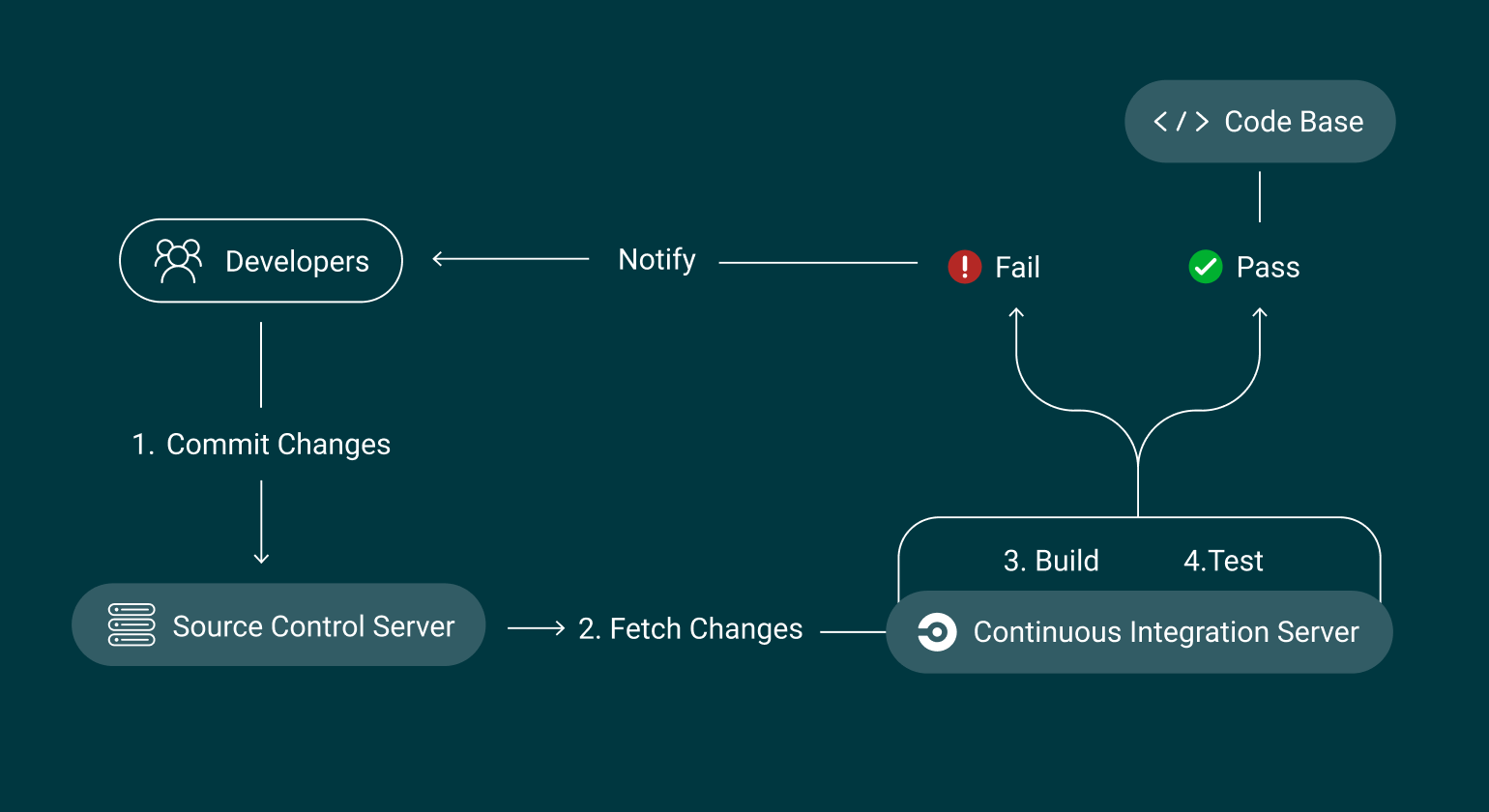
**Continuous Integration Testing: circleci**

1. **Introduction:**

Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository, after which automated builds and tests run.

Continuous integration (CI) is a software development strategy that improves both the speed and quality of code deployments. In CI, developers frequently commit code changes, often several times a day. Each change triggers an automated build and test sequence, ensuring that new code works with the existing codebase. If any issues are discovered during the test phase, the CI platform blocks the code from merging and alerts the team so they can quickly fix any errors.



**Figure 1: Continuous Integration (CI) Workflow Diagram**

The above diagram depicts a Continuous Integration (CI) workflow diagram.

Here’s a description of diagram and it’s elements:

1. **Developers:**

The process begins with developers making changes to the codebase. This step is labeled as "1. Commit Changes."

1. **Source Control Server:**

Once the developers commit their changes, these changes are stored in the Source Control Server.

The diagram labels this as "2. Fetch Changes," indicating that the CI server fetches the latest changes from the source control.

1. **Continuous Integration Server:**

The CI server retrieves the latest changes and starts the CI pipeline. This stage includes two primary steps:

**Step 1: Build**

The system builds the code to ensure it compiles correctly.

**Step 2: Test**

Automated tests are run to validate the correctness of the code.

1. **Pass/Fail Branch:**

The diagram then splits into two outcomes after testing:

**Pass:** If the tests pass, the process moves forward, and the code is successfully integrated.

**Fail:** If the tests fail, the CI server notifies the developers, and they must resolve the issues before proceeding.

This process ensures that code changes are continuously integrated and tested, helping to maintain code quality and reduce integration issues.

1. **How Continuous Integration Works**

CI is a way to build software faster and better by automating repetitive tasks. It helps prevent errors and makes the development process more efficient. Here’s a step-by-step look at how it works:

1. **Commit:**

Developers continuously push their code changes to a shared repository, often multiple times a day. This practice ensures that new code is consistently integrated with the existing codebase.

1. **Build:**

Once code changes are committed, CI systems automatically build the application. This ensures that the new code works with the existing codebase and that the application is deploy-ready at all times.

1. **Test:**

After the build process, the CI server runs automated tests to assess the changes’ impact on the application’s functionality, security, and adherence to your organization’s policies. Common tests include unit tests, integration tests, security and compliance scans, and code quality checks.

1. **Inform:**

CI provides rapid feedback to development teams regarding the success or failure of their code changes. This information keeps developers constantly informed about the health of their application, enabling them to iterate quickly and with greater confidence.

1. **Integrate:**

Once the build and test processes complete successfully, the changes are automatically merged into the main branch. This step ensures that updates are available to all team members and that the mainline stays up to date with the latest working version.

1. **Deploy:**

CI is often combined with continuous delivery (CD), creating an automated deployment pipeline. Once the code passes all tests, it can be automatically deployed to staging environments for further evaluation or sent directly to production, depending on the organization’s policies.

1. **CI/CD Tools:**
2. **Jenkins**
3. **CircleCI**
4. **GitLab CI**
5. **GitHub Actions**
6. **Travis CI**
7. **TeamCity**
8. **Azure Devops**

Now I am writing about the benefits of using **circleci** for continuous integration and also discussing how to build a CI/CD pipeline with **circleci**:

1. **Key advantages of using CircleCI for CI/CD pipelines**

CircleCI is a popular continuous integration and continuous delivery (CI/CD) platform that offers several advantages for developers and organizations looking to automate their software development lifecycle. Here are some key benefits of using CircleCI:

1. **Ease of Setup and Configuration:**

CircleCI pipelines are configured using simple YAML files, making it easy to define and customize workflows. It provides pre-built configurations for popular languages and frameworks, simplifying the setup process for developers.

1. **Scalability and performance:**

CircleCI automatically scales infrastructure based on the demands of your project. Whether you're running a few tests or thousands, it can handle the load with minimal delay. It supports parallel job execution, enabling faster builds by distributing tests across multiple containers. It offers robust caching mechanisms to save build dependencies and results, reducing redundant steps in future builds and improving pipeline speed.

1. **Integration with popular tools:**

CircleCI integrates with popular version control systems (like GitHub, GitLab, Bitbucket), cloud platforms (AWS, GCP, Azure), and other DevOps tools (Slack, Jira, etc.).

1. **Cross-Platform Support:**

CircleCI supports a wide range of programming languages, build tools, and environments, making it suitable for teams. It supports Linux, macOS, and Windows, which is useful for testing software across different operating systems.

1. **How to build a CI/CD pipeline with circleci:**

Continuous Integration (CI) is employed to determine whether incorporating a smaller module into the broader project will cause any problems or conflicts with the existing codebase.

A CI pipeline is triggered by code changes and verifies the compatibility of all modifications with the rest of the code. It includes steps for compiling, testing, and ensuring the code's functionality.

In this guide, we will walk through the steps to set up a CI/CD pipeline for your Android Studio project using CircleCI and GitHub. This setup will automate building, testing, and deploying our Android app whenever you push code to your GitHub repository.

**Prerequisites:**

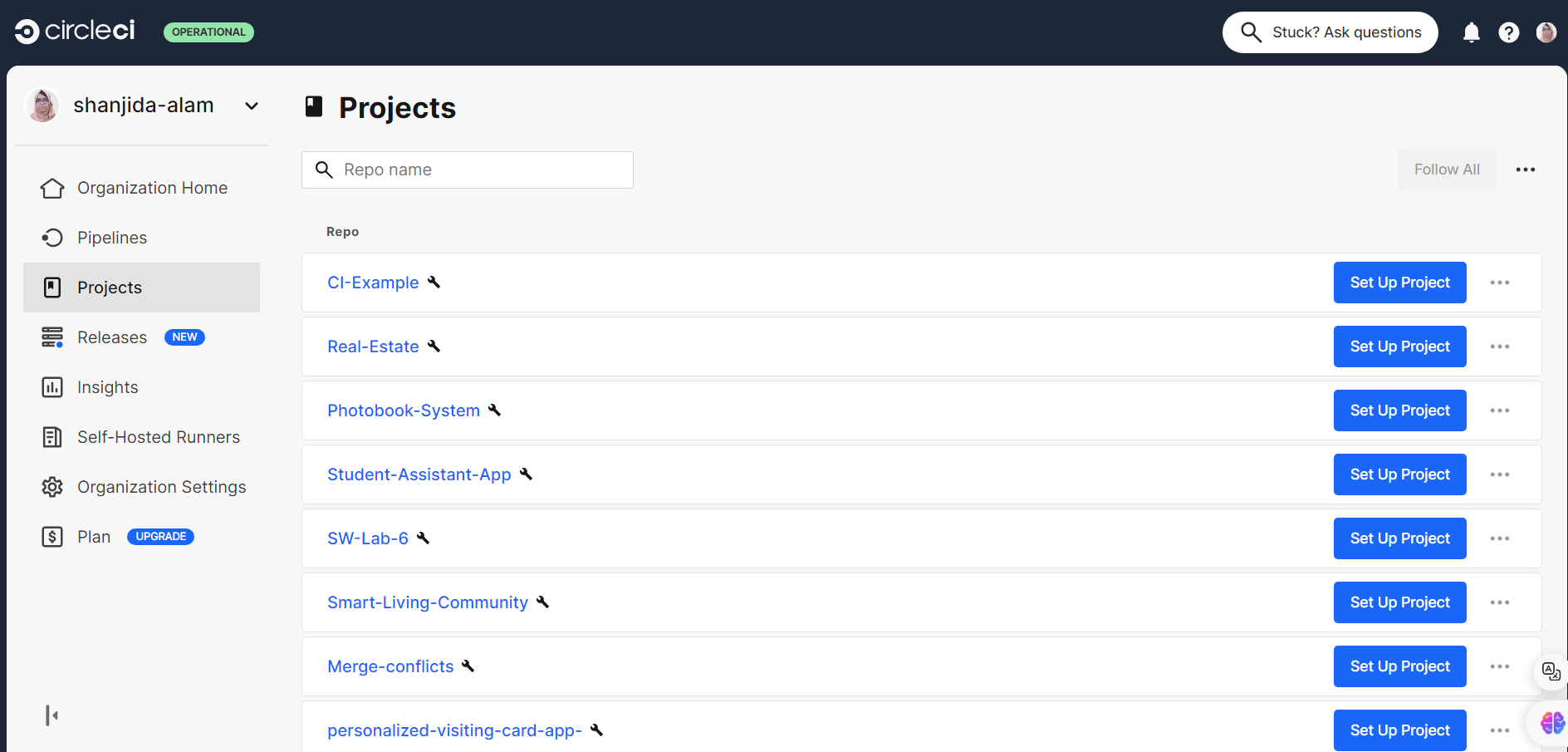
1. A GitHub account and repository for your Android project.
2. CircleCI account connected to your GitHub account.
3. Android Studio with a project ready for integration.

**Step 1: Create a GitHub Repository and Push Android Project**

* + Create a new GitHub repository for your android project.
  + Push your android project to GitHhub.

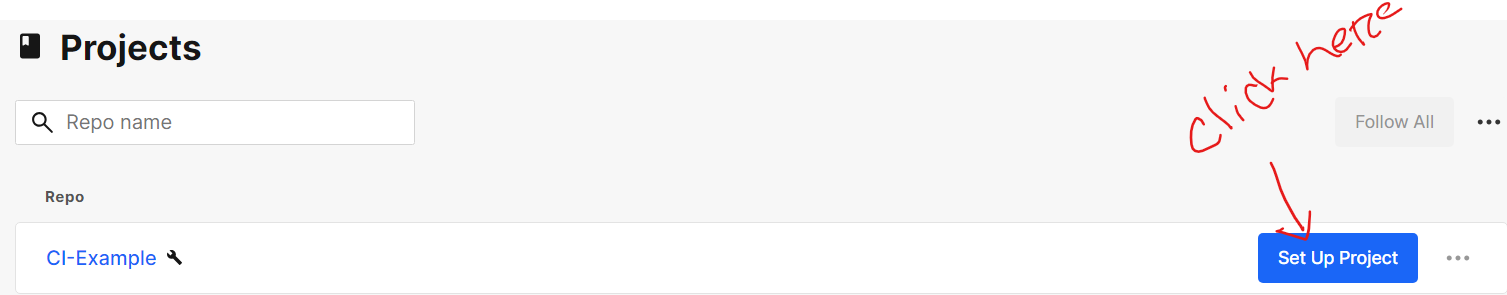
**Step 2: Create a CircleCI Account and Link Your GitHub Repo**

1. Sign Up for CircleCI
2. Link GitHub repository to CircleCI
   1. Once logged in, click Projects in CircleCI.



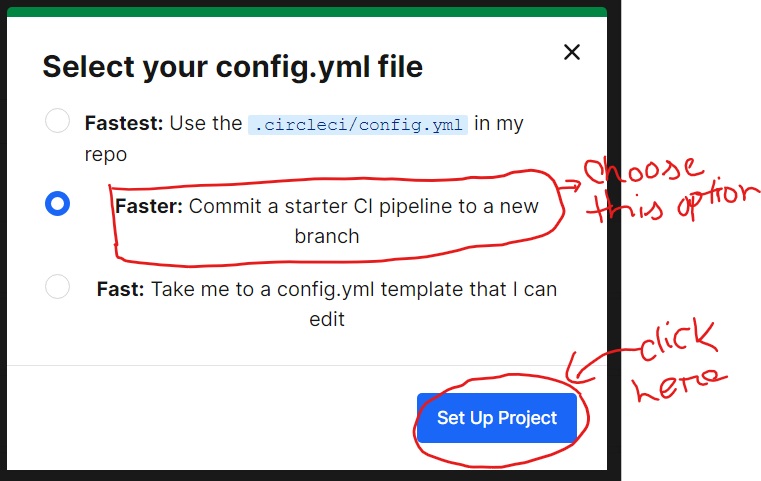
**Figure 2: CircleCI dashboard showing a list of projects.**

* 1. Click **Set Up Project** next to your Android repository.



**Figure 3: Click 'Set Up Project' to start your CI/CD workflow on CircleCI.**

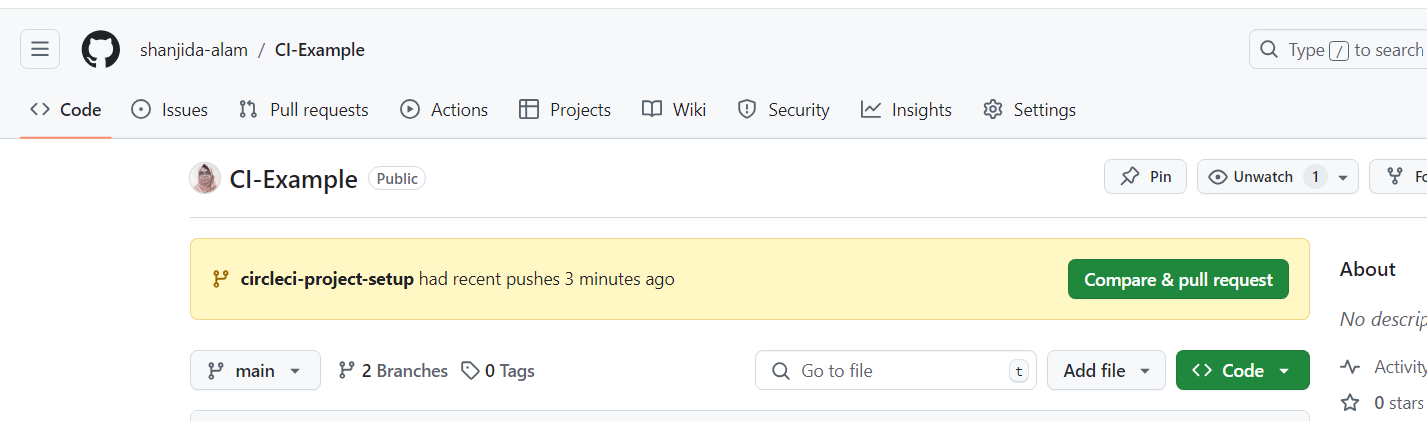
* 1. CircleCI will suggest creating a .circleci/config.yml file.

****

**Figure 4:Configuring your project on CircleCI: select the faster option.**

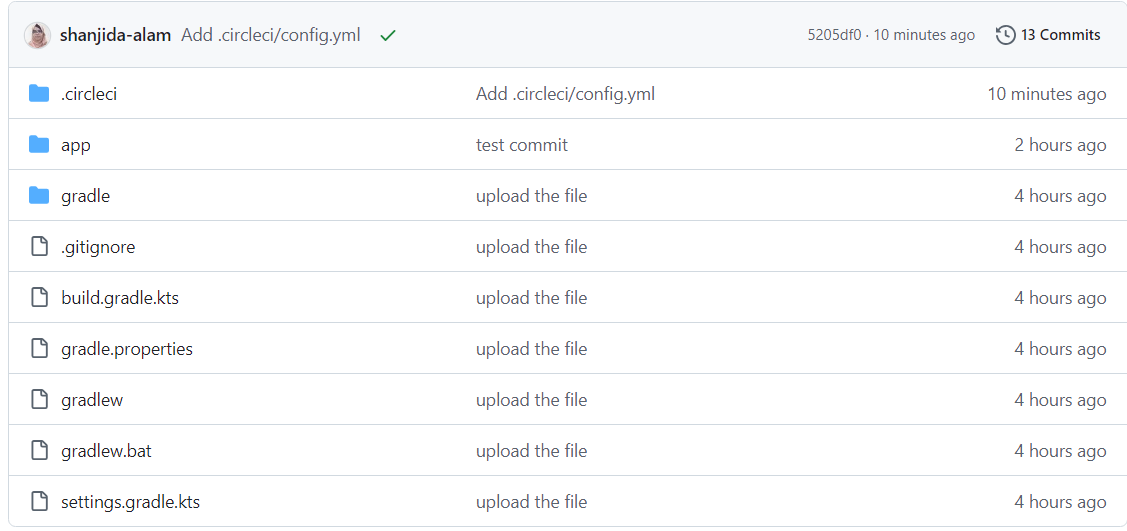
**Step 3: Create a CircleCI Configuration File (config.yml)**

The CircleCI pipeline for Android Studio projects is defined by a .circleci/config.yml file in the root of your project. This file tells CircleCI how to build, test, and deploy your Android app.



**Figure 5:Viewing recent activity on the circleci-project-setup branch after creating a config.yml file**

* 1. **Creating the .circleci Folder:**
* In the root directory of your Android project, create a folder named .circleci.
* Inside that folder, create a file named config.yml.

****

**Figure 6:Recent commit on the circleci-project-setup branch: adding the .circleci/config.yml file.**

* 1. **Add a Basic Android CI Configuration:**
* **Pull the latest code** in your android project and then update the config.yml with the following code.
* Then, **switch the new branch** where introduced the .circleci/config file.

**‘git checkout circleci-project-setup’**

Here’s a simple config.yml file that you can start with:

version: 2.1  
  
executors:  
 android-executor:  
 docker:  
 - image: circleci/android:api-30  
 working\_directory: ~/project  
  
jobs:  
 build:  
 executor: android-executor  
 steps:  
 - checkout  
  
 *# Install SDKMAN! to manage JDK versions* - run:  
 name: Install SDKMAN  
 command: |  
 curl -s "https://get.sdkman.io" | bash  
 source "$HOME/.sdkman/bin/sdkman-init.sh"  
  
 *# Install Java 17 via SDKMAN and set it as the default* - run:  
 name: Install JDK 17  
 command: |  
 source "$HOME/.sdkman/bin/sdkman-init.sh"  
 sdk install java 17.0.8-zulu  
 sdk use java 17.0.8-zulu # Explicitly use Java 17  
  
 *# Set JAVA\_HOME to point to Java 17* - run:  
 name: Set JAVA\_HOME to Java 17  
 command: |  
 echo 'export JAVA\_HOME=$HOME/.sdkman/candidates/java/current' >> $BASH\_ENV  
 source $BASH\_ENV  
  
 *# Verify Java 17 is the active version* - run:  
 name: Verify Java Version  
 command: java -version  
  
 *# Set executable permission for gradlew* - run:  
 name: Set executable permission for gradlew  
 command: chmod +x ./gradlew  
  
 *# Download dependencies* - run:  
 name: Download Dependencies  
 command: ./gradlew dependencies  
  
 *# Build the APK* - run:  
 name: Build APK  
 command: ./gradlew assembleDebug  
  
 *# Store the APK as an artifact* - store\_artifacts:  
 path: app/build/outputs/apk/debug/app-debug.apk  
 destination: app-debug.apk  
  
 test:  
 executor: android-executor  
 steps:  
 - checkout  
  
 *# Install SDKMAN! to manage JDK versions* - run:  
 name: Install SDKMAN  
 command: |  
 curl -s "https://get.sdkman.io" | bash  
 source "$HOME/.sdkman/bin/sdkman-init.sh"  
  
 *# Install Java 17 via SDKMAN and set it as the default* - run:  
 name: Install JDK 17  
 command: |  
 source "$HOME/.sdkman/bin/sdkman-init.sh"  
 sdk install java 17.0.8-zulu  
 sdk use java 17.0.8-zulu # Explicitly use Java 17  
  
 *# Set JAVA\_HOME to point to Java 17* - run:  
 name: Set JAVA\_HOME to Java 17  
 command: |  
 echo 'export JAVA\_HOME=$HOME/.sdkman/candidates/java/current' >> $BASH\_ENV  
 source $BASH\_ENV  
  
 *# Verify Java 17 is the active version* - run:  
 name: Verify Java Version  
 command: java -version  
  
 *# Set executable permission for gradlew* - run:  
 name: Set executable permission for gradlew  
 command: chmod +x ./gradlew  
  
 *# Run unit tests* - run:  
 name: Run Unit Tests  
 command: ./gradlew testDebug  
  
 *# Store test results* - store\_test\_results:  
 path: app/build/test-results/testDebugUnitTest  
  
workflows:  
 version: 2  
 build\_and\_test:  
 jobs:  
 - build  
 - test

**Step 4: Commit and Push Changes**

* 1. **Commit the new CircleCI configuration:**

git add .circleci/config.yml

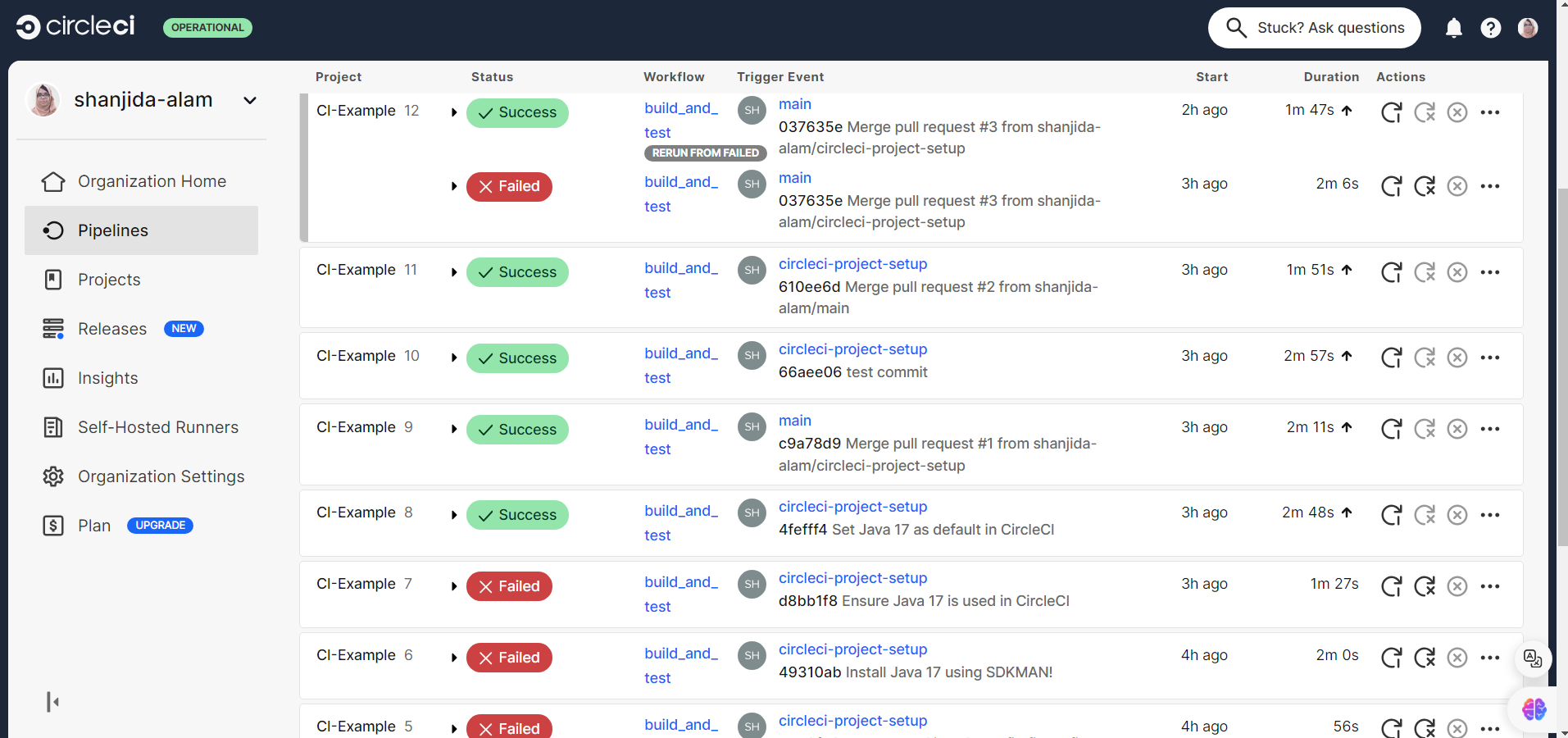
git commit -m "Add CircleCI config for Android project"

git push origin main

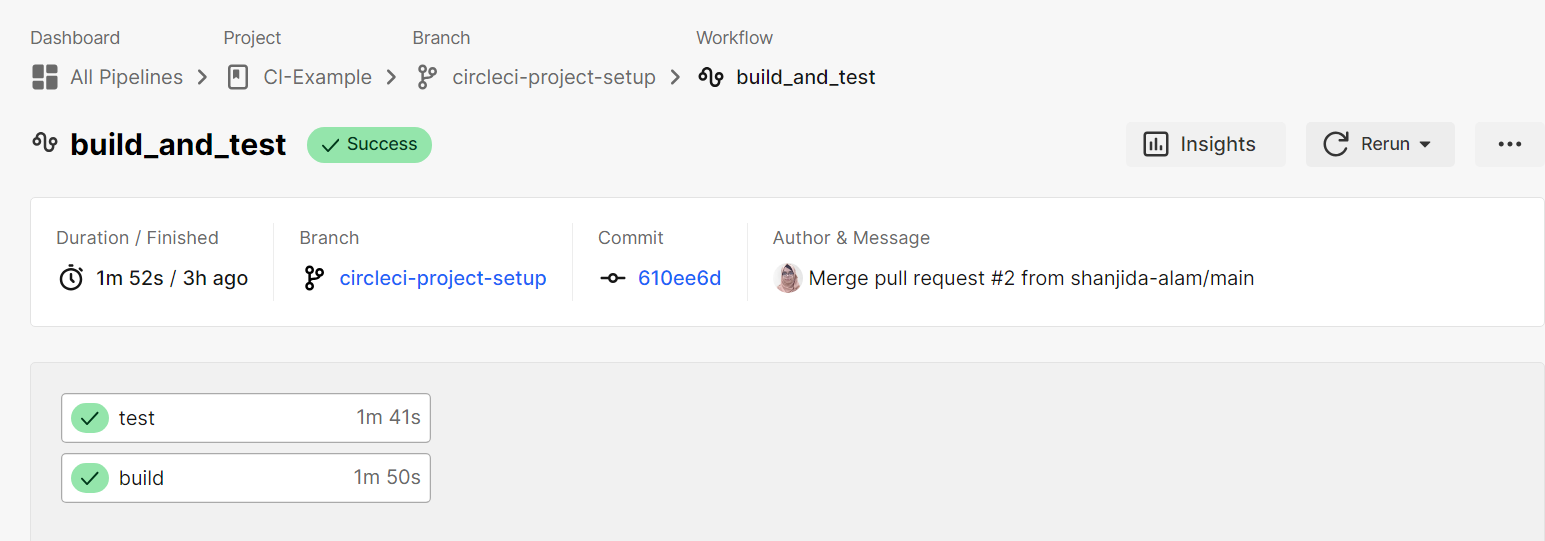
Once the configuration is pushed to your repository, CircleCI will detect it and start the build process automatically.

**Step 5: Monitor the Build on CircleCI**

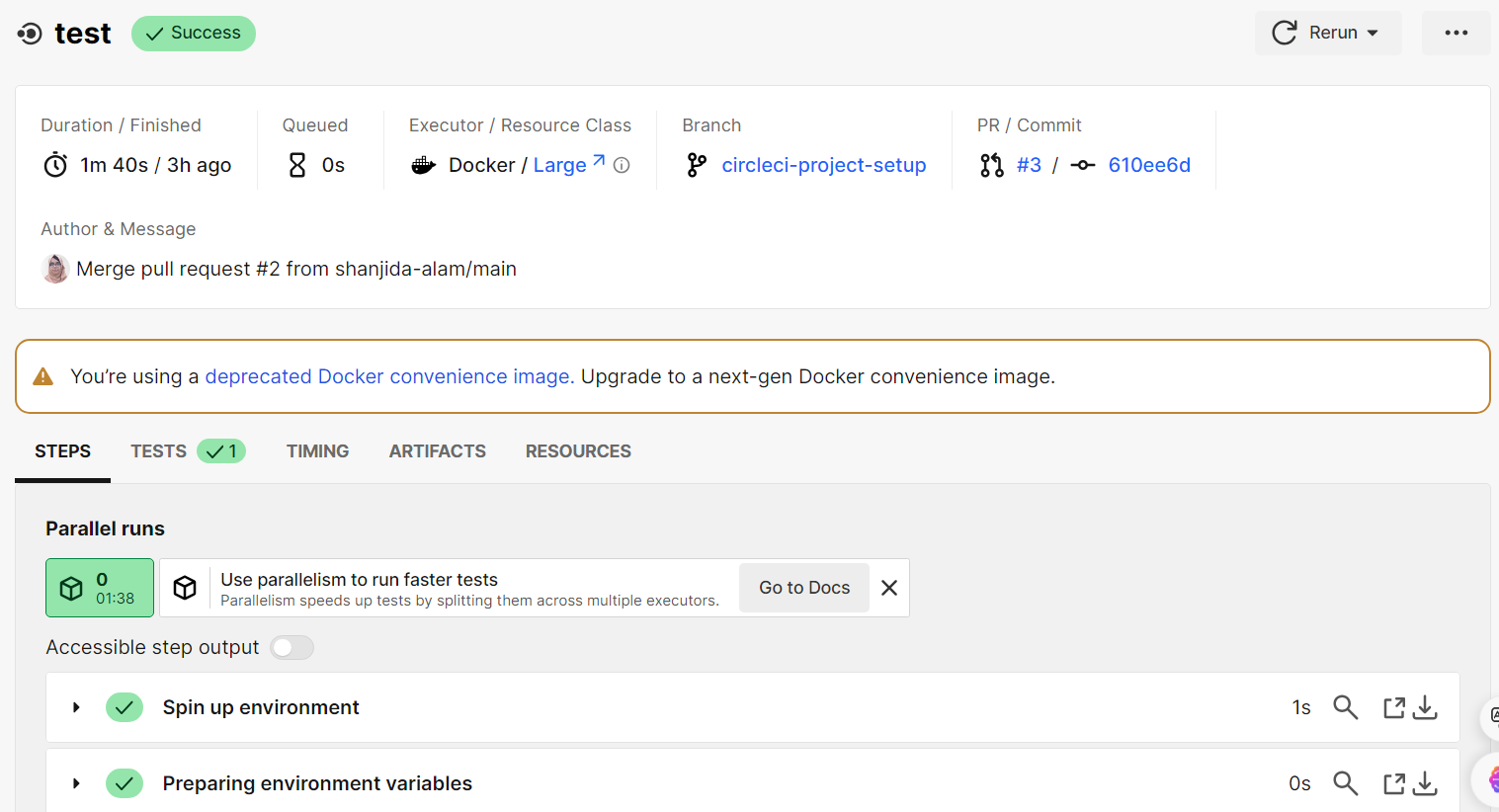
Go to your CircleCI dashboard, where you can see the pipeline running based on your .circleci/config.yml file.

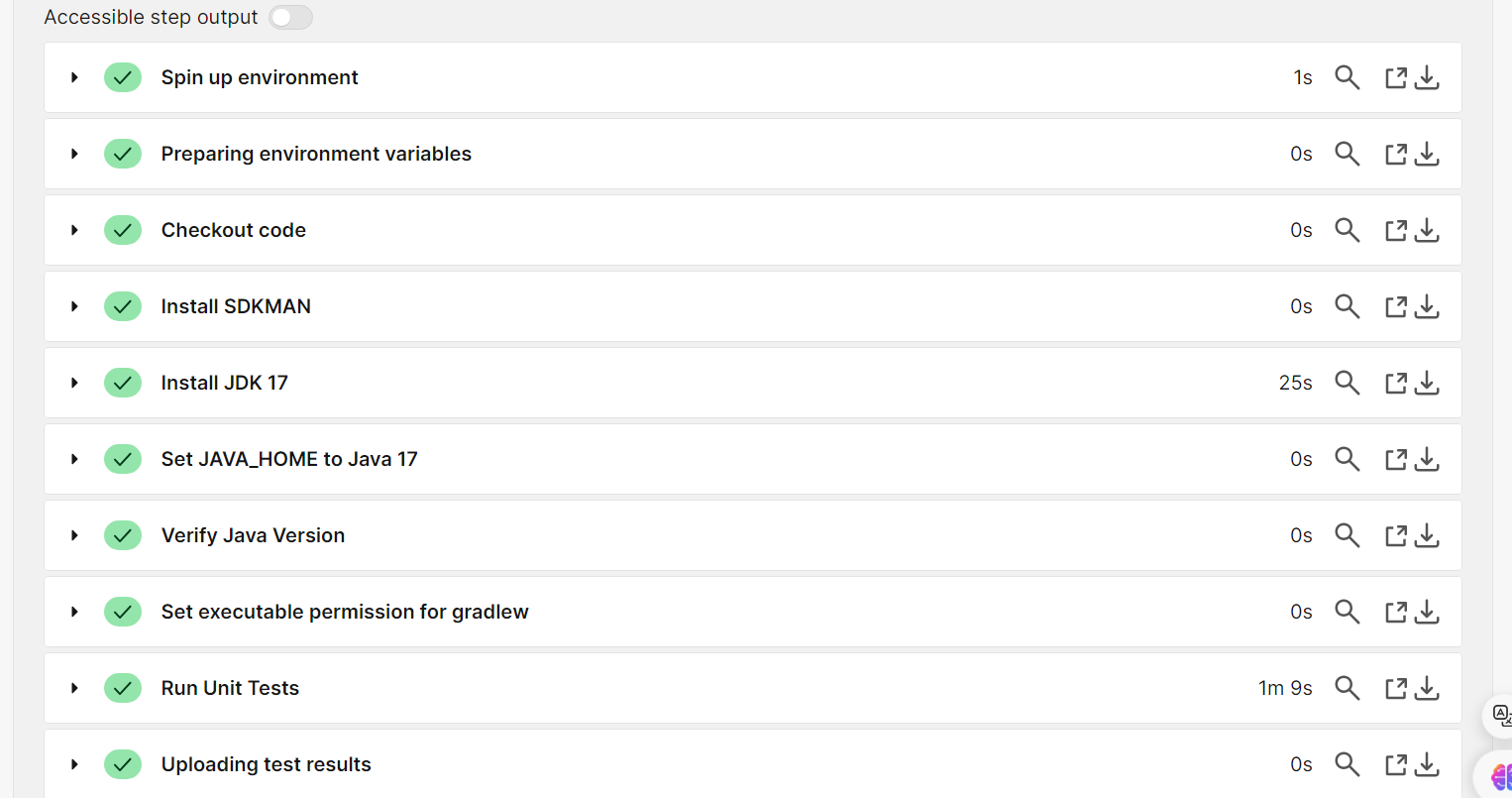


**Figure 7:Tracking recent build and test results on CircleCI.**

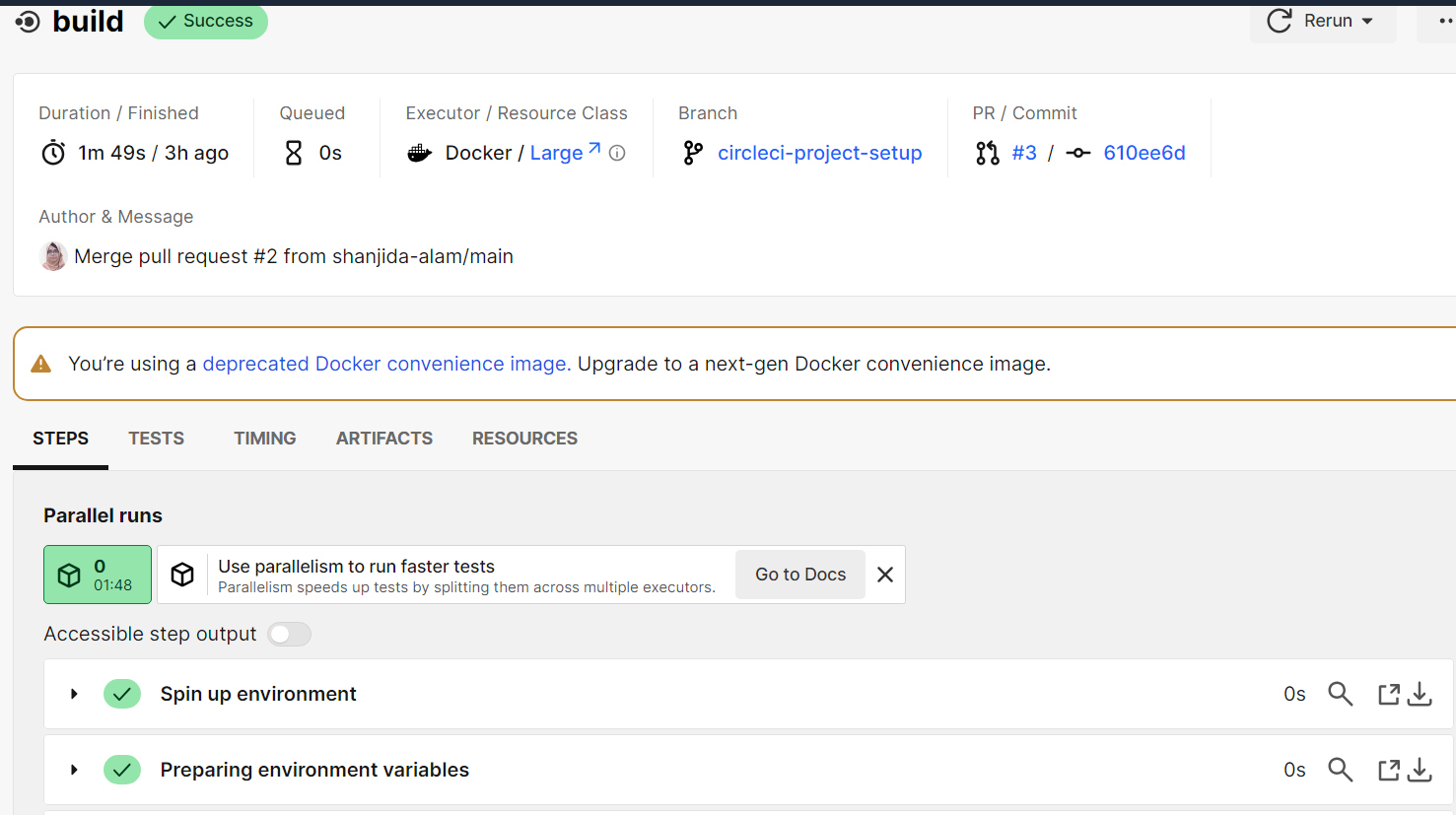


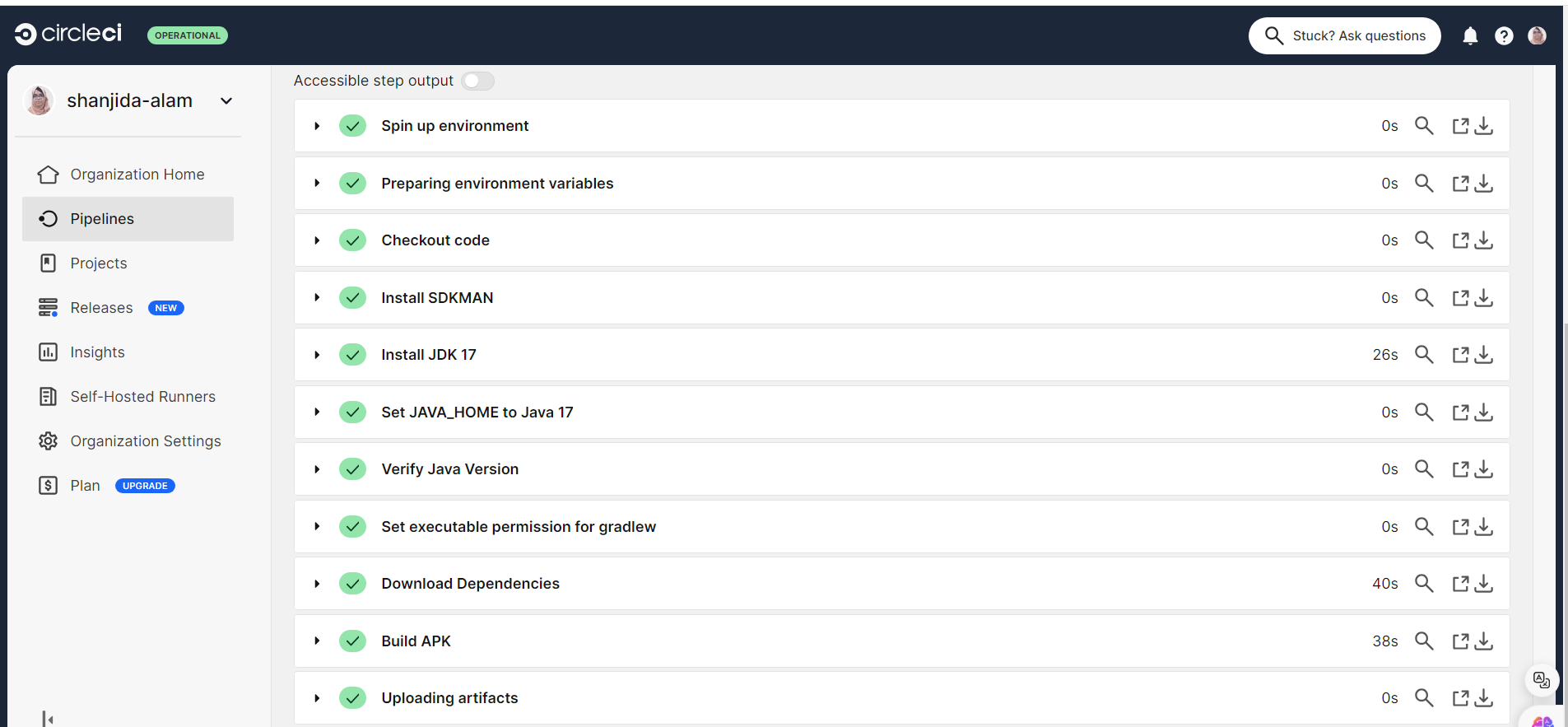
**Figure 8: CircleCI pipeline details for the build\_and\_test workflow.**





**Figure 9:Viewing detailed step output for a successful CircleCI pipeline.**





**Figure 10:Successful CircleCI Pipeline Execution for Android Build Process**

1. **References:**

* [**https://circleci.com/continuous-integration/**](https://circleci.com/continuous-integration/)
* [**https://youtu.be/qtJqTbf6l18?si=I5hta4q7WLn2gUo-**](https://youtu.be/qtJqTbf6l18?si=I5hta4q7WLn2gUo-)
* [**https://circleci.com/docs/**](https://circleci.com/docs/)