What is Gradle ?

- ❑ **Gradle** is an open source **build automation** tool that is based on the concept of **Apache Maven** and **Ant**.
- ❑ It introduces a **Kotlin and Groovy-based DSL(Domain Specific Language)** instead of XML (Extensible Markup Language) for declaring the project configuration.
- ❑ Gradle offers an elastic model that can help the development lifecycle from compiling and packaging code for web and mobile applications.
- ❑ It provides support for the **building**, **testing**, and **deploying software** on different platforms.
- ❑ It has been developed for building automation on many languages and platforms, including Java, Scala, Android, C / C ++, and Groovy. Gradle provides integration with several development tools and servers, including Eclipse, IntelliJ, Jenkins, and Android Studio.
- ❑ Some Leading Enterprise companies like **LinkedIn, Google** and **Netflix** use Gradle
- ❑ It is developed to **overcome the drawbacks of Maven and Ant** and supports a wide range of IDEs.

Gradle vs. Maven

Gradle

Maven	Gradle
It is a software project management system that is primarily used for java projects	It is a build automation system that uses a Groovy or Kotlin based DSL (domain-specific language)
It uses an XML file for declaring the project, its dependencies, the build order, and its required plugin.	It does not use an XML file for declaring the project configuration.
It is based on the phases of the fixed and linear model.	It is based on a graph of task dependencies that do the work.
It does not use the build cache; thus, its build time is slower than Gradle.	It avoids the work by tracking input and output tasks and only runs the tasks that have been changed. Therefore it gives a faster performance.
Maven has a limited number of parameters and requirements, so customization is a bit complicated.	Gradle is highly customizable; it provides a wide range of IDE support custom builds.

Why to Use Gradle ?

The following is the list of features that Gradle provides:

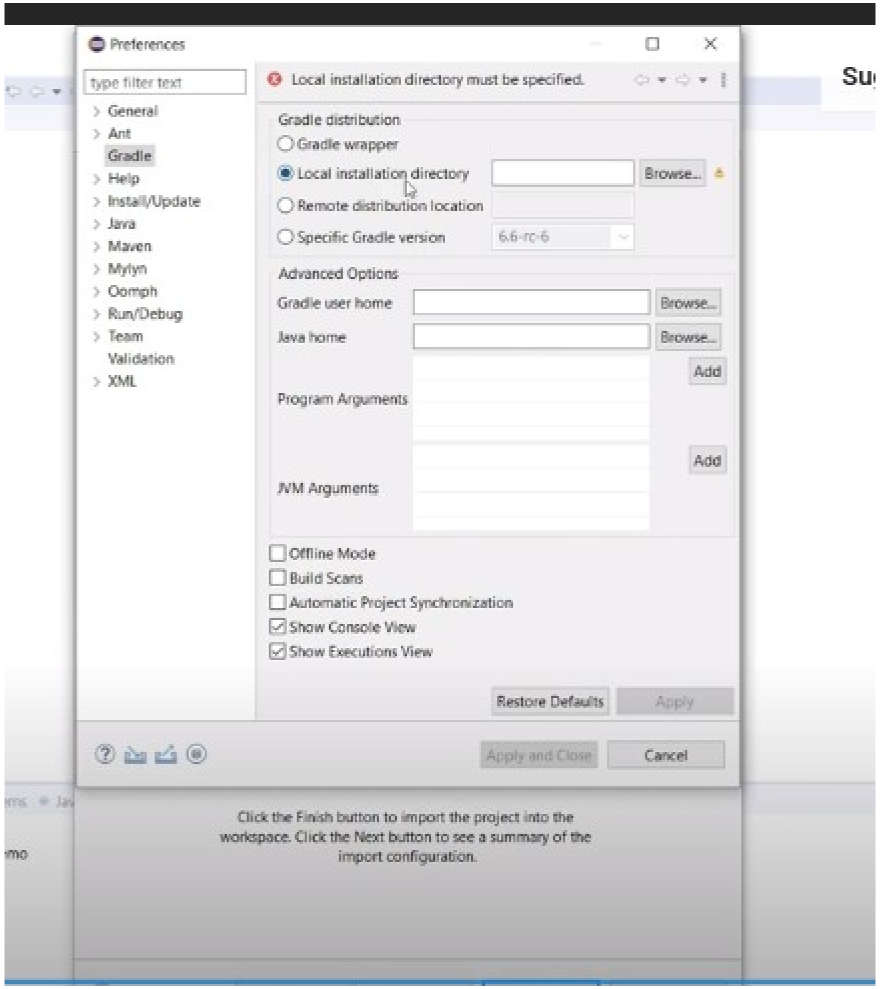
- Free and open-source
- High Performance
- Highly Customizable
- Performance
- Flexibility
- Multi-project build support
- Extensibility
- Incremental Builds
- Familiar with the Java
- Gradle Wrapper
- User Experience

Why to Use Gradle ?

- Free and open-source
  - Gradle is an open source project, and licensed under the Apache Software License (ASL).
- High Performance
  - Quickly completes the task by reusing the output from the previous execution.
  - Processes tasks whose only input is changed and executes the task in parallel.
- Performance
  - Faster than Maven in all scenarios including large builds using build-cache.

Why to Use Gradle ?

- Multi-project build support
  - Powerful support for the multi-project builds
- Extensibility
  - Easily extend the Gradle to provide our task types or build models.
- Incremental Builds
  - If we compile source code, it will check if the sources have changed since the previous execution. If the code is changed, then it will be executed but if the code is not changed, then it will skip the execution, and the task is marked as updated.
- Familiar with the Java
  - We need a JVM to run the Gradle, so our machine should have a Java Development Kit (JDK).
  - Bonus for developers as we can use the standard Java APIs in our build logic



Preferences

Local installation directory must be specified.

Gradle distribution

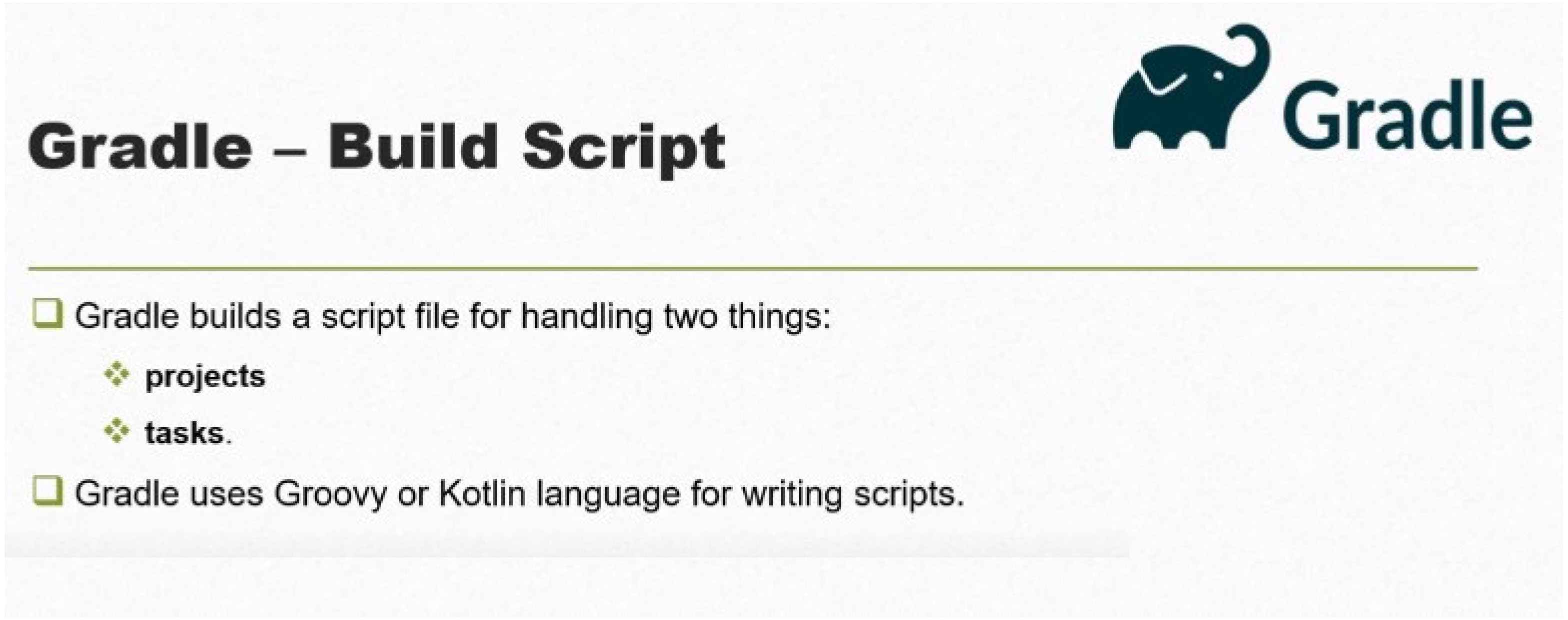
( ) Gradle wrapper
(●) Local installation directory [ ] Browse...
( ) Remote distribution location
( ) Specific Gradle version 6.6-rc-6

Advanced Options

Gradle user home [ ] Browse...
Java home [ ] Browse...
Program Arguments Add
JVM Arguments Add

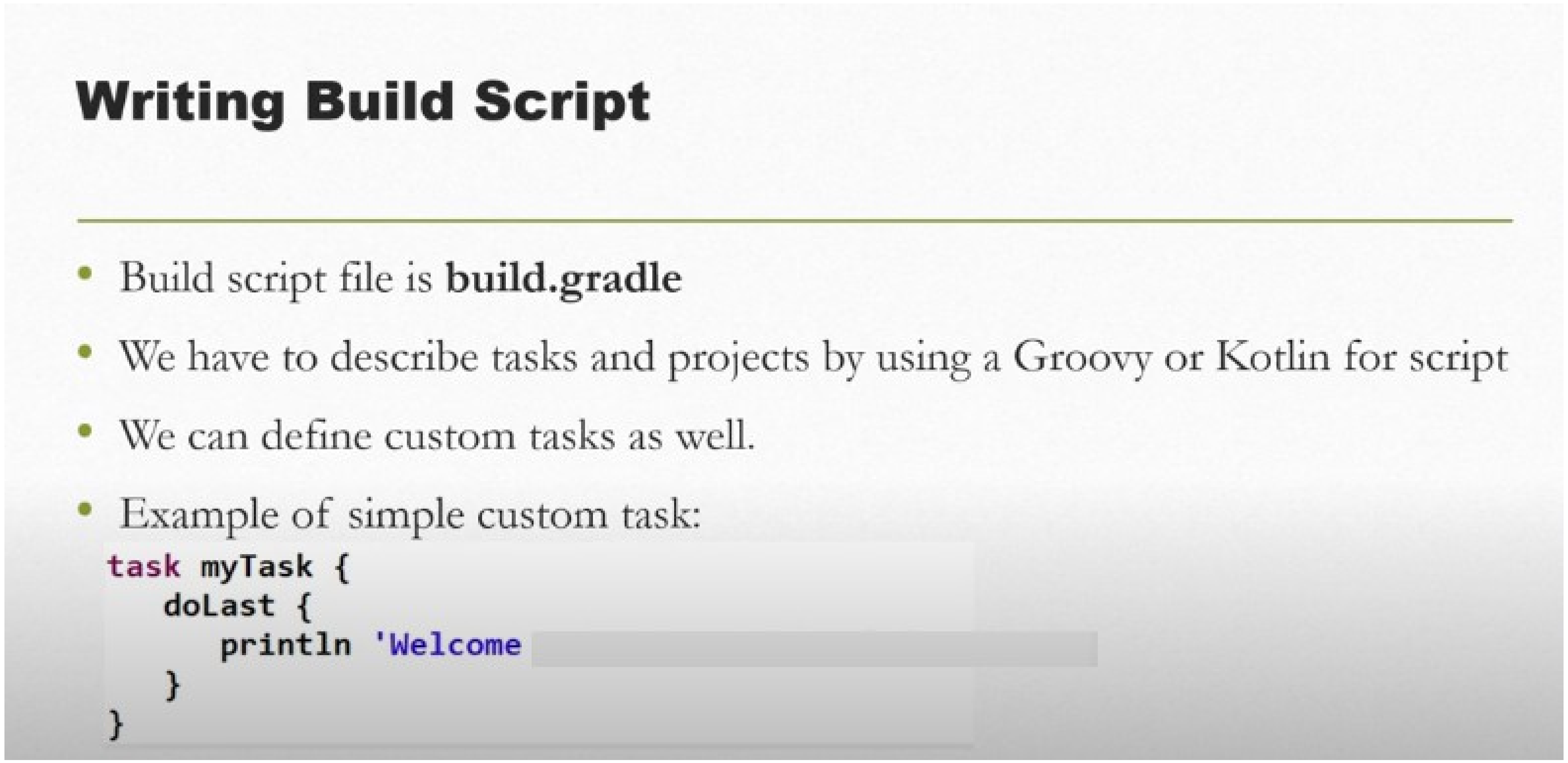
[ ] Offline Mode
[ ] Build Scans
[ ] Automatic Project Synchronization
[✓] Show Console View
[✓] Show Executions View

Click the Finish button to import the project into the workspace. Click the Next button to see a summary of the import configuration.



Gradle – Build Script

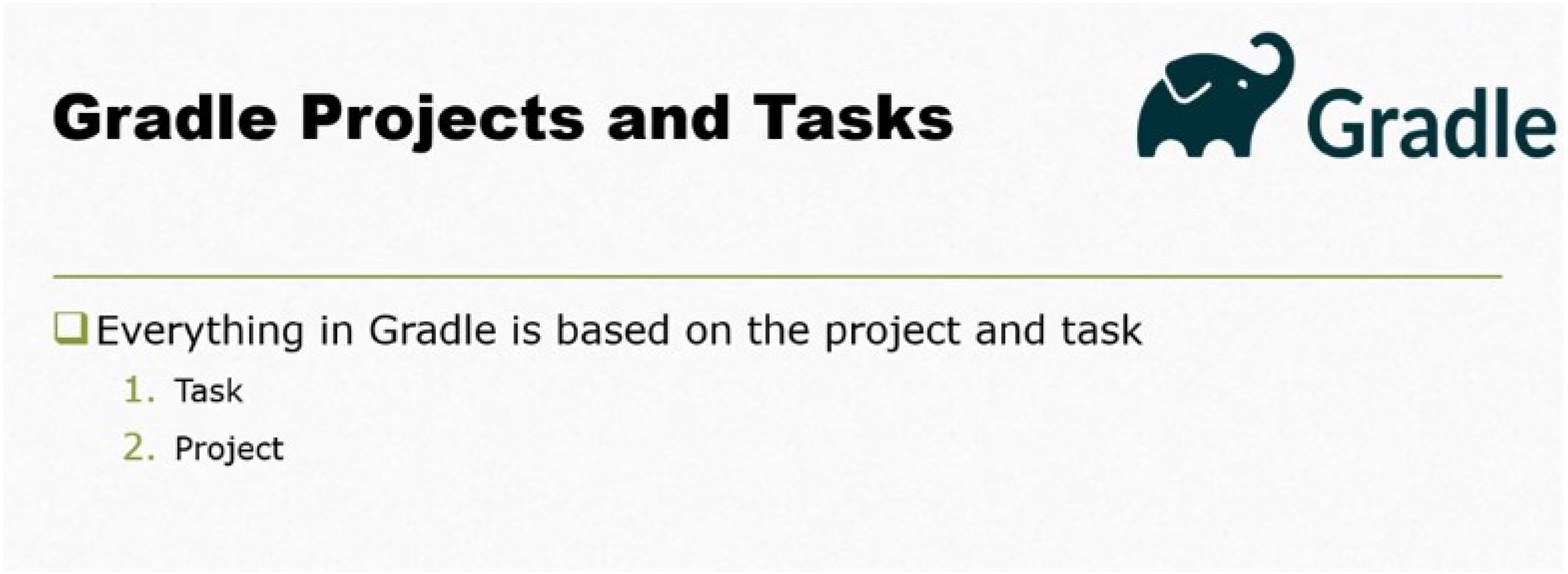
- Gradle builds a script file for handling two things:
  - **projects**
  - **tasks**.
- Gradle uses Groovy or Kotlin language for writing scripts.



Writing Build Script

- Build script file is **build.gradle**
- We have to describe tasks and projects by using a Groovy or Kotlin for script
- We can define custom tasks as well.
- Example of simple custom task:

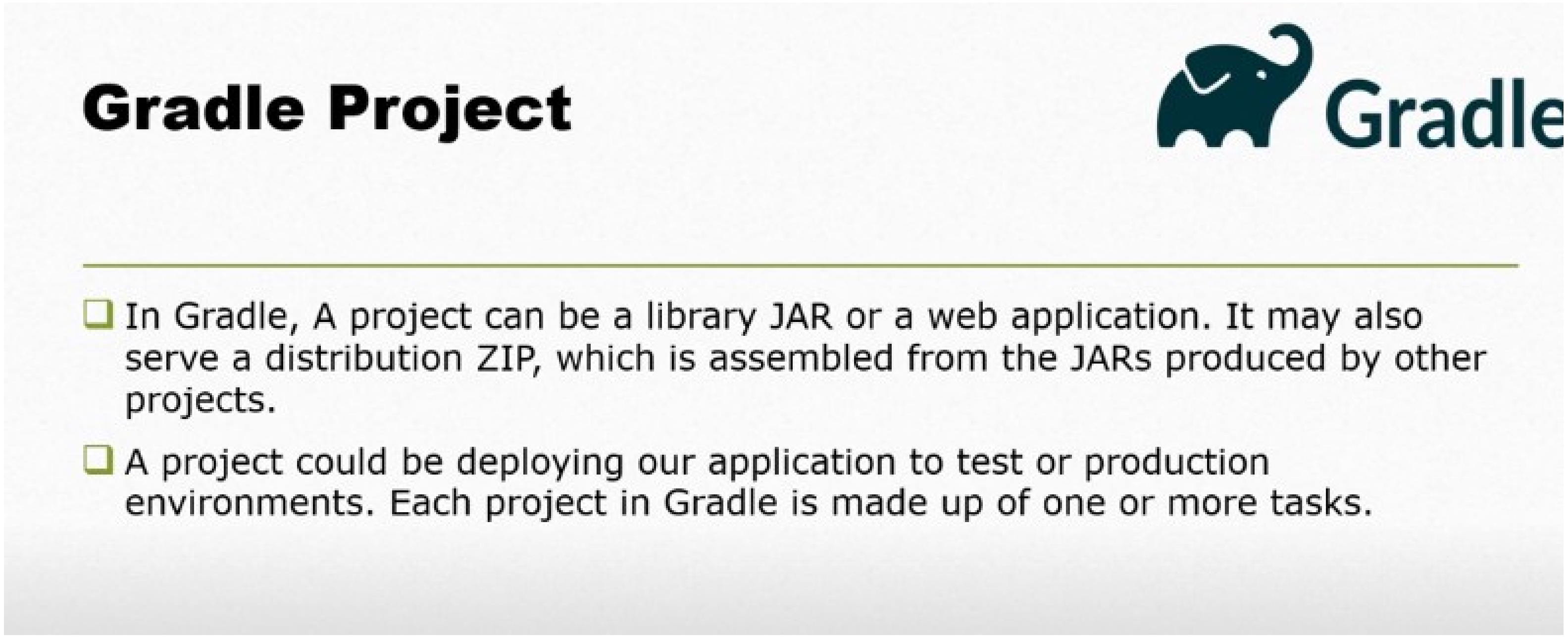
```
task myTask {
   doLast {
      println 'Welcome [redacted]
   }
}
```



Gradle Projects and Tasks

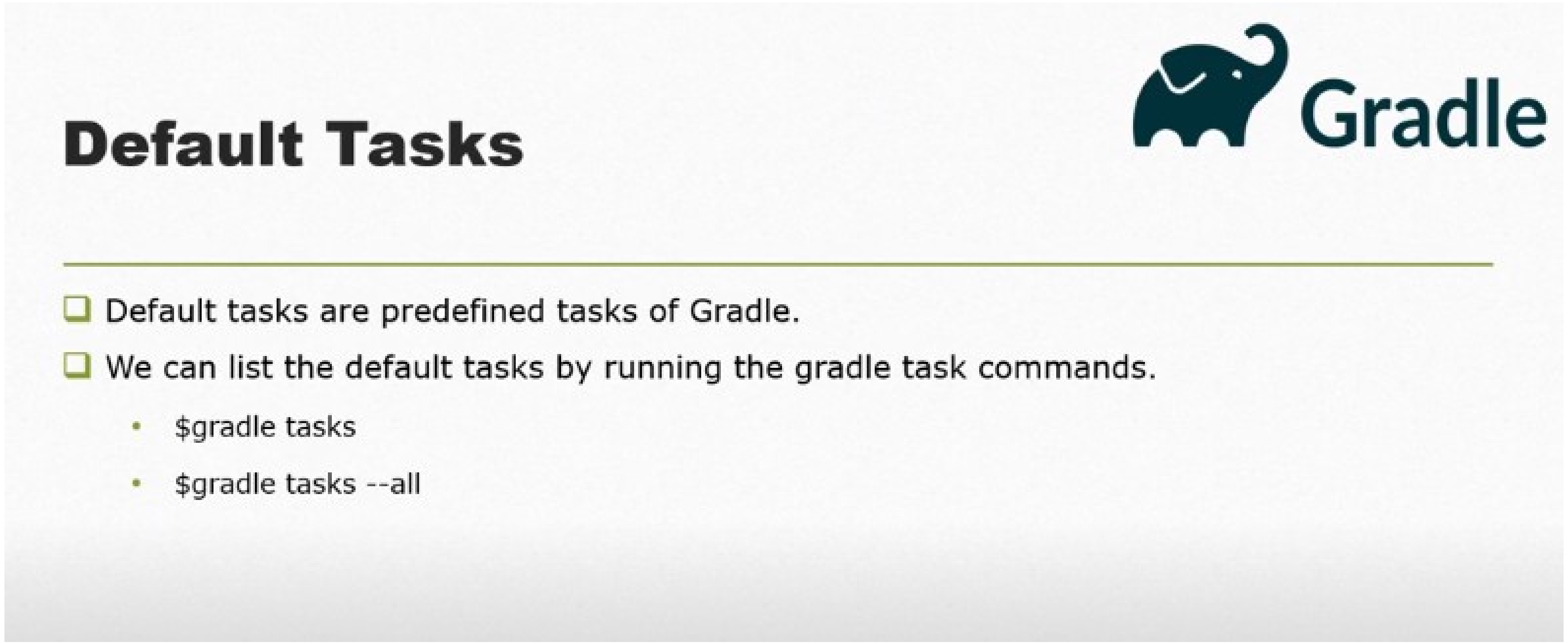
Gradle

- Everything in Gradle is based on the project and task
  1. Task
  2. Project



Gradle Project

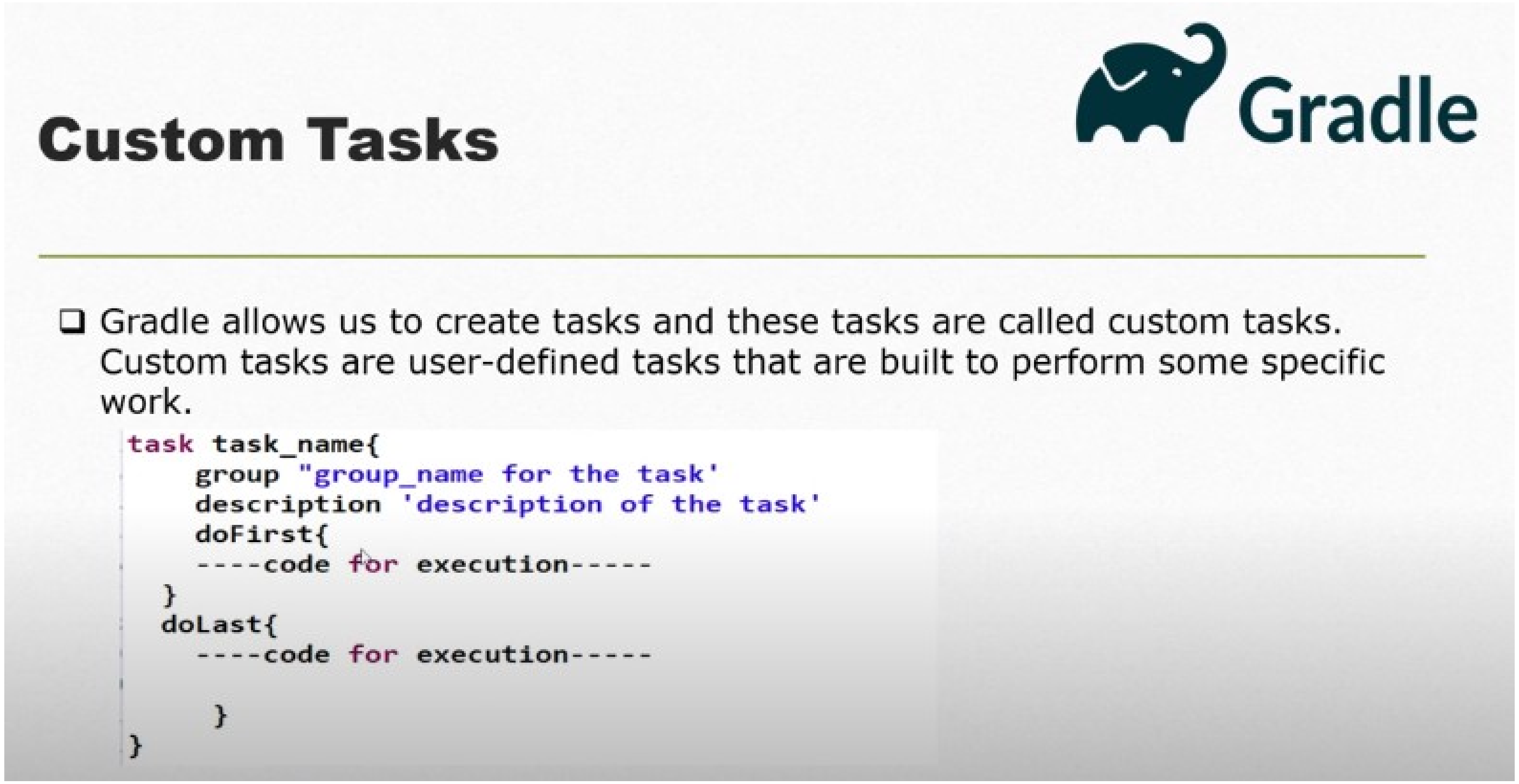
- In Gradle, A project can be a library JAR or a web application. It may also serve a distribution ZIP, which is assembled from the JARs produced by other projects.
- A project could be deploying our application to test or production environments. Each project in Gradle is made up of one or more tasks.



Default Tasks

Gradle

- ❑ Default tasks are predefined tasks of Gradle.
- ❑ We can list the default tasks by running the gradle task commands.
  - $gradle tasks
  - $gradle tasks --all



Custom Tasks

- Gradle allows us to create tasks and these tasks are called custom tasks. Custom tasks are user-defined tasks that are built to perform some specific work.

```
task task_name{
    group "group_name for the task'
    description 'description of the task'
    doFirst{
    ----code for execution-----
  }
  doLast{
    ----code for execution-----

      }
}
```

Gradle Repository

- Specify the location of modules/libraries so that the gradle build can consume them. The location for storing modules/libraries is called a **repository**.
- Repositories can be in different forms, such as a local or remote repository.
- At runtime, Gradle will discover the declared dependencies required for operating a specific task. Once a dependency is resolved, the resolution mechanism stores the essential files of dependency in the local cache memory also call dependency cache.
- Future builds reuse the files saved in the cache to skip unnecessary network calls.

configure Maven central as dependency source.

- In your build file you specify the remote repositories to look for dependencies. Gradle supports Maven and Ivy repositories to search for dependencies. The following listing shows how to configure Maven central as dependency source.

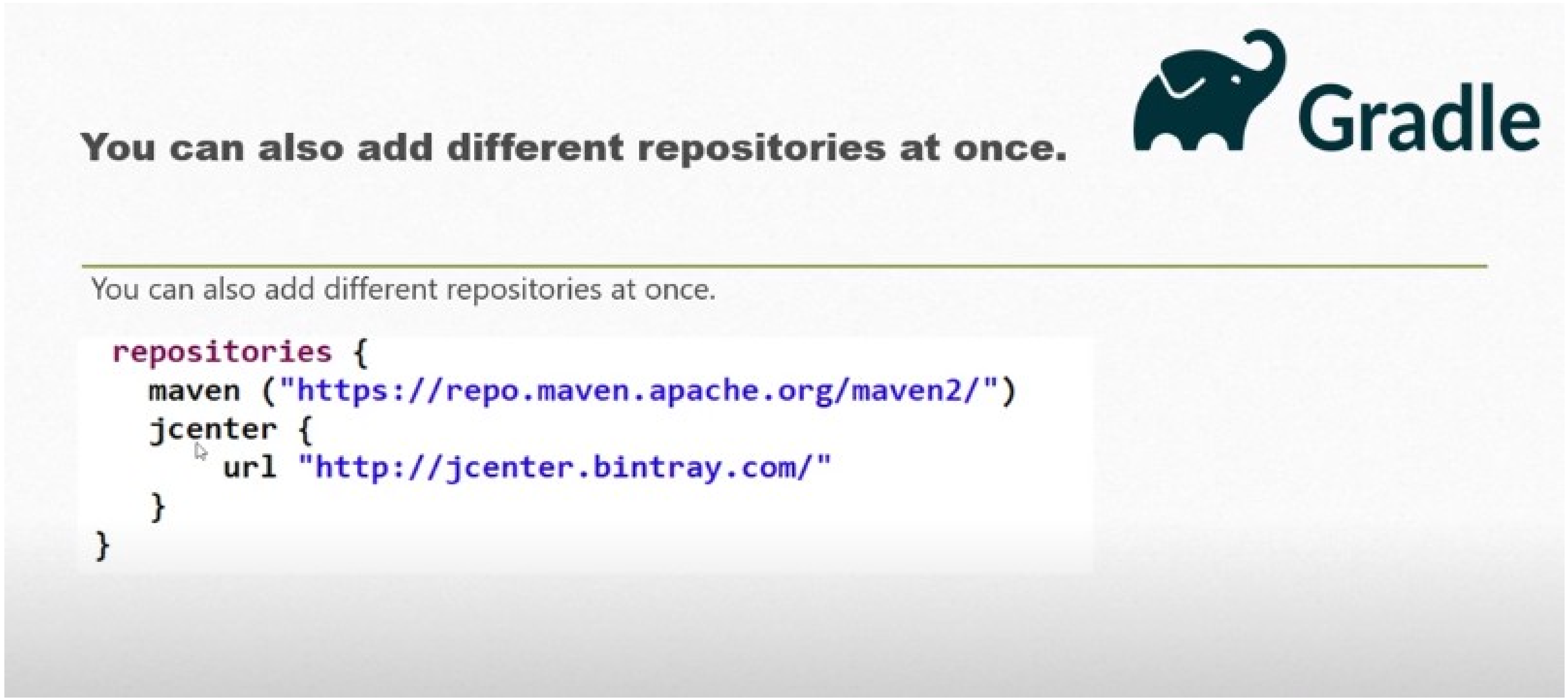
```
repositories {
    mavenCentral()
}

 OR

repositories {
    maven {
        url "https://repo.maven.apache.org/maven2/"
    }
}
```

You can also specify other targets, for example Bintray as Maven repository

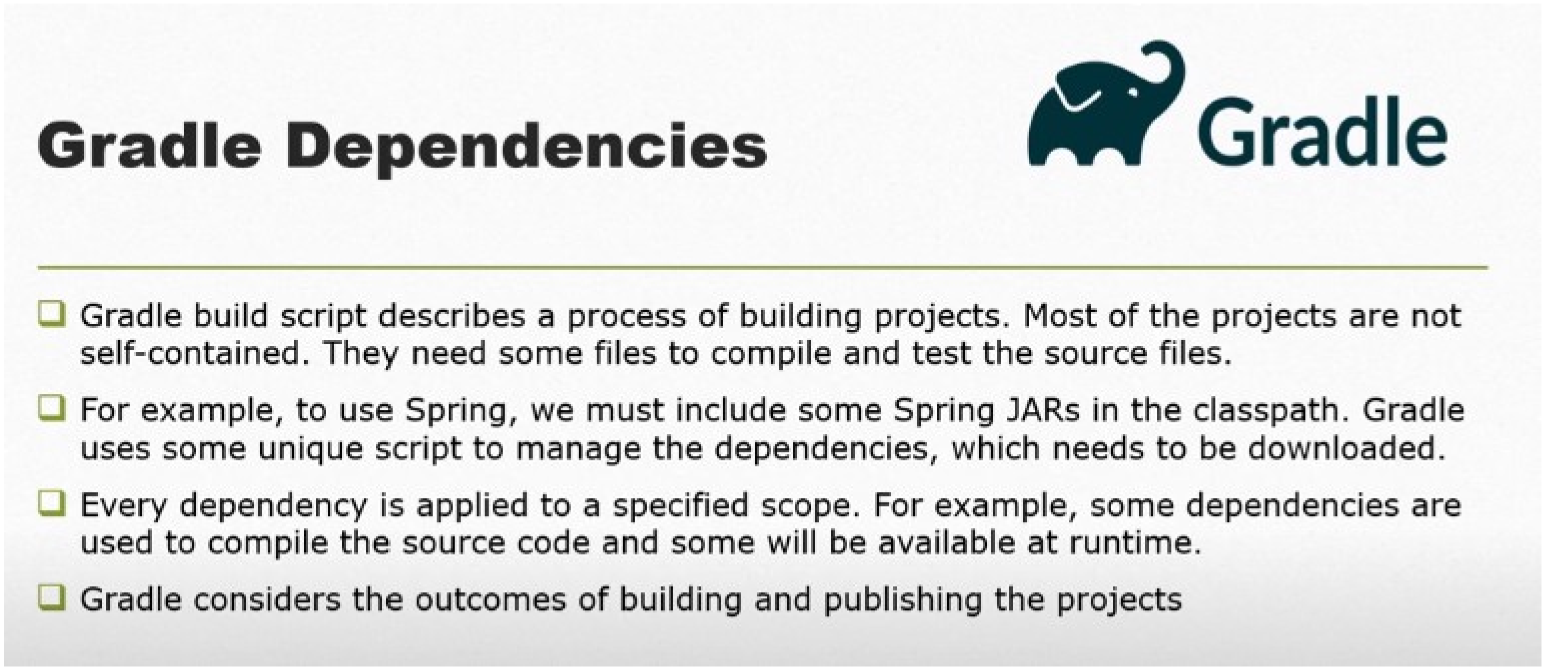
```
repositories {
   maven ("http://jcenter.bintray.com/")
}
```



You can also add different repositories at once.

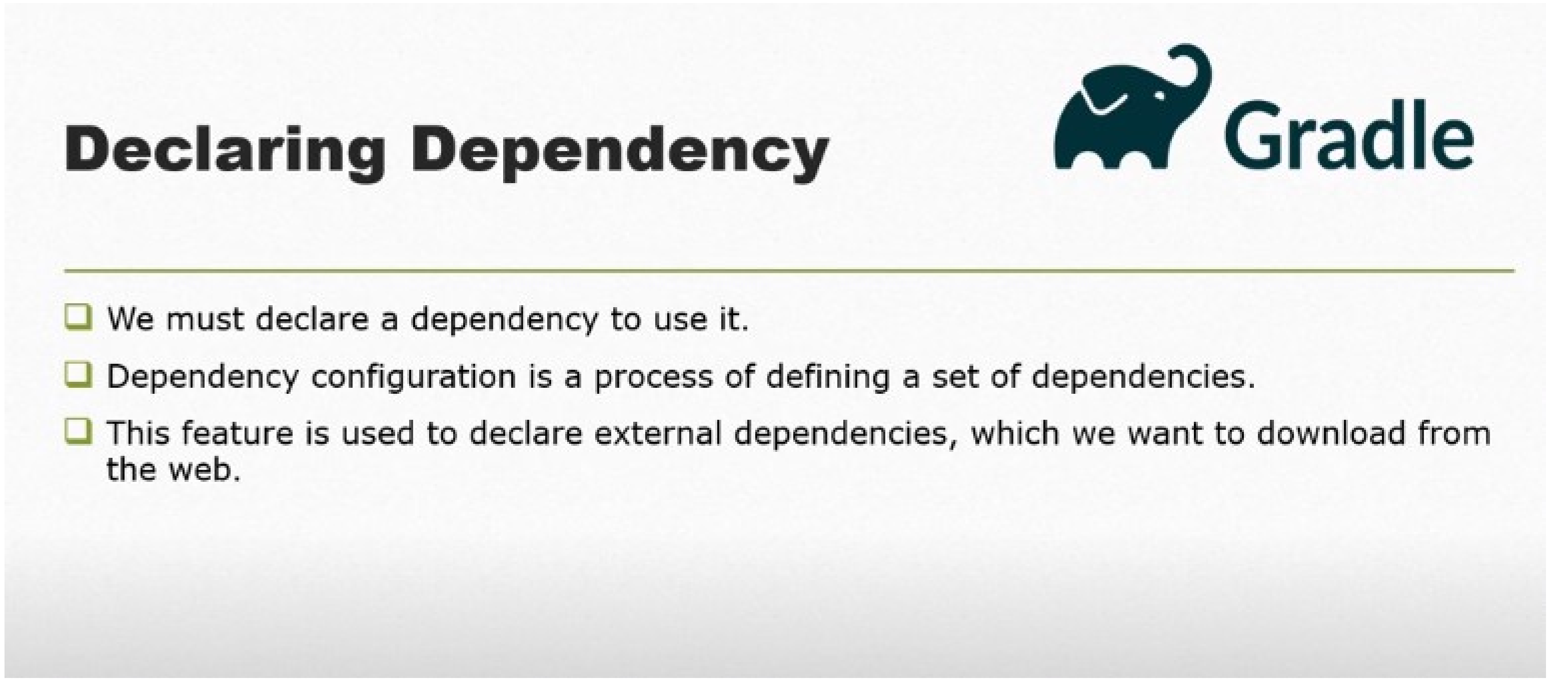
You can also add different repositories at once.

```
repositories {
  maven ("https://repo.maven.apache.org/maven2/")
  jcenter {
      url "http://jcenter.bintray.com/"
  }
}
```



Gradle Dependencies

- Gradle build script describes a process of building projects. Most of the projects are not self-contained. They need some files to compile and test the source files.
- For example, to use Spring, we must include some Spring JARs in the classpath. Gradle uses some unique script to manage the dependencies, which needs to be downloaded.
- Every dependency is applied to a specified scope. For example, some dependencies are used to compile the source code and some will be available at runtime.
- Gradle considers the outcomes of building and publishing the projects



Declaring Dependency

Gradle

- We must declare a dependency to use it.
- Dependency configuration is a process of defining a set of dependencies.
- This feature is used to declare external dependencies, which we want to download from the web.

Resolving Dependencies

- External dependencies are the one of the type of dependencies. This is a dependency on some files built outside on the current build, and stored in a repository of some kind, such as Maven central, or a corporate Maven or Ivy repository, or a directory in the local file system.
- The following code snippet is to define the external dependency. Use this code in **build.gradle** file.

```
dependencies {
    compile group: 'org.springframework', name: 'spring-core', version: '5.2.8.RELEASE'

}
```

- An external dependency is declaring in the shortcut form like "group: name: version".

Exposing Artifacts for Consumption

- Dependency configurations are also used to publish files. These published files are called artifacts. Usually, we use plug-ins to define artifacts.
- You need to tell Gradle where to publish the artifacts. You can achieve this by attaching repositories to the upload archives task.
- Use this code in **build.gradle** file.

```
apply plugin: 'maven'

uploadArchives {
   repositories {
      mavenDeployer {
         repository(url: "file://localhost/myDir/repo/")
      }
   }
}
```

Custom tasks

- Gradle allows us to create tasks; these tasks are called custom tasks. Custom tasks are user-defined tasks that are built to perform some specific work.

```
task task_name{
    group "group_name for the task'
    description 'description of the task'
    doFirst{
    ----code for execution-----
  }
  doLast{
    ----code for execution-----

      }
}
```

The doFirst and doLast block in Custom Task

- In Gradle, doFirst and doLast are two different blocks of custom task. In doFirst block, we determine the task that we want to be executed first, whereas the doLast block is used to run the task at last.

```
task myTask{
    doFirst{
        println "Task has started"
    }
    doLast{
        println "Task is completed"
    }
}
```

Copy Task in Gradle

- Gradle allows us to copy a task from one directory to another directory. Following syntax is used to copy the task:

```
task copyReportsDirForArchiving(type: Copy) {
    from "$buildDir/reports"
    into "$buildDir/toArchive"
}
```

Task dependency in Gradle

- Task dependency is used when a task depends on another task. The **dependsOn** keyword is used with the task name to make a task dependent.
- Consider the below example, it contains two tasks called taskX and taskY. The second task has 'dependsOn' keyword with first task, it means the second task only executes when the first task is successfully executed:

```
task taskX {
       println 'i am a taskX'
}
task taskY(dependsOn: taskX) {
       println "i am taskY depends on taskX"
}
```