSet-AWSControlTowerBP-VPC-AC...	CREATE_COMPLETE	2019-10-15 16:44:56 UTC+0530	This template creates a Multi-AZ, multi-subnet VPC infra...
StackSet-AWSControlTowerBP-BASELI...	CREATE_COMPLETE	2019-10-15 16:40:52 UTC+0530	Configure Cloudwatch Rule, local SNS Topic, forwarding ...
StackSet-AWSControlTowerBP-BASELI...	CREATE_COMPLETE	2019-10-15 16:37:46 UTC+0530	Configure AWS Config
StackSet-AWSControlTowerBP-BASELI...	CREATE_COMPLETE	2019-10-15 16:35:51 UTC+0530	Configure AWS CloudTrail
StackSet-AWSControlTowerBP-BASELI...	CREATE_COMPLETE	2019-10-15 16:32:54 UTC+0530	Configure AWS Config and SNS Notification Forward IAM...
StackSet-AWSControlTowerBP-BASELI...	CREATE_COMPLETE	2019-10-15 16:32:53 UTC+0530	Configure the Cross-Account IAM Security Roles for the ...

The previous console is being deprecated in favor of the new console
The previous console is being deprecated in favor of this new AWS CloudFormation Console, that we are continually improving based on customer feedback. There will be no new features added or improvements made to the previous console. We will continue to support bug fixes and patches until 03/31/2020 after which the previous console will no longer be accessible.

CloudFormation > Stacks > americold-billingapp-nodeapis-qa

Stacks (8)

americold-billingapp-nodeapis-qa

Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Overview

Stack ID	Description
arn:aws:cloudformation:us-west-2:017586350412:stack/americold-billingapp-nodeapis-qa/f3dcc030-048a-11ea-9ac8-068cd0948850	The AWS CloudFormation template for this Serverless application
Status	Status reason
UPDATE_COMPLETE	-
Root stack	Parent stack
-	-
Created time	Deleted time
2019-11-11 19:25:43 UTC+0530	-
Updated time	Last drift check time
2019-12-16 23:53:45 UTC+0530	-
Drift status	-

We can check our **events triggered** by our cloud formation stack.

The previous console is being deprecated in favor of the new console
The previous console is being deprecated in favor of this new AWS CloudFormation Console, that we are continually improving based on customer feedback. There will be no new features added or improvements made to the previous console. We will continue to support bug fixes and patches until 03/31/2020 after which the previous console will no longer be accessible.

CloudFormation

- Stacks
 - Stack details**
 - Drifts
 - StackSets
 - Exports
- Designer
- CloudFormation registry
 - Resource types
- Previous console
- Feedback

CloudFormation > Stacks > americold-billingapp-nodeapis-qa

Events

Timestamp	Logical ID	Status	Status reason
2019-12-16 23:53:59 UTC+0530	americold-billingapp-nodeapis-qa	UPDATE_COMPLETE	-
2019-12-16 23:53:59 UTC+0530	ApiGatewayDeployment 1576457954376	DELETE_COMPLETE	-
2019-12-16 23:53:58 UTC+0530	ApiGatewayDeployment 1576457954376	DELETE_IN_PROGRESS	-
2019-12-16 23:53:58 UTC+0530	LoginLambdaVersion82 bZemv4qDEULGe9bj AlfmrCrSnLcOafTQ93EyWU	DELETE_SKIPPED	-

Go to the Lambda service.

History

- CloudFormation
- S3
- CodeBuild
- Console Home
- EC2
- Trusted Advisor

Services

- lamb
- Lambda Run Code without Thinking about Servers
- Amazon Lex Build Voice and Text Chatbots
- CodeBuild Build and Test Code
- IoT 1-Click Trigger AWS Lambda functions from simple devices
- Elastic Beanstalk
- Serverless Application Repository
- AWS Outposts
- EC2 Image Builder
- Storage
 - S3
 - EFS
 - FSx
 - S3 Glacier
 - Storage Gateway
 - AWS Backup
- Databases

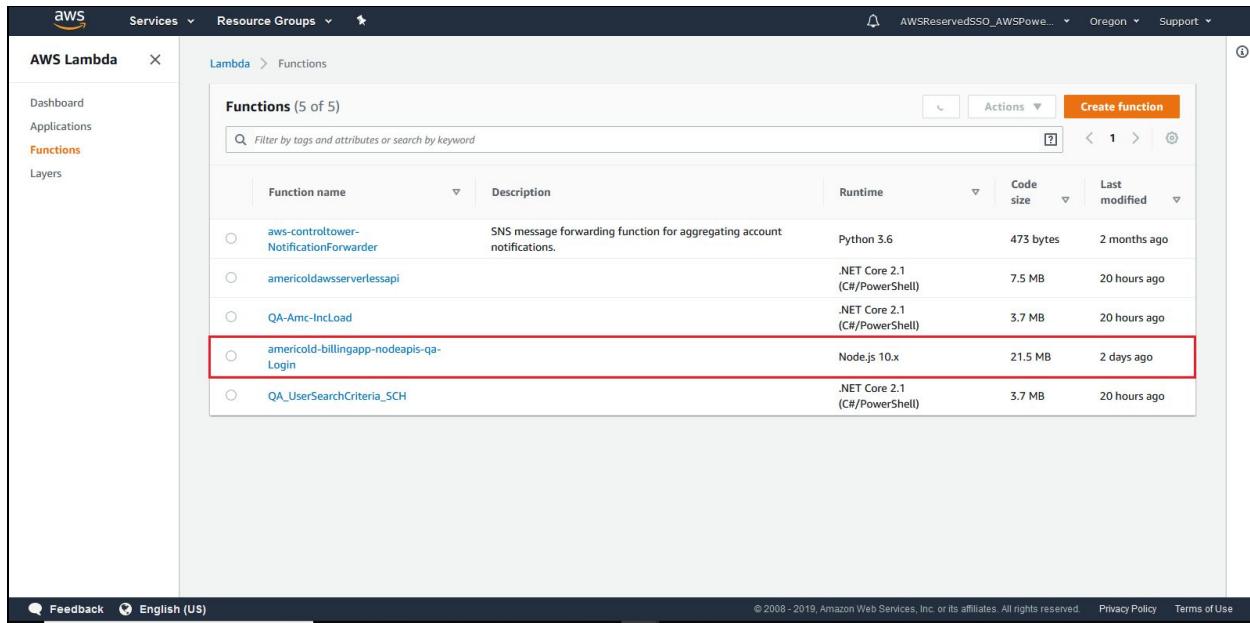
Resource Groups

CloudFormation > Stacks > americold-billingapp-nodeapis-qa

Events

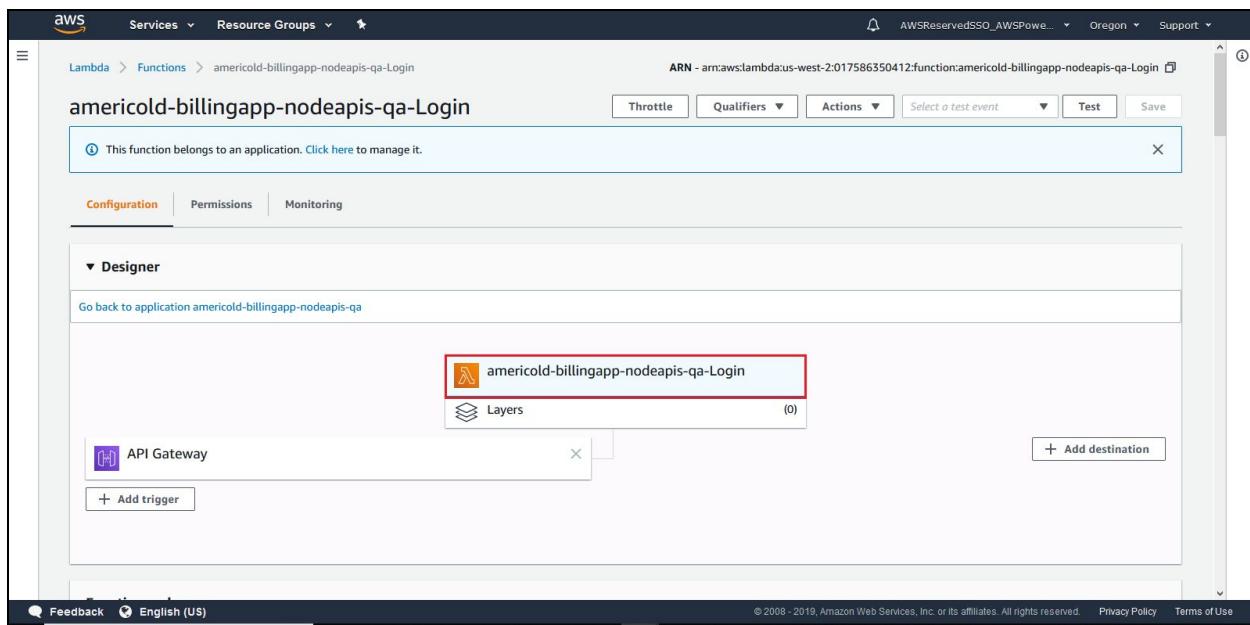
Timestamp	Logical ID	Status	Status reason
2019-12-16 23:53:59 UTC+0530	americold-billingapp-nodeapis-qa	UPDATE_COMPLETE	-
2019-12-16 23:53:59 UTC+0530	ApiGatewayDeployment 1576457954376	DELETE_COMPLETE	-
2019-12-16 23:53:58 UTC+0530	ApiGatewayDeployment 1576457954376	DELETE_IN_PROGRESS	-
2019-12-16 23:53:58 UTC+0530	LoginLambdaVersion82 bZemv4qDEULGe9bj AlfmrCrSnLcOafTQ93EyWU	DELETE_SKIPPED	-

Following is the **lambda function** that got updated.



The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with 'AWS Lambda' selected under 'Functions'. The main area is titled 'Functions (5 of 5)' and lists five functions:

Function name	Description	Runtime	Code size	Last modified
aws-controltower-NotificationForwarder	SNS message forwarding function for aggregating account notifications.	Python 3.6	473 bytes	2 months ago
americoldawsserverlessapi		.NET Core 2.1 (C#/PowerShell)	7.5 MB	20 hours ago
QA-Amc-IncLoad		.NET Core 2.1 (C#/PowerShell)	3.7 MB	20 hours ago
americold-billingapp-nodeapis-qa-Login		Node.js 10.x	21.5 MB	2 days ago
QA_UserSearchCriteria_SCH		.NET Core 2.1 (C#/PowerShell)	3.7 MB	20 hours ago



The screenshot shows the configuration page for the 'americold-billingapp-nodeapis-qa-Login' function. At the top, the ARN is displayed: ARN - arn:aws:lambda:us-west-2:017586350412:function:americold-billingapp-nodeapis-qa-Login. The page has tabs for Configuration, Permissions, and Monitoring. The Configuration tab is active, showing the 'Designer' section. It includes a 'Go back to application americold-billingapp-nodeapis-qa' link. Below that is a box containing the function icon and the name 'americold-billingapp-nodeapis-qa-Login', which is also outlined in red. To the right of this box is a 'Layers' section showing '(0)'. Below this is an 'API Gateway' section with a 'Trigger' button labeled '+ Add trigger' and a 'Destination' button labeled '+ Add destination'.

Here is the **endpoint** of API gateway.

The screenshot shows the AWS API Gateway Designer interface. At the top, the application name is 'americold-billingapp-nodeapis-qa-Login'. Below the header, there's a 'Designer' section with a 'Go back to application' link. The main area displays the API structure, which includes an 'API Gateway' entry. A red box highlights the 'API Gateway' section, which contains the API ID 'qa-americold-billingapp-nodeapis', its ARN, and its endpoint URL 'https://thexv56jod.execute-api.us-west-2.amazonaws.com/qa/api/userinfo'. The API is listed as 'Enabled'.

Goto AWS console and select **API gateway** service.

The screenshot shows the AWS Services menu. The 'Services' dropdown is open, and the 'API Gateway' service is selected, highlighted with a red box. Other services listed include Lambda, CloudFormation, S3, CodeBuild, and Console Home. To the right, a sidebar lists various AWS services grouped by category such as Computing, Machine Learning, and Game Development.

select the **API gateway** which is created/updated.

The screenshot shows the AWS API Gateway console. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, and account information (AWSReservedSSO_AWSPower... Oregon Support). Below the navigation is a search bar labeled 'Find APIs' and a 'Create API' button. The main area is titled 'APIs (2)' and contains a table with two rows:

Name	Description	ID	Protocol	Endpoint type	Created
americoldawsserverlessapi		u2lbh534uj	REST	Edge	2019-11-19
qa-americold-billingapp-nodeapis		thexv56jod	REST	Regional	2019-11-11

Here is the dashboard of the **API gateway**.

The screenshot shows the AWS API Gateway console with the URL 'APIs > qa-americold-billingapp-nodeapis... (thexv56jod) > Resources > / (8mo98xa316)'. The left sidebar has a red box around the 'Resources' link under the 'API: qa-americold-b...' section. The main pane shows a tree view of API resources under the path '/':

- /
- /api
 - /customerData
 - GET
 - OPTIONS
 - /deleteSavedSearch
 - /[Search_Name]
 - DELETE
 - OPTIONS
 - /facilityData
 - GET
 - OPTIONS
 - /inventoryConsolidation
 - OPTIONS
 - POST
 - /inventoryTransaction
 - OPTIONS
 - POST
 - /savedSearch
 - OPTIONS
 - POST
 - /savedSearchApply
 - GET
 - OPTIONS
 - /savedSearchView
 - GET
 - OPTIONS
 - /userinfo

No methods defined for the resource.

For API stage URL click on the stages and click on qa

The screenshot shows the AWS API Gateway interface. In the top navigation bar, 'Services' is selected under 'Resource Groups'. Below the navigation, the path is 'APIs > qa-americold-billingapp-nodeapis... (thexv56jod) > Stages'. On the left sidebar, 'Stages' is highlighted. In the main content area, there is a 'Create' button and a list of stages. The 'qa' stage is selected, indicated by a red box around its name. A placeholder text 'Select a stage' is visible above the stage list.

Here we can seed the Node.js API URL

The screenshot shows the 'qa Stage Editor' page. The top navigation bar shows the same API path as the previous screenshot. The left sidebar has 'Stages' selected. The main area is titled 'qa Stage Editor' and contains a 'Invoke URL' field with the value 'https://thexv56jod.execute-api.us-west-2.amazonaws.com/qa', which is also highlighted with a red box. Below this, there are tabs for 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History', 'Documentation History', and 'Canary'. Under the 'Settings' tab, there are sections for 'Cache Settings' (with an 'Enable API cache' checkbox), 'Default Method Throttling' (with an 'Enable throttling' checkbox checked, 'Rate' set to 10000, and 'Burst' set to 5000), and 'Web Application Firewall (WAF)' (with a 'Web ACL' dropdown set to 'Americold_API').

For backend API's we are configuring AWS WAF to provide security.

The screenshot shows the AWS API Gateway Stage Editor for the 'qa' stage of the 'qa-americold-b...' API. The left sidebar lists various resources like Authorizers, Models, and Settings. The main area shows Cache Settings, Default Method Throttling (with Rate set to 10000 and Burst to 5000), and a Web Application Firewall (WAF) section. A dropdown for 'Web ACL' is open, showing 'Americold_API' selected. Below it, a 'Client Certificate' dropdown is set to 'None'. At the bottom, there are links for Feedback, English (US), and a footer with copyright information.

Similarly, we are creating the CI/CD pipeline for the ASP dotnet core application. Following is the **buildspec.yml** file that we are using for integration and deployment for the ASP dotnet core application.

Following is the **buildspec.yml** file for ASP dotnet core application.

```

38 - cd Americold.Billing.Api
39 - dotnet restore
40 - dotnet tool install -g Amazon.Lambda.Tools
41 - dotnet lambda package --configuration release --framework netcoreapp2.1 --output-packs
42 # - command
43 # - command
44 post_build:
45 commands:
46 # - cd Americold.Billing.Api/src/Americold.Billing.Api
47 # - dotnet lambda deploy-serverless --region us-west-2 --stack-name americoldawsserverless
48 # - command
49 # - command
50 #artifacts:
51 #files:
52 # - location
53 # - location
54 #name: $(date +%Y-%m-%d)
55 #discard-paths: yes
56 #base-directory: location
57 #cache:
58 #paths:
59 # - paths|

```

Switch to single line

Artifacts

Below is the step by step explanation for the above script:

Install stage:

- **runtime-versions:**

dotnet : 2.2: It defines the runtime environment and version for the application

Pre_build stage:

- **apt install jq -y:** Install node modules using the following command
- **unset AWS_SESSION_TOKEN:** Clear the AWS session token
- **export AWS_REGION=us-west-2:** Set the AWS region
- **temp_role=\$(aws sts assume-role --role-arn "arn:aws:iam::017586350412:role/Americold_CrossAccount_Role" --role-session-name "QA"):** Create temporary AWS IAM role and store in the variable as shown above.
- **export AWS_ACCESS_KEY_ID=\$(echo \$temp_role | jq .Credentials.AccessKeyId | xargs)**

export AWS_SECRET_ACCESS_KEY=\$(echo \$temp_role | jq .Credentials.SecretAccessKey | xargs)

export AWS_SESSION_TOKEN=\$(echo \$temp_role | jq .Credentials.SessionToken | xargs): Set the AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_SESSION_TOKEN variables with their respective values using jq tool as shown above

Build Stage:

- **cd Inventory_consolidation:** Go to the folder called Inventory_consolidation
- **dotnet restore:** Following command uses NuGet to **restore** dependencies
- **dotnet tool install -g Amazon.Lambda.Tools:** Following commands used for Dotnet CLI

- to deploy AWS Lambda functions
- **dotnet lambda package --configuration release --framework netcoreapp2.1 --output-package bin/release/netcoreapp2.1/Americold.Billing.Api.zip:** To create a package to deploy we are using the following command

Post-Build stage:

- **dotnet lambda deploy-serverless --region us-west-2 --stack-name americoldawsserverlessapi --s3-bucket americold.qa.services.billingapp.netcore --s3-prefix Americold.Billing.Api/ --template-parameters "EnvironmentName=QA;LambdaSecurityGroups=\"sg-0a0182ed805d10d2f\";LambdaSubnetID=\"subnet-0b68418e0d291027c,subnet-05105151bc2487a14,subnet-085bab28792c2831f\""**

To deploy ASP Net Core application using serverless. template file we will run the following command. Here we have given some extra configuration for S3 bucket, VPC, and Cloudformation stack