**Date: 30-Jun-2020**

**Code Pipeline Setup for Containerized Application**

Contents

[AWS CodePilne 2](#_Toc46502358)

[Process Flow 2](#_Toc46502359)

[AWS Services 2](#_Toc46502360)

[CodeBuild 2](#_Toc46502361)

[CloudTrail 2](#_Toc46502362)

[CloudWatch 2](#_Toc46502363)

[CodePipeline 2](#_Toc46502364)

[CloudFormation for Deploy 3](#_Toc46502365)

[S3 3](#_Toc46502366)

[IAM Role Policies 3](#_Toc46502367)

[Source Code and GitHub 4](#_Toc46502368)

[Code 4](#_Toc46502369)

[GitHub 5](#_Toc46502370)

[Appendix 5](#_Toc46502371)

[Project 1 BuildSpec.YAML 5](#_Toc46502372)

[Project 2 BuildSpec.YAML 5](#_Toc46502373)

[CloudFormation Template used in Deploy step of CodePipeline 6](#_Toc46502374)

[CloudFormation Template used for creating CodePipeline: 6](#_Toc46502375)

# AWS CodePilne

AWS CodePipline service is meant for ‘Continuous Delivery’. For Chronos / Pavo applications, as per our standardize Git processes and defined release model, we will be using CodePipline for **DEV and QA** branch for CI/CD. Whenever code is getting merged to DEV from multiple feature branches / QA from DEV, it will trigger AWS CodePipline via Git Webhooks.

# Process Flow

GitHub code Push / Merge 🡪 Webhooks 🡪 Code Build Project 1 🡪 Creates source code zip file and push it to S3 bucket 🡪 Cloud Trail data event for S3 file push 🡪 Satisfies Cloud watch event rule 🡪 Code Pipeline 🡪 Code Build Project 2 will create and push image to ECR 🡪 Executes CloudFormation stack

**Please find elaborated details below for each step in above process flow**.

# AWS Services

## CodeBuild

As current NBC AWS CodePipeline is not supporting Code Build from Enterprise GitHub, we need to create two Code Build projects as detailed below for each Git Repository:-

1. Project 1: This project will create zip file of source code from GitHub on code Push or Merge to GitHub via webhooks and will add this ‘**SourceCode.zip**’ file to designated S3 Bucket Folder.
2. Project 2: This project will use SourceCode.zip file created by ‘Project 1’ and will create Docker image and push it to designated ECR using ‘**BuildSpec.YAML**’ file within source code

## CloudTrail

Need to add new Data Event to existing CloudTrail. Note that only 5 Cloud Trails are allowed per account. Hence we have added this data event into existing CloudTrail.

## CloudWatch

Need to create an event for each repository. ‘Event Pattern’ for this event is to look for new source code zip file pushed to designated S3 bucket, as a part of CodeBuild ‘Project 1’. For this we need to setup Data Event within CloudTrail for respective S3 bucket ‘SourceCode.zip’ file. Once this rule is satisfied, cloud watch event will trigger respective Code Pipeline as Target. For this target execution, cloudwatch event uses ‘ecsEventsRole’.

**Event Pattern:**

{ "source": [ "aws.s3" ], "detail-type": [ "AWS API Call via CloudTrail" ], "detail": { "eventSource": [ "s3.amazonaws.com" ], "eventName": [ "CopyObject", "PutObject", "CompleteMultipartUpload" ], "requestParameters": { "bucketName": [ "s3-w2-7006051-d-chronos" ], "key": [ "SourceCode.zip" ] } } }

**Target**

Code Pipeline ARN

## CodePipeline

CodePipeline need to be created for each Git Repository. This is a three step process as detailed below

1. Source: we need to provide two files as source from designated S3 buckets
   1. Source Code Zip File
   2. CloudFormation template
2. Build: in this step it will use ‘Project 2’ from Code Build. As per BuildSpec.YAML file, it will store build output artifacts in zip file containing ‘ImageDetail.Json’ to designated S3 bucket which stores ECR Image URL.
3. Deploy: This step uses CloudFormation Template and executes stack. While executing stack this step will pass on latest image name from ‘ImageDetail.Json’ to CF template ImageURL parameter. Due to this, when stack is getting executed, it will end up creating new task definition for a given cloud service and will terminate running tasks of old task definition.

## CloudFormation for Deploy

CloudFormation template for each repository which creates resources like CloudWatch Logs, Task Definition, Service, Target Group and Load Balancer Listener rules and it should be stored in S3 Bucket in zip file.

## S3

You need to create three folders for each repository. Hence within your bucket setup parent code pipeline folder for each repository and then within that create three sub folders

1. Folder 1: to store source code copy which will be used to create respective SourceCode.zip file in main bucket
2. Folder 2: to store build output zip file containing ImageDetail.json
3. Folder 3: to store CloudFormation template zip file

## IAM Role Policies

1. ecsEventsRole

Added new policy so that this role used within CloudWatch event will be able to trigger CodePipeline.

**PolicyName**; CodePipeline-Permissions-Policy-for-CWE

**JSON**:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"codepipeline:StartPipelineExecution"

],

"Resource": "\*"

}

]

}

1. Project Role (Research\_Chronos\_Dev\_Team) policies
   1. [AWSCodePipelineFullAccess](https://console.aws.amazon.com/iam/home?region=us-west-2#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAWSCodePipelineFullAccess)
   2. [AWSCloudFormationReadOnlyAccess](https://console.aws.amazon.com/iam/home?region=us-west-2#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAWSCloudFormationReadOnlyAccess)
   3. Other actions permissions

"cloudformation:\*",

"cloudfront:\*",

"cloudtrail:DescribeTrails",

"cloudtrail:Get\*",

"cloudtrail:List\*",

"cloudtrail:LookupEvents",

"cloudwatch:\*",

"codebuild:\*",

"codecommit:\*",

"codedeploy:\*",

"codepipeline:\*",

"iam:Get\*",

"iam:List\*",

"iam:PassRole",

"iam:AttachRolePolicy",

"iam:AttachGroupPolicy",

"iam:AttachUserPolicy",

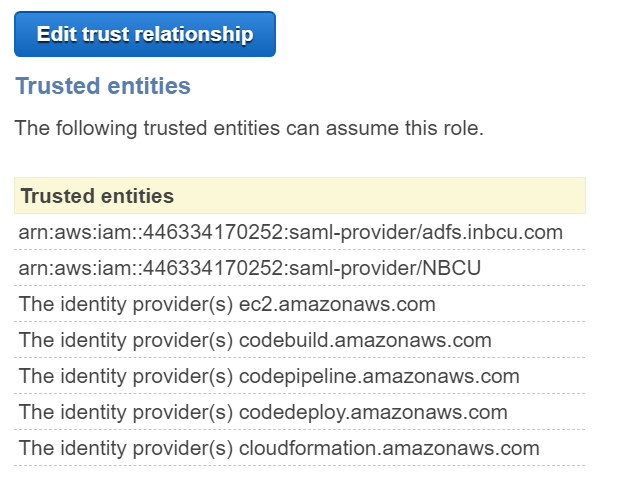
"iam:CreatePolicyVersion",

"iam:DeletePolicyVersion",

"iam:GetInstanceProfile",

"iam:PutRolePolicy"

* 1. Trust Policies



# Source Code and GitHub

## Code

1. Need to add BuildSpec.YAML file to respective branch in GitHub repository which will create Docker image and will push image to ECR. Please refer to Appendix section for details.

(or) Buildspec.yaml can be mentioned in cloudformation template which is used to create CodePipeline.

1. Need to remove proxy environment variables from ‘.dockerfile’. If not removed, it will end up failing build for ‘npm install’ command.

## GitHub

CodeBuild project 1 will generate web hooks. We need to add respective web hook details to respective GitHub repository

# Appendix

## Project 1 BuildSpec.YAML

version: 0.2

phases:

build:

commands:

- zip -r chronos\_ui\_SourceCode.zip .

- aws s3 cp chronos\_ui\_SourceCode.zip s3://s3-w2-7006051-d-chronos/

## Project 2 BuildSpec.YAML

version: 0.2

phases:

pre\_build:

commands:

- echo Logging in to Amazon ECR...

- aws --version

- $(aws ecr get-login --region us-west-2 --no-include-email)

- IMAGE\_NAME=chronos\_api\_master\_mvp

- REPOSITORY\_URI=<<ECR Image URI>>

- COMMIT\_HASH\_CODE=$(echo $CODEBUILD\_RESOLVED\_SOURCE\_VERSION | cut -c 1-7)

- IMAGE\_TAG=${COMMIT\_HASH\_CODE:=latest}

build:

commands:

- echo Build started on `date`

- echo Building the Docker image...

- docker-compose build

- docker tag $IMAGE\_NAME:latest $REPOSITORY\_URI:latest

- docker tag $REPOSITORY\_URI:latest $REPOSITORY\_URI:$IMAGE\_TAG

post\_build:

commands:

- echo Build completed on `date`

- echo Pushing the Docker images to ECR ...

- docker push $REPOSITORY\_URI:latest

- docker push $REPOSITORY\_URI:$IMAGE\_TAG

- printf '{"ImageURI":"%s:%s"}' $REPOSITORY\_URI $IMAGE\_TAG > imageDetail.json

artifacts:

files:

- imageDetail.json

## CloudFormation Template used in Deploy step of CodePipeline



## CloudFormation Template used for creating CodePipeline:



The above attached CloudFormation Template creates

1. CodeBuild Project for pulling source code from GitHub Enterprise Repository
2. CodePipeline with below stages

* Source
* Build
* Deploy

1. Buildspec files for both Project 1 and Project 2