Introduction

This document describes the Continuous Integration and Continuous Development (CICD) deployment into multiple application accounts of staging, QA and subsequently production. This CICD architecture is based on Lambda with serverless framework. This architecture is motivated by the compelling evidence to suggest that serverless computing platforms are potentially the future of cloud computing as it enables developers to execute arbitrary code on demand, without concern for the underlying hardware. Moreover, ECS is in the business of delivering enterprise transformation by utilizing innovative cloud technologies, therefore, providing an architecture that uses serverless framework may be useful to ECS and in the right direction.

Description of the architecture

This architecture comprises of two accounts. The central AWS Codepipeline account and the Deployments accounts (Staging and QA account). The deployments are carried out from the central count across other AWS accounts. As shown in the diagram, the first step is to create the necessary cross-account IAM roles. The IAM role (Cross-account role) shown in the Deployment account will provide the required trusts to AWS Codepipeline account. This trust provides the necessary permissions to CodeBuild and for the CodeDeploy to carry out the deployments. Moreover, the CloudFormation execution IAM role that has permissions to create the Lambda functions and API gateway are created in the deployment account. The CICD Pipeline is created using the CloudFormation template to provision the required resources (Source, Build and Deploy, Pipeline trigger and S3 bucket for the code pipeline).

Each part of the diagram is labelled to show the flow of events and for ease of reference and simplicity.

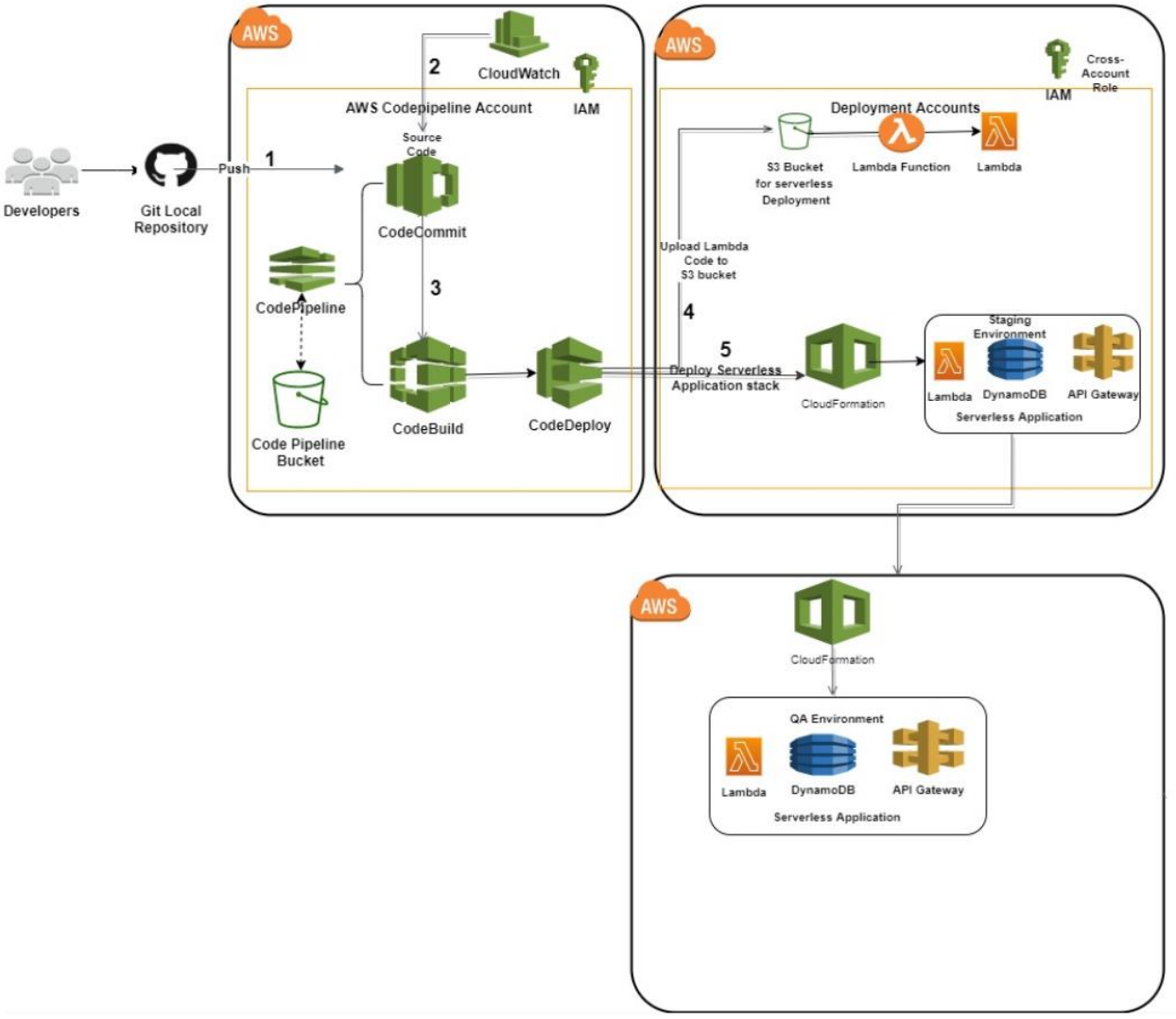

Figure 1: CICD Architecture

1. The CI/CD pipelines starts from the local git repository through the central AWS Codepipeline. The pipeline starts automatically every time the source code for the API are checked into the CodeCommit repository.
2. The CloudWatch event monitors the changes done to the CodeCommit repository and trigger the code pipeline when code is checked into the master repository
3. A code build is triggered at this stage using the build specification and stored in the Codepipeline bucket.
4. Using the necessary trust from IAM role, CodePipeline packages and uploads the Lambda code to the S3 bucket in the deployment account.
5. In this step, the deployment of the serverless application stack are done. The CloudFormation provision all the resources which includes the Lambda function and the associated API Gateway

and DynamoDB and deploys to the respective deployment accounts (Staging, QA and subsequently production).

NOTE 1. The necessary Deployment files for the serverless framework (buildspec.yml, serverless.yml and the lambda sources codes files) are used.

NOTE 2: The necessary integration tests are carried out before the final deployment to the production account

Advantage of the proposed AWS CICD architecture:

1. All pipelines are in the centralized account, which consolidates the security controls and grants increased visibility. One can argue that this will support the security pillar of the AWS well architected framework.
2. The AWS Identity and Access Management (IAM) permission model is greatly simplified because the pipelines can now share common IAM roles and policies.
3. Logs for all pipelines are in a single AWS Pipeline account under Amazon CloudWatch. This simplifies the monitoring of events in the deployment accounts.
4. Using Lambda enable the roll out of changes to the multiple deployment accounts by updating an AWS CloudFormation template.
5. Lambda enables the Creation of resources on demand in one stage of a pipeline using AWS CloudFormation and delete them in another stage.

Conclusion

This document has described a CICD pipeline architecture that is based on a serverless framework with Lambda to deploy application to multiple accounts. This deployment adheres to AWS best practices. The author believes that this architecture can be potentially reviewed and refined to meet the standards and expectation of ECS.

Reference

1. AWS (June 2017). Practicing Continuous Integration and Continuous Delivery on AWS Accelerating Software Delivery with DevOps. Available from https://d0.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf [Accessed 13/08/2020].
2. AWS (2020). AWS Serverless Application Model (AWS SAM) Specification. Available from https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification.html [Accessed 14/08/2020]
3. Deploying Serverless Applications. Available from: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-deploying.html [Accessed 14/08/2020]