UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

Practical 8 : Building Application in CLI (Gradle)

Gradle is a build automation tool for multi-language software development. Gradle build on concepts of Apache Ant and Apache Maven. Unlike Maven, which faces some problems in dependency management and does not handle well conflicts between versions of the same library, Gradle became the first build integration tool which fully supported ANT tasks, Maven and lvy repository infrastructure for publishing and retrieving dependencies. In this practical, usage of Gradle in build automation for JAVA program is explored.

Preliminaries.

Firstly, verify Java installation in workstation; ensure Java Software Development Kit is installed with the `java -version` command in CLI as shown in Figure 1.

Figure 1: Ensure Java installation with Java version command.

Secondly, download Gradle build file from the following website:
https://services.gradle.org/distributions/gradle-6.5.1-all.zip
Extract the downloaded zip file in **C:\Gradle** (create the directory).
Then, add **C:\Gradle** and **C:\Gradle\bin** directories to the GRADLE_HOME and PATH system variables.
Follow the given instruction:

1) Right click on “My Computers” or “This PC”
2) Click on “Properties”
3) Choose “Advanced system settings” on left navigation pane
4) Click on “Environmental Variables...” button.

AThere you will find a dialog box will pop up for creating and editing system variables. Then, click on “New...” button for creating GRADLE_HOME variable (follow the left side screenshot in Figure 2). Click on “Edit...” for editing the existing Path system variable (follow the right side screenshot in Figure 2).

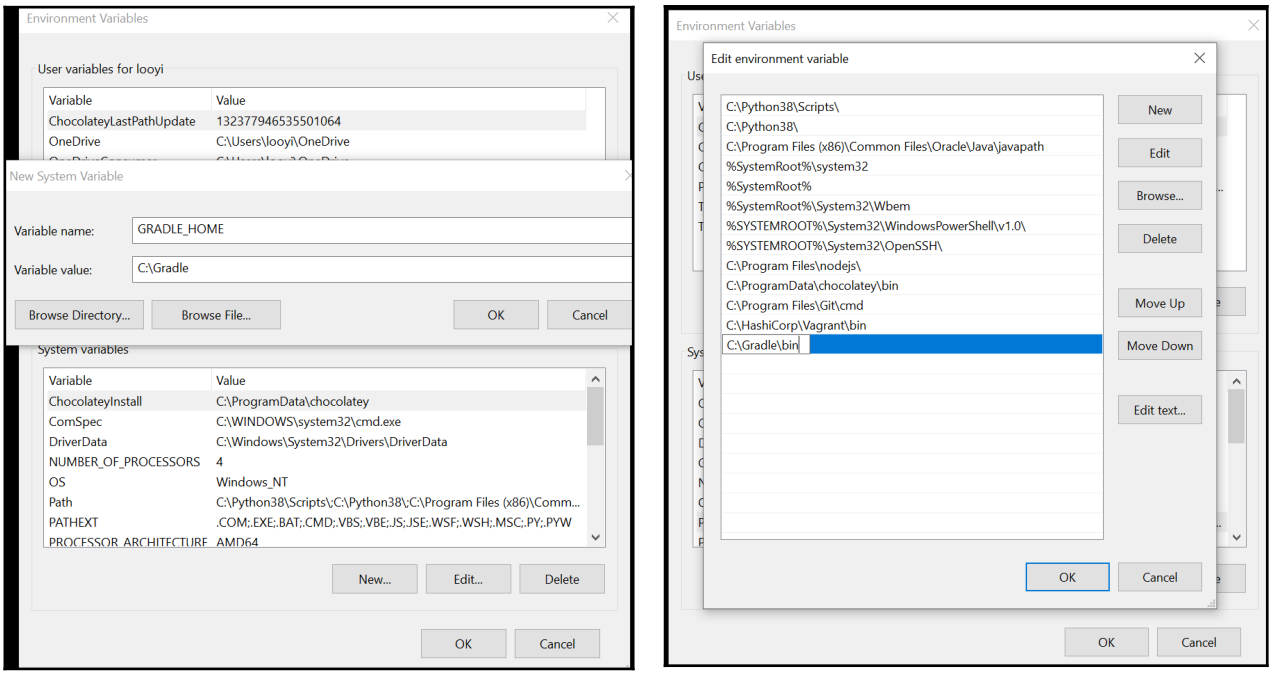

Figure 2: Set GRADLE_HOME and edit PATH variable.

Lastly, verify Gradle installation by executing `gradle -v` command in CLI as shown in Figure 3.

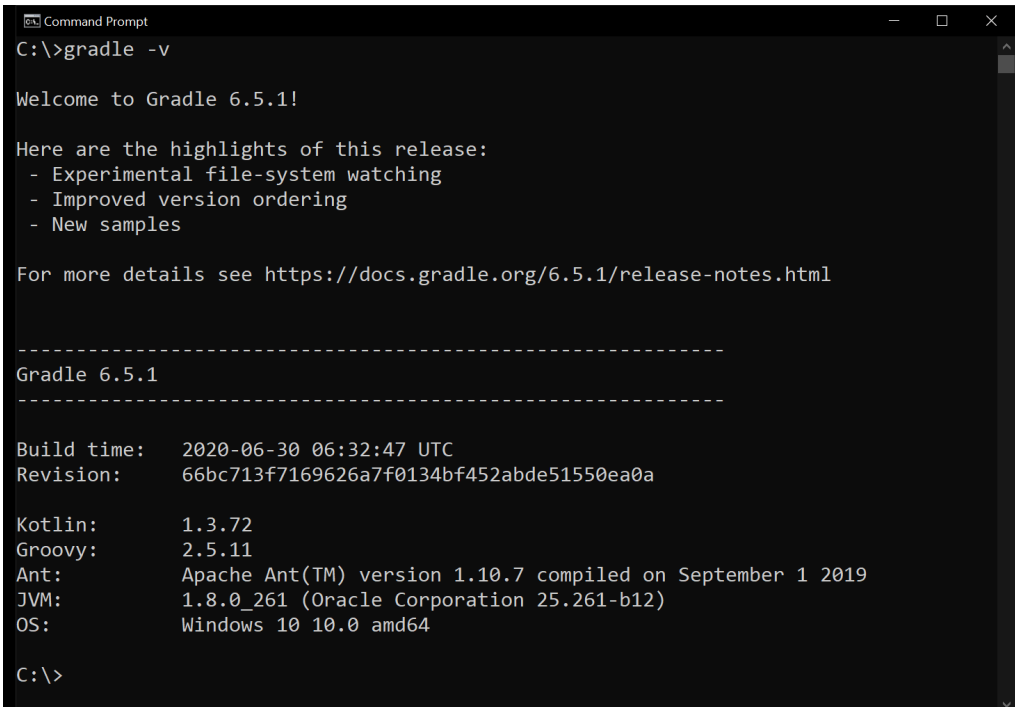

Figure 3: Verify Gradle installation.

Getting Started.

The following fundamental practical will explore on initiation of a Gradle project, invoke some of the basic Gradle commands, and get a sense of how Gradle manages the project. In order to initiate a Gradle project, ensure that a new folder is created for this exploration. Then, within the folder, execute `gradle init` command to initialize a new project as shown in Figure 4.

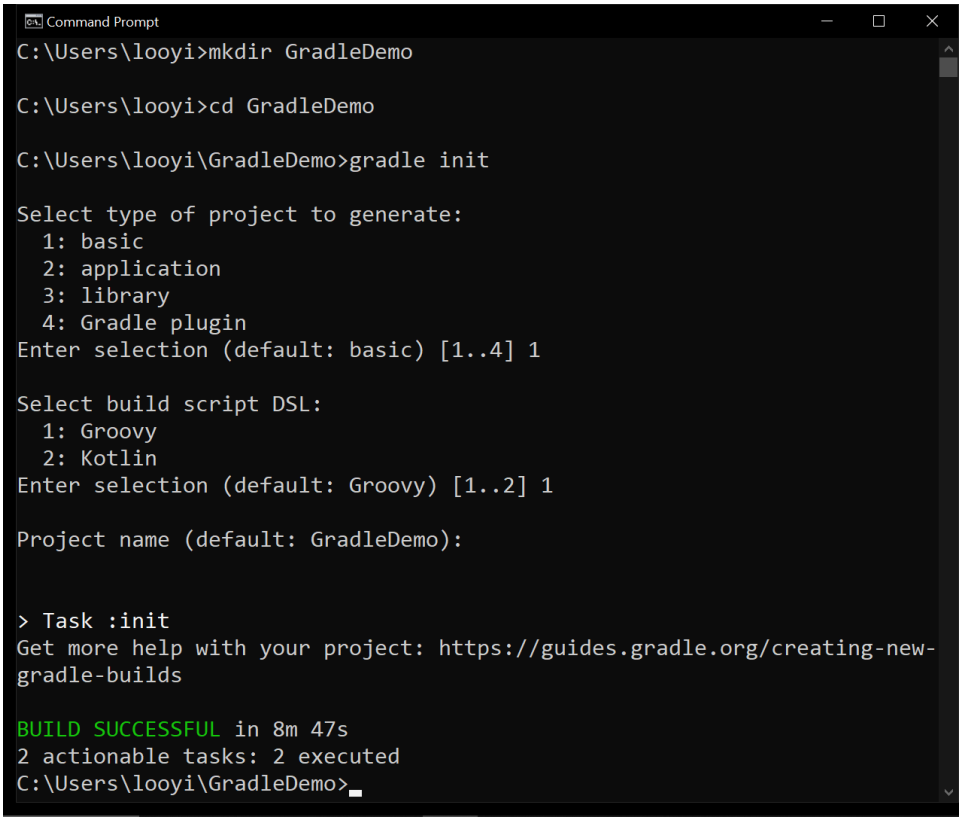

Figure 4: Initialize a Gradle project.

The initialization of Gradle project creates the following files and folders:

1. build.gradle – Gradle build script for configuring the current project
2. gradle-wrapper.jar – Gradle Wrapper executable JAR
3. gradle-wrapper.properties – Gradle Wrapper configuration properties
4. gradlew – Gradle Wrapper script for Unix-based systems
5. gradlew.bat – Gradle Wrapper script for Windows
6. settings.gradle – Gradle settings script for configuring the Gradle build

Create Gradle Builds.

Gradle provides APIs for creating and configuring tasks through a Groovy or Kotlin-based DSL. A project consists of a collection of tasks; each task performs some basic operations. Gradle comes with a library of tasks that one can configure for the project. For example, there is a core type called "Copy", which copies files from one location to another.

Create a folder named "src" and a file inside the folder named "demoFile.txt". For the txt file, just put in a line, for instance "Hello, World!" as shown in Figure 5.

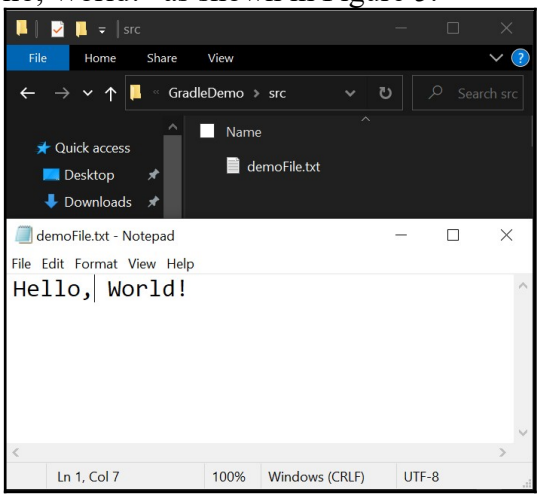

Figure 5: Create a new file in the newly created src folder.

Then, in build.gradle file, define a task called "copy" of type "Copy" (take note on the capital letter) in the build file which copies the "src" directory to a new directory called "copyDest" as shown in Figure 6. *Note: Don't have to create the copyDest directory; the task will automate that.*

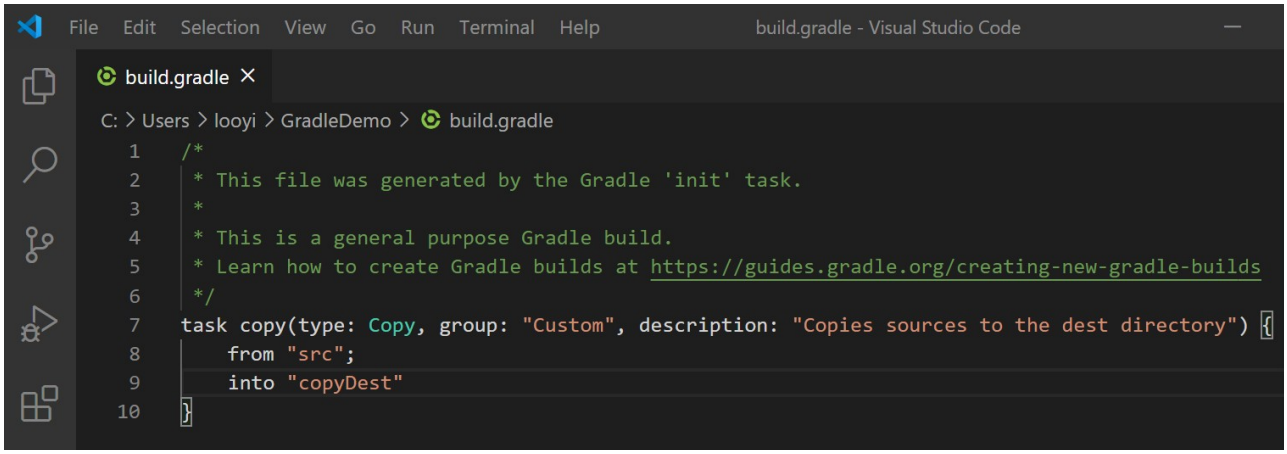

Figure 6: Insert copy task in Gradle build.

Upon saving the Gradle build file, execute the task by executing `gradlew copy` command. Then confirm that the "copyDest" folder contains the same content as "src" folder as shown in Figure 7.

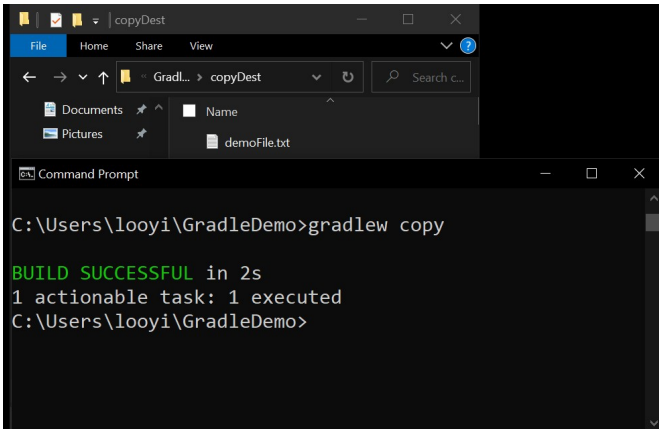

Figure 6: Copy task executed in Gradle build.

Now let's try a different task; Zip task. In order to enable Gradle to automate the task of zipping a project insert the codes into Gradle build file as shown in Figure 7.

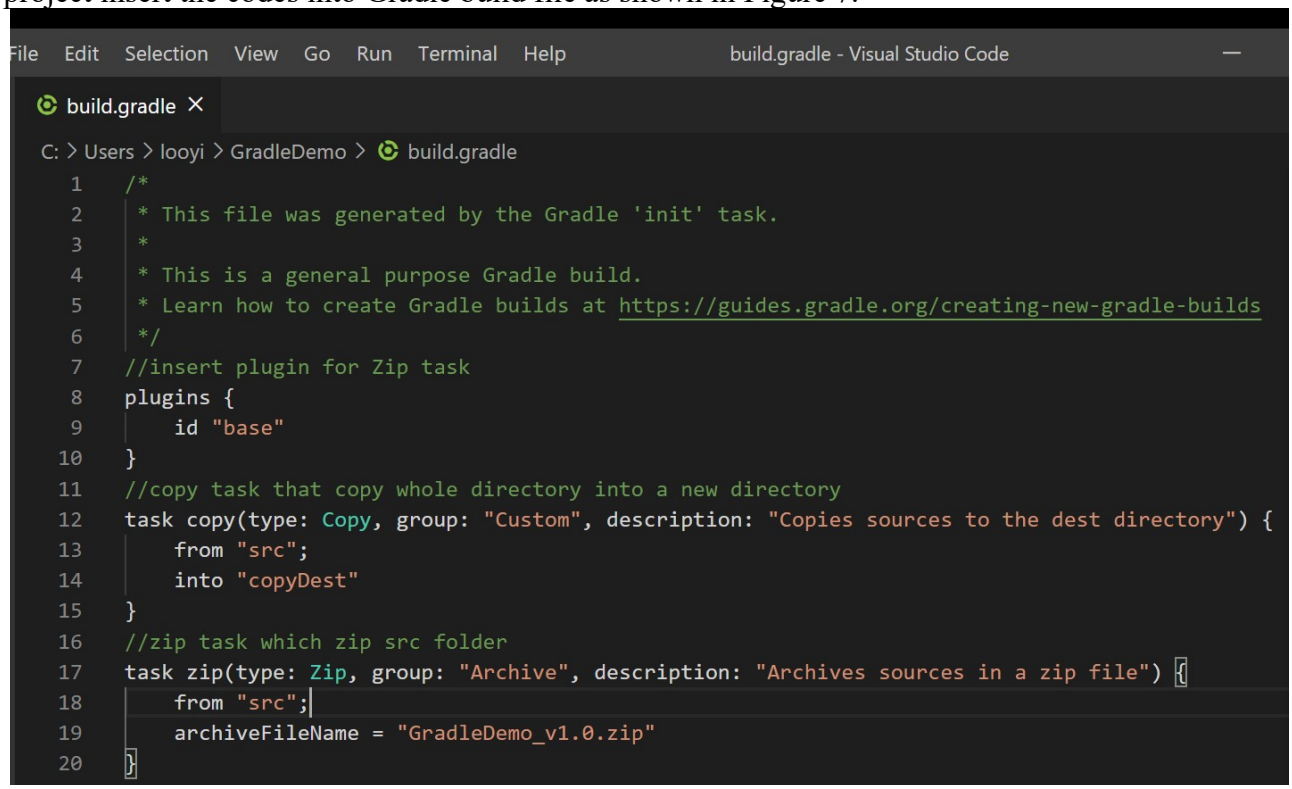

Figure 7: Adding another task into Gradle build file.

Then, execute `gradlew zip` and see that GradleDemo_v1.0.zip had been created with files of src folder into ./build/distrbutions as shown in Figure 8.

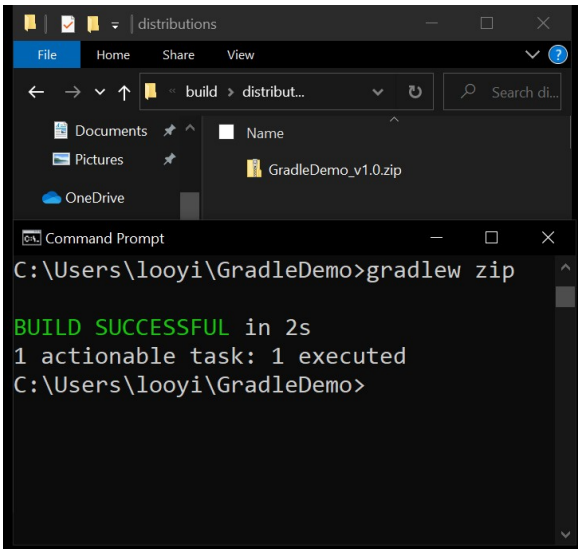

Figure 8: Execute Zip task in Gradle build.

To discover executable tasks in Gradle, execute `gradlew tasks` to list all the available tasks as shown in Figure 9.

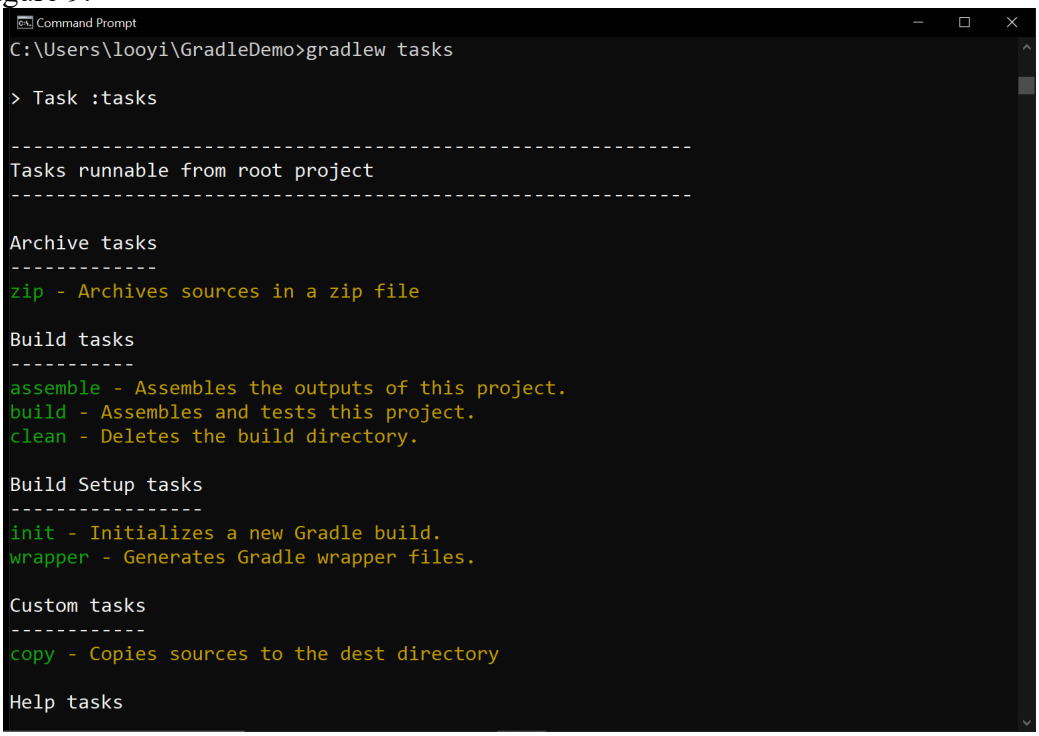

Figure 9: List of tasks in Gradle build.

Create Build Scans.

A build scan is a shareable and centralized record of a build that provides insights into what happened in the build and why. By applying the build scan plugin to the project, one can publish build scans to https://scans.gradle.com for free.

Clone or download the sample project from: https://github.com/gradle/gradle-build-scan-quickstart in order to see the potential of build scans.

In this practical, the project is cloned as shown in Figure 10.

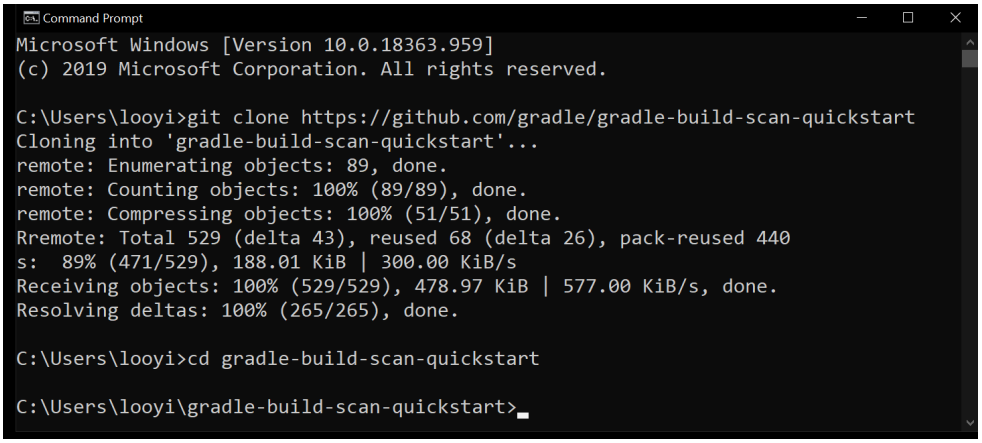

Figure 10: Clone sample project for build scans.

Then, execute `gradlew build --scan` in order to generate a record of the build of this sample project. Note: Starting from Gradle 4.3, build scan plugin is automatically installed prior to the `--scan` option of `build` command. The build scan will be generated and one can access the build scan with the url link provided in CLI.

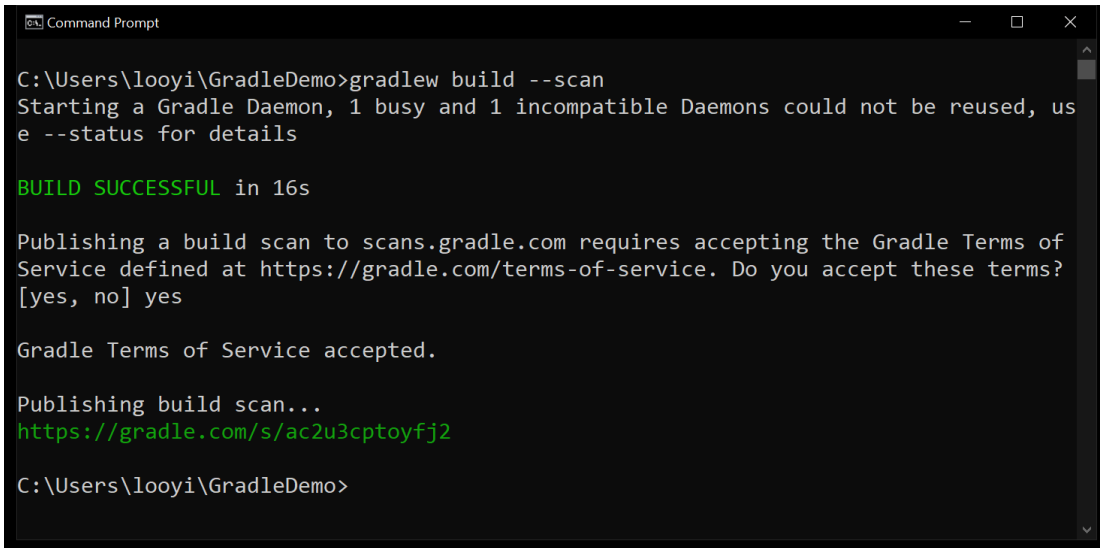

Figure 11: Successful build of sample project with published build scan.

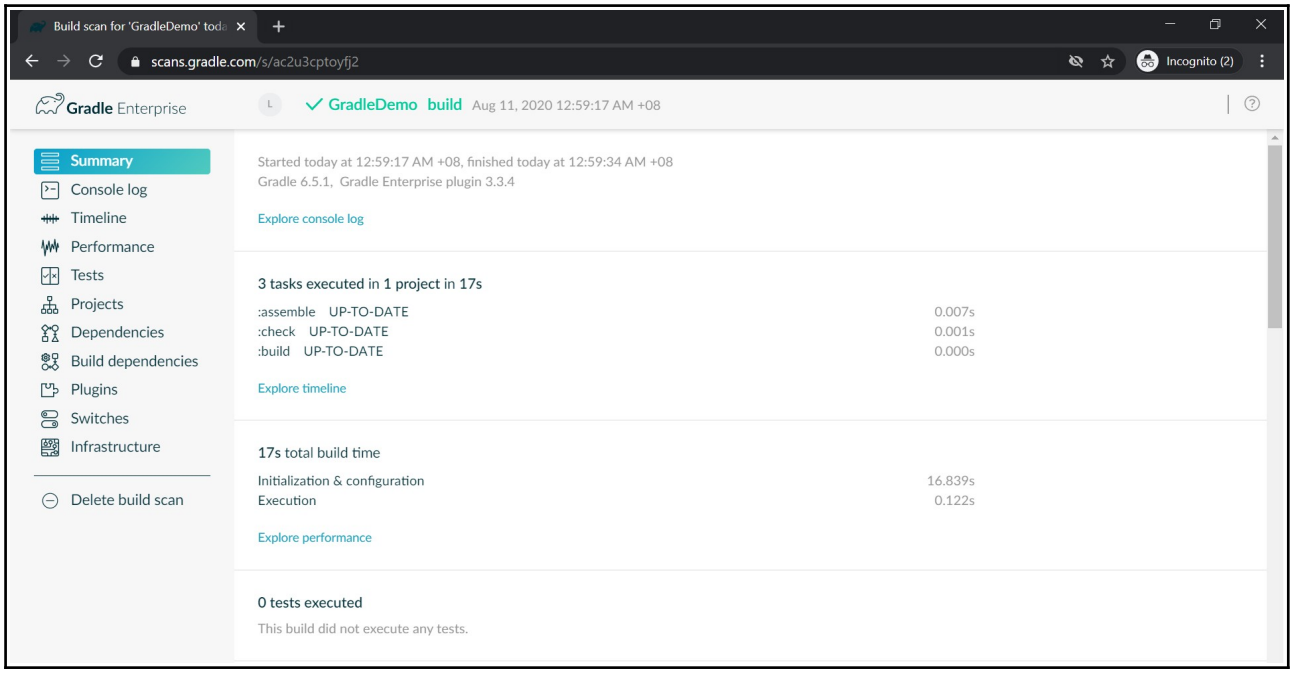

Figure 12: Published build scan in Gradle Enterprise website.