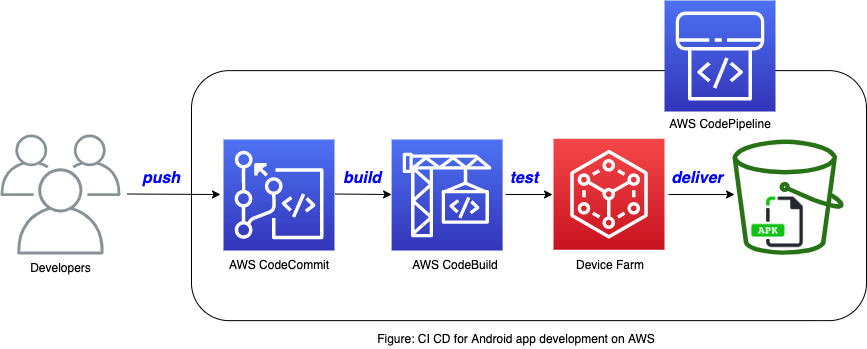
**Architecture**

In this workshop, we will be adding a continuous integration and continuous delivery (CI/CD) pipeline to the Android application development process.

To achieve this, we will be using the following AWS services -

* AWS CodeCommit - As a code repository for the Android app.
* AWS CodeBuild - To compile, package, and deliver the Android app.
* AWS Device Farm - To test the Android app on real physical devices.
* Amazon S3 - to store build artifacts (apk file).
* AWS CodePipeline - Overall pipeline - - To monitor the changes in CodeCommit repository - To trigger the build using CodeBuild and - To test using Device Farm.
* AWS CloudFormation - To provision all of these resources



Previous

Next

**Prerequisites**

This workshop requires an AWS account and access to manage the following services:

* AWS CodeCommit
* AWS CodeBuild
* AWS CodePipelines
* AWS Device Farm
* AWS Lambda
* Amazon Simple Storage Service
* AWS Cloudformation

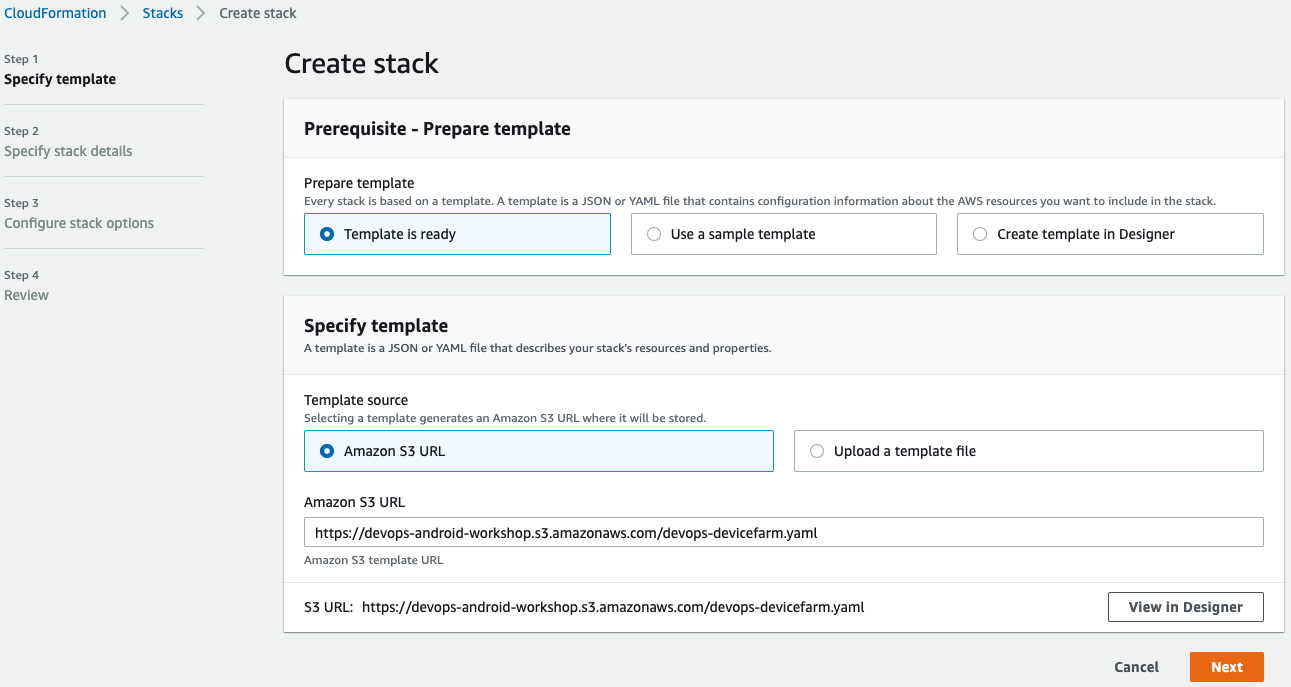
If not using an IAM user with AWS AdministratorAccess Policy, the user (or the role) would need the below AWS managed policies attached in order to successfully run this workshop -

* AWSCodeCommitPowerUser
* AWSCodeBuildAdminAccess
* AWSCodePipelineFullAccess
* AWSDeviceFarmFullAccess
* AWSLambdaFullAccess
* AmazonS3FullAccess
* AWSCloudFormationFullAccess

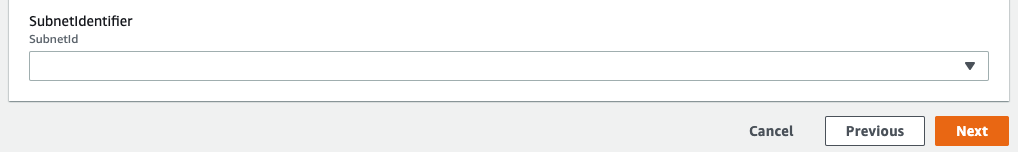
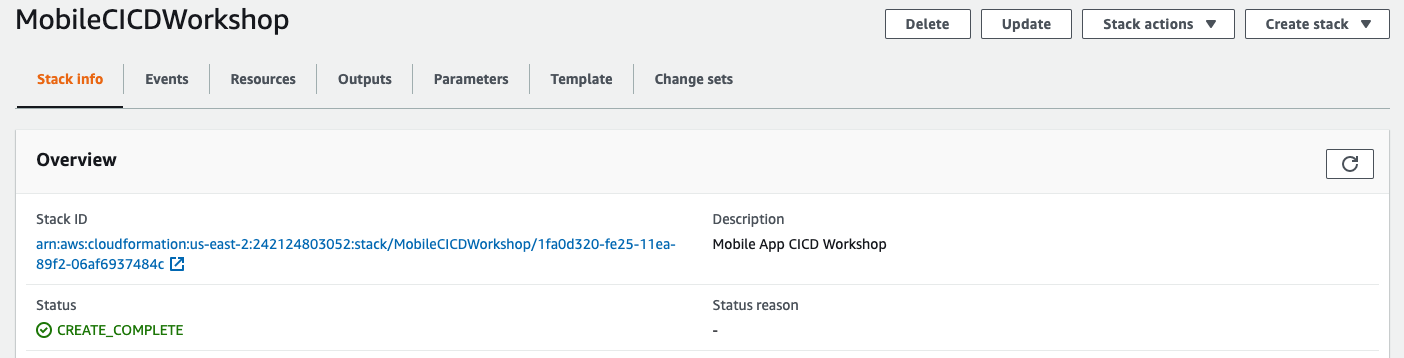
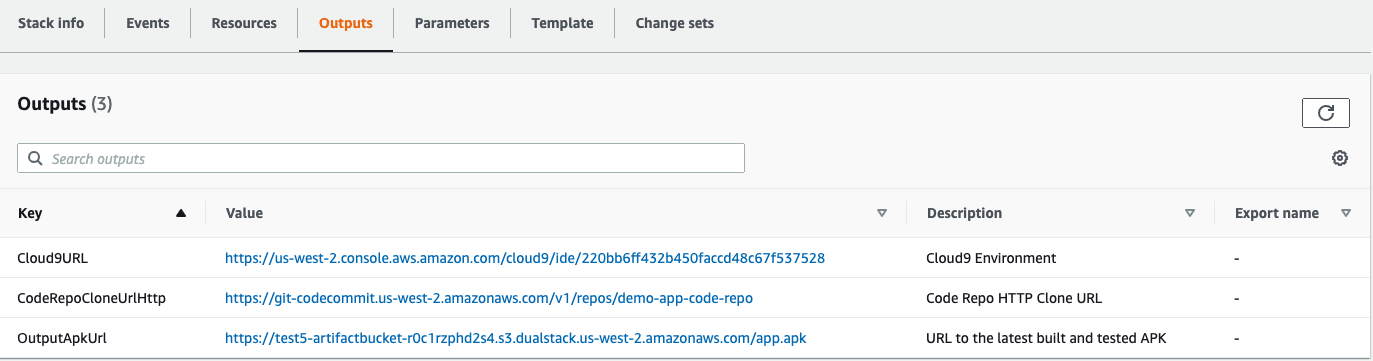
Refer here for more details on access management using IAM.

**Application Stack**

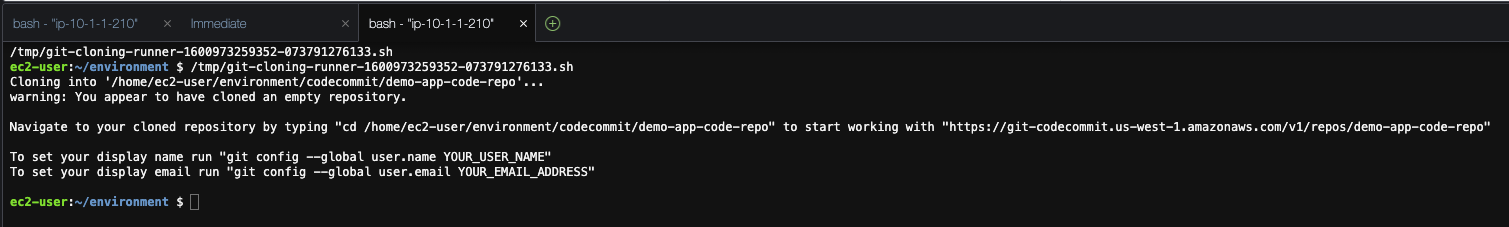
Follow below steps to deploy the application stack using AWS CloudFormation template:

1. Click on the button below to launch the stack Chapter Image
2. In Step 1 - Specific template, confirm the URL in the template source is correct - <https://devops-android-workshop.s3.amazonaws.com/devops-devicefarm.yaml> 

Please note that the US-East-2 region is be selected by default.

1. In Step 2 - Specify stack details, scroll to the bottom of the page and select any Subnet Identifier from the dropdown. Keep everything else as default. 
2. Keep everything as default in Step 3
3. In Step 4 - Review screen, scroll to the bottom of the page and Select I acknowledge that AWS CloudFormation might create IAM resources. (The stack uses IAM roles for Lambda, CodeBuild, and CodePipeline.).
4. Finally, click on 'Create stack'.
5. Wait for CloudFormation to create your stack. Once finished, you should notice the status as Create\_Complete as below - 
6. Now go to Outputs tab and take a note of the three values as seen below - 
7. Our application stack is ready. Open the Cloud9 URL in a new tab and proceed to the next section.

**Pre-Build Setup**

* We will use Cloud9 as our IDE to commit our sample android app code to the CodeCommit repository, which would trigger the CodePipeline.
* You should notice the CodeCommit repository is already linked to the environment as seen in the Cloud9 Terminal - 
* Lets execute below steps to get the sample code and trigger the build pipeline (the following commands are ran from the Cloud9 IDE Terminal ) -

1. Goto the demo-app-code-repo repository folder

cd codecommit/demo-app-code-repo/

1. Get the sample android app code

wget https://devops-android-workshop.s3.amazonaws.com/helloworld-app.tgz

1. Extract files from the tar ball and delete the tar ball -

tar -xvf helloworld-app.tgz

rm -rf helloworld-app.tgz

1. Add all of the files to the CodeCommit repository

git add -A

1. Commit and Push the changes to the CodeCommit repository

git commit -a -m "Initial checkin of the code"

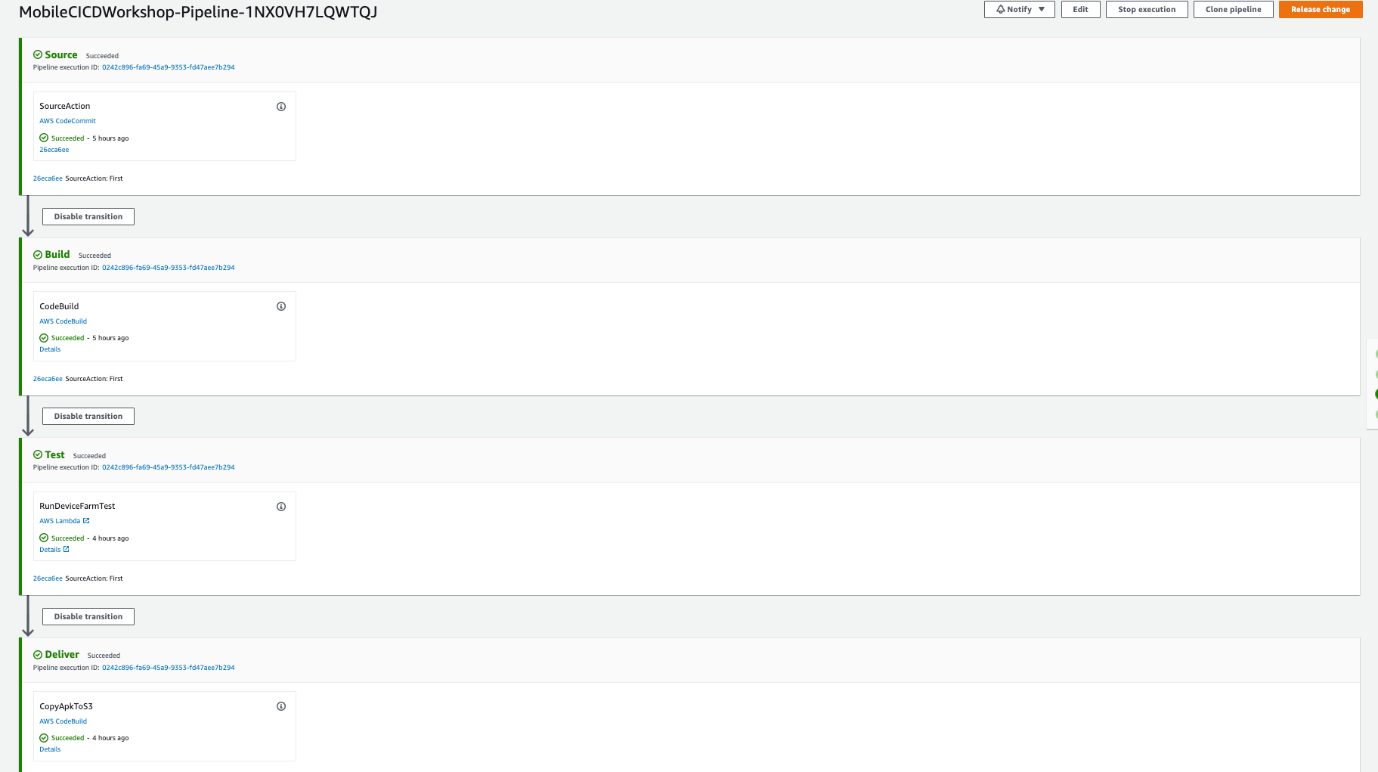
git push

* This push to the CodeCommit repository should trigger the execution of the CodePipeline. To view the progress of the pipeline execution, open the CodePipeline console and click on pipeline MobileCICDWorkshop-Pipeline-xxxxxx
* Now lets proceed to the next section to track the build and pipeline progress.

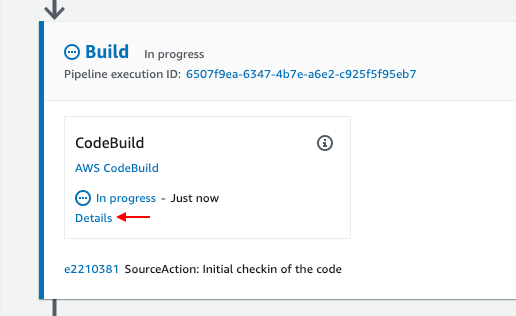
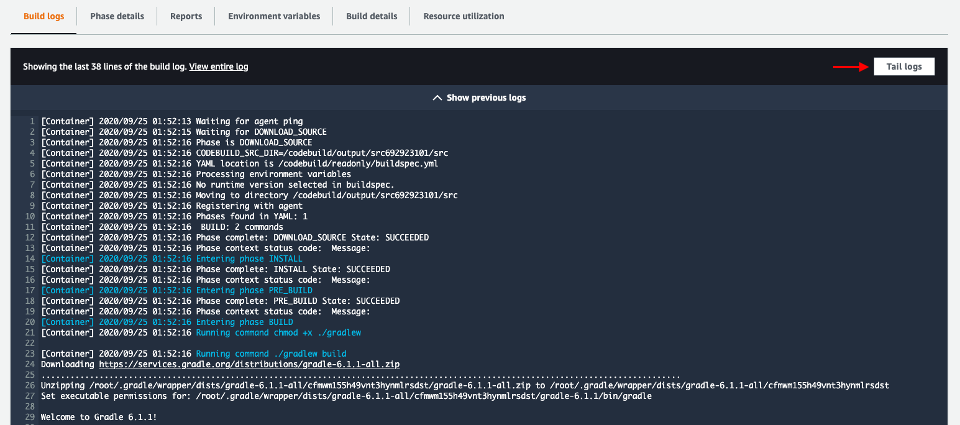
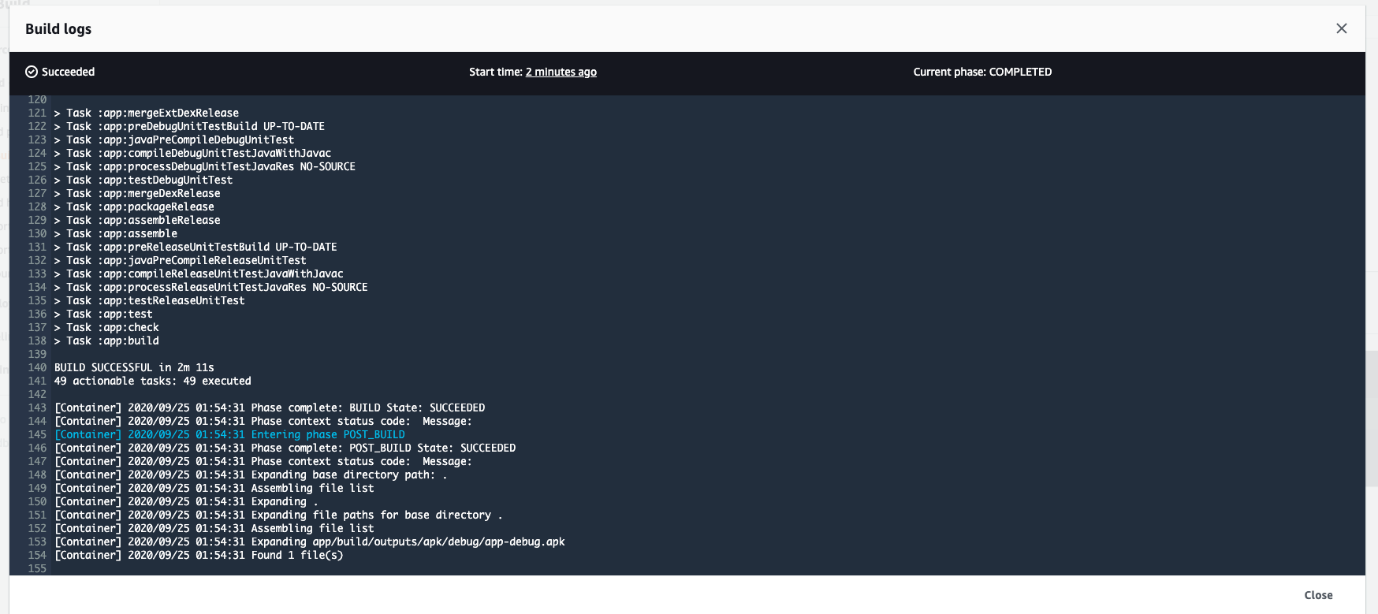
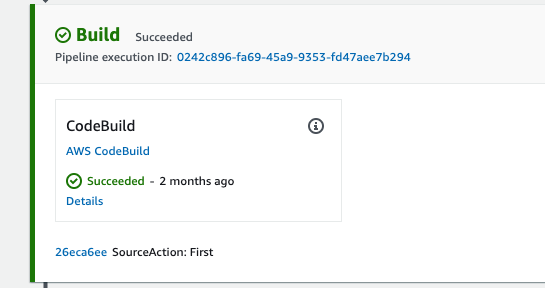
**Pipeline**

The CodePipeline execution goes through four stages (Source, Build, Test, and Deliver).

* SourceThe latest code is taken from the CodeCommit repository and packaged as a ZIP file for use by other stages.
* [Build](https://catalog.us-east-1.prod.workshops.aws/workshops/cc4e013e-6779-4574-9672-ff201b76282d/en-US/pipeline/build)
* [Test](https://catalog.us-east-1.prod.workshops.aws/workshops/cc4e013e-6779-4574-9672-ff201b76282d/en-US/pipeline/test)
* [Deliver](https://catalog.us-east-1.prod.workshops.aws/workshops/cc4e013e-6779-4574-9672-ff201b76282d/en-US/pipeline/deliver)

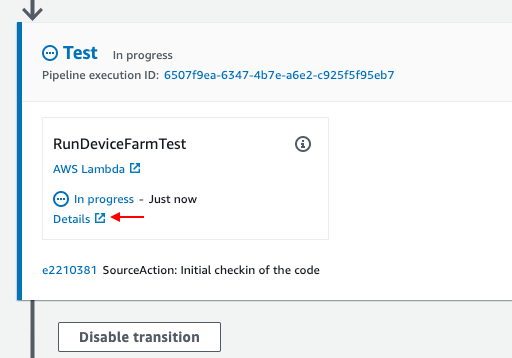
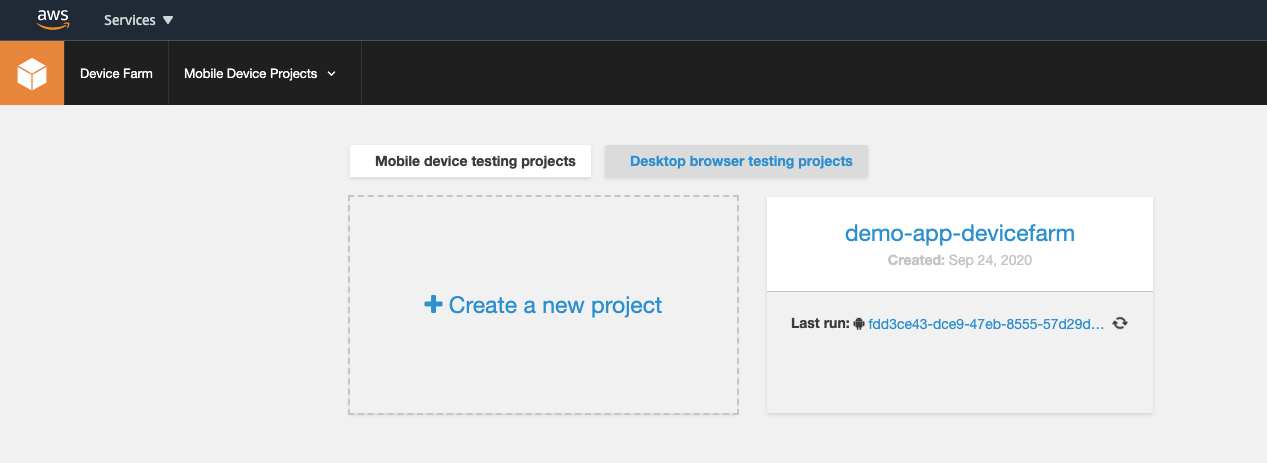
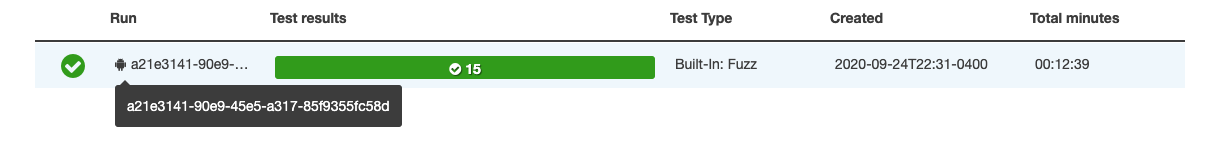
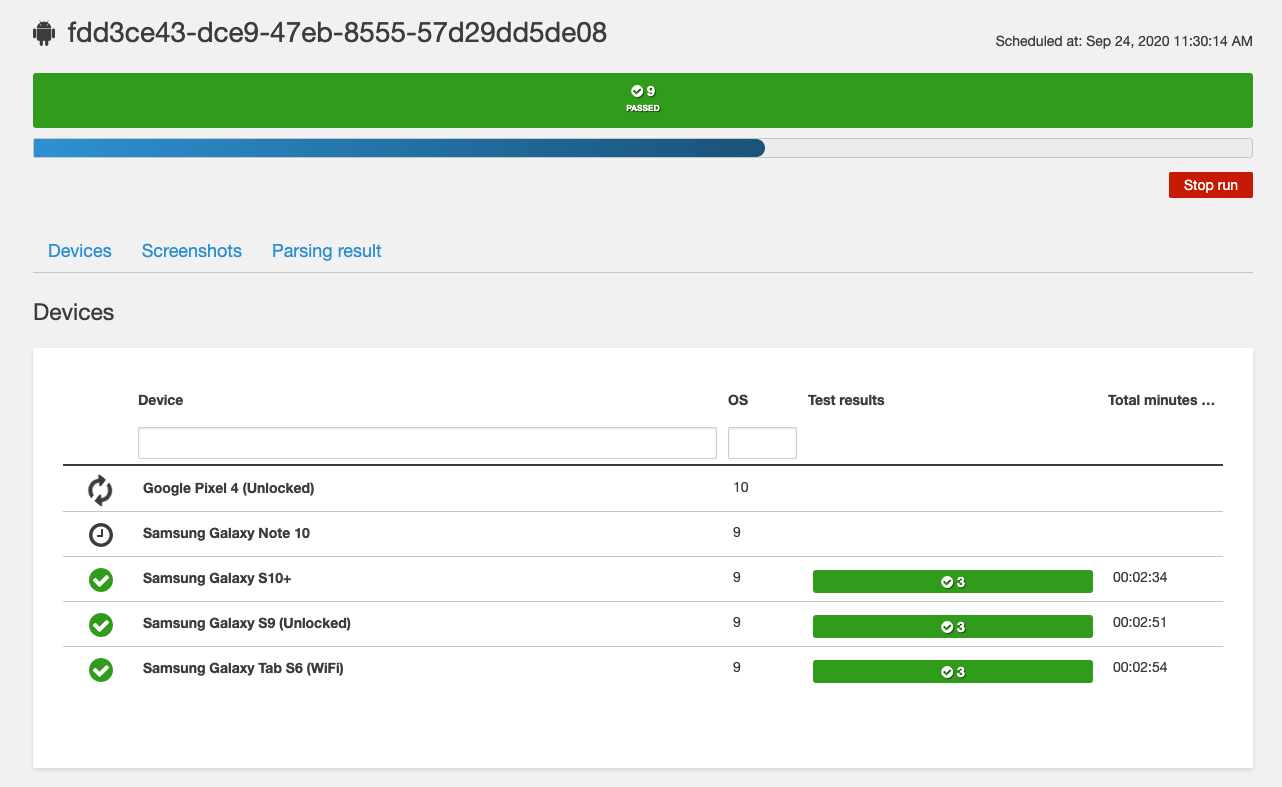
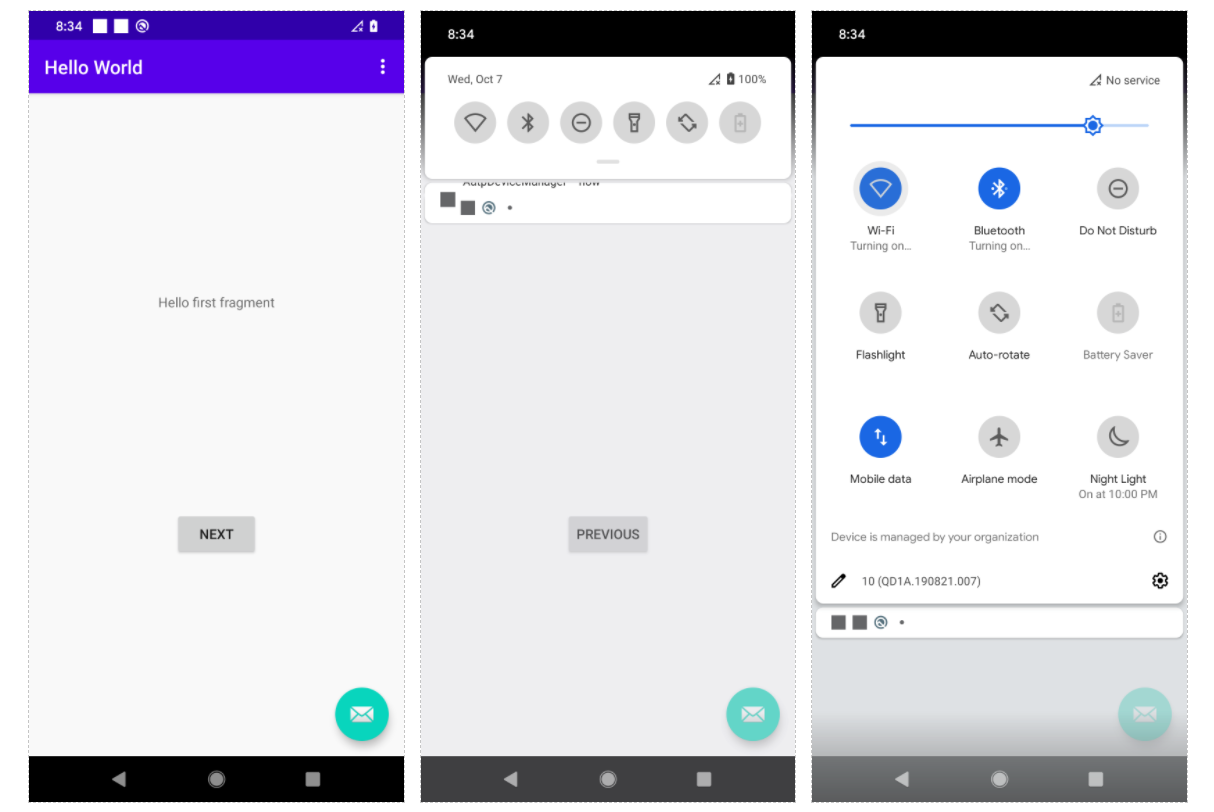
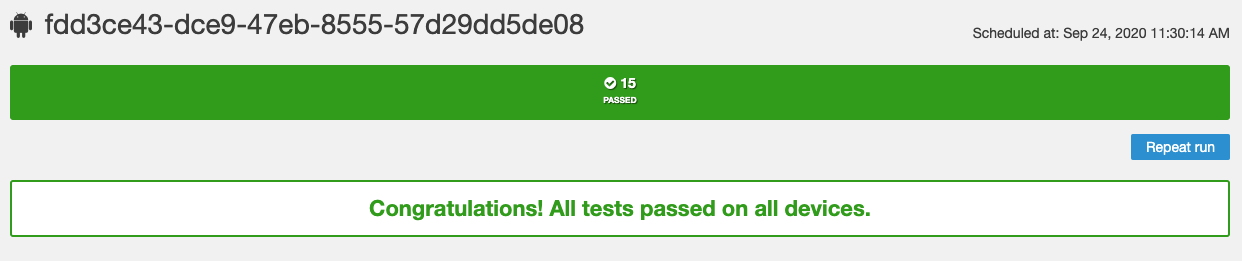


**Build**

* In this step, CodePipeline triggers the build of the android app using CodeBuild. To check the details of the build project open the CodeBuild console and check the Builder-xxxxxxx build project.
* You can also check the logs of the build process by clicking on Details from the CodePipeline section (as seen below). 
* Clicking on the Details takes you to the CodeBuild screen, where you can Tail Logs for in-depth tracking  
* Successful Build will mark the Build stage of the CodePipeline as Succeeded, and trigger the Test stage. 

In this CodeBuild project, we use a managed image aws/codebuild/standard:4.0 which is based on the Ubuntu 18.04 platform. With CodeBuild, there is an option to use Managed Images or Custom Images hosted on Amazon ECR or external Docker registry.

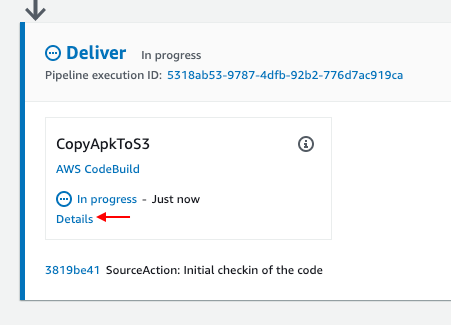
**Test**

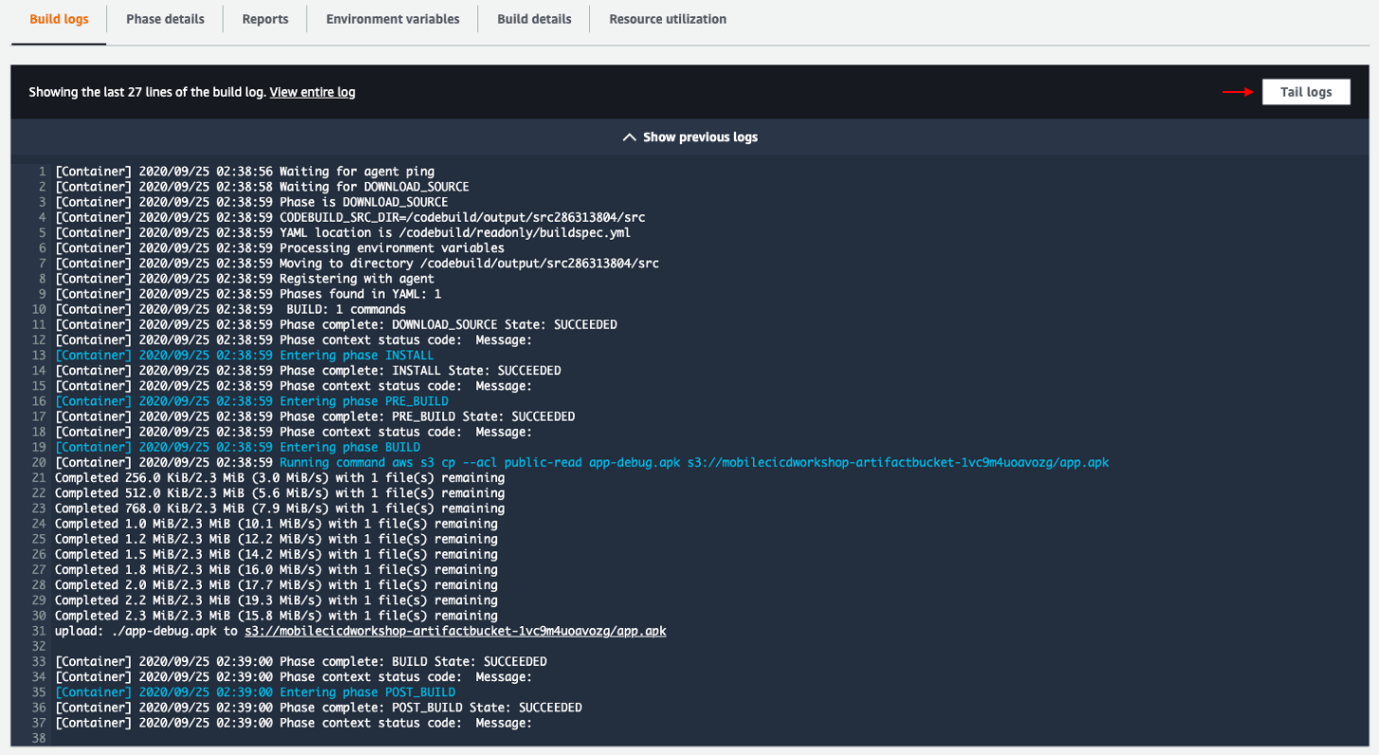
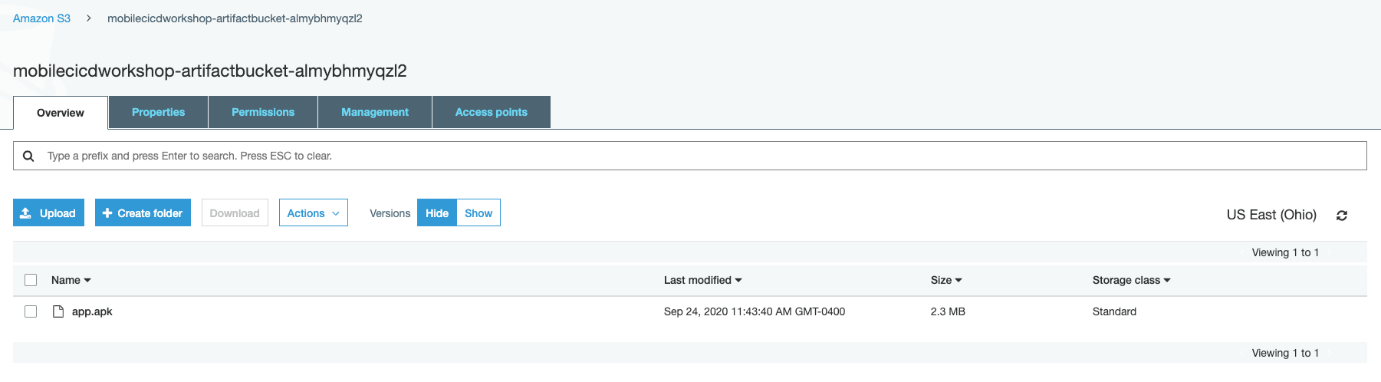
* In this step, CodePipeline triggers the a Lambda function which gets the build artifact generated by the build stage, uploads it into Device Farm, and starts the Device Farm test.
* Clicking on Details will take you to CloudWatch where you can track execution logs of the Lambda function 
* While the lambda function executes, go to the Device Farm console to view the test results. Click the project named demo-app-devicefarm 
* And click on the alphanumeric run-id. 
* Here you should see the progress of the test results on various devices including screenshots, as seen below   
* It should take around 10-15 mins for all the tests to complete successfully; and once complete, go back to the CodePipeline console to verify the Test stage marked Succeeded, and triggering the final Deliver stage.

For this sample app, we use Device Farm's Built-in: Fuzz test. For more serious uses cases, Device Farm also supports running Appium test scripts written in Java, Python, Ruby or Node.js. Test Types in AWS Device Farm

**Deliver**

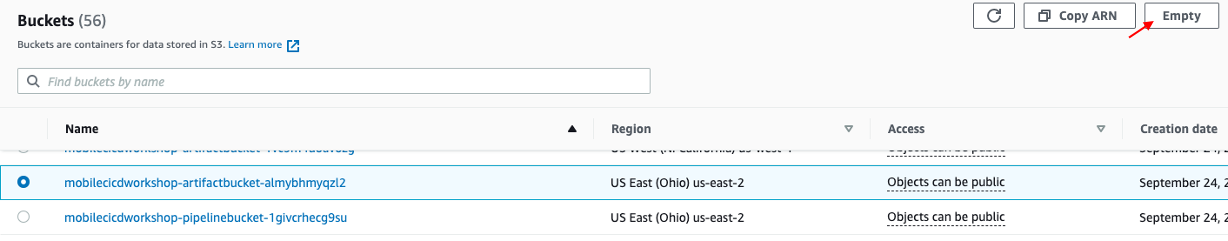
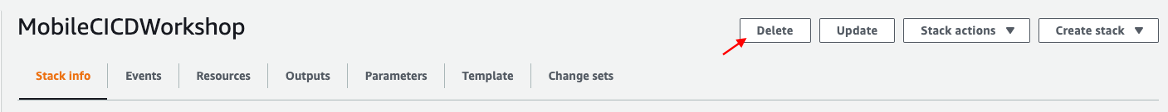
* Once the Device Farm tests are completed successfully, CodePipeline triggers a CodeBuild project to deliver the artifact (APK file) Amazon S3



* While the CodeBuild project is in progress, you can track the logs by clicking on Details 
* Finally, the artifact (APK file) is uploaded to a S3 bucket named mobilecicdworkshop-artifactbucket-xxxxx (this name might vary if you changed the name of the cloudformation stack) 

The CloudFormation stack has an output with a key named OutputApkUrl. The value contains a URL that you can use to download the APK file directly. You can also use the URL to install the APK file on an Android device.

**Wrap**

* Next Steps - Opportunities to explore
  + In this workshop, we have automated the building, testing, and delivery for an Android app by using AWS services.
  + To automate the delivery to the Google Play store, you can use tools such as fastlane. You can invoke these tools from CodeBuild. (Same can be done for iOS apps in Apple app store)
  + Beyond using the available managed images in CodeBuild, you can search Docker Hub for suitable images or you can create your own image and store it in Amazon EC2 Container Registry.
  + With Device Farm, you can run Appium, Calabash and Espresso tests for your android app.
  + Finally, you can develop Android (and iOS) apps with AWS Amplify and implement CICD with the approach discussed in this workshop.
* Clean up
  + Go to Amazon S3 console and empty the below buckets
    - mobilecicdworkshop-pipelinebucket-xxxx
    - mobilecicdworkshop-artifactbucket-xxxx 
  + Go to the CloudFormation Console and select the MobileCICDWorkshop stack that was created, and click on Delete. 
  + The related nested stack will also be automatically deleted