Introduction to Gradle Build Tool

The fundamentals of building projects with Gradle

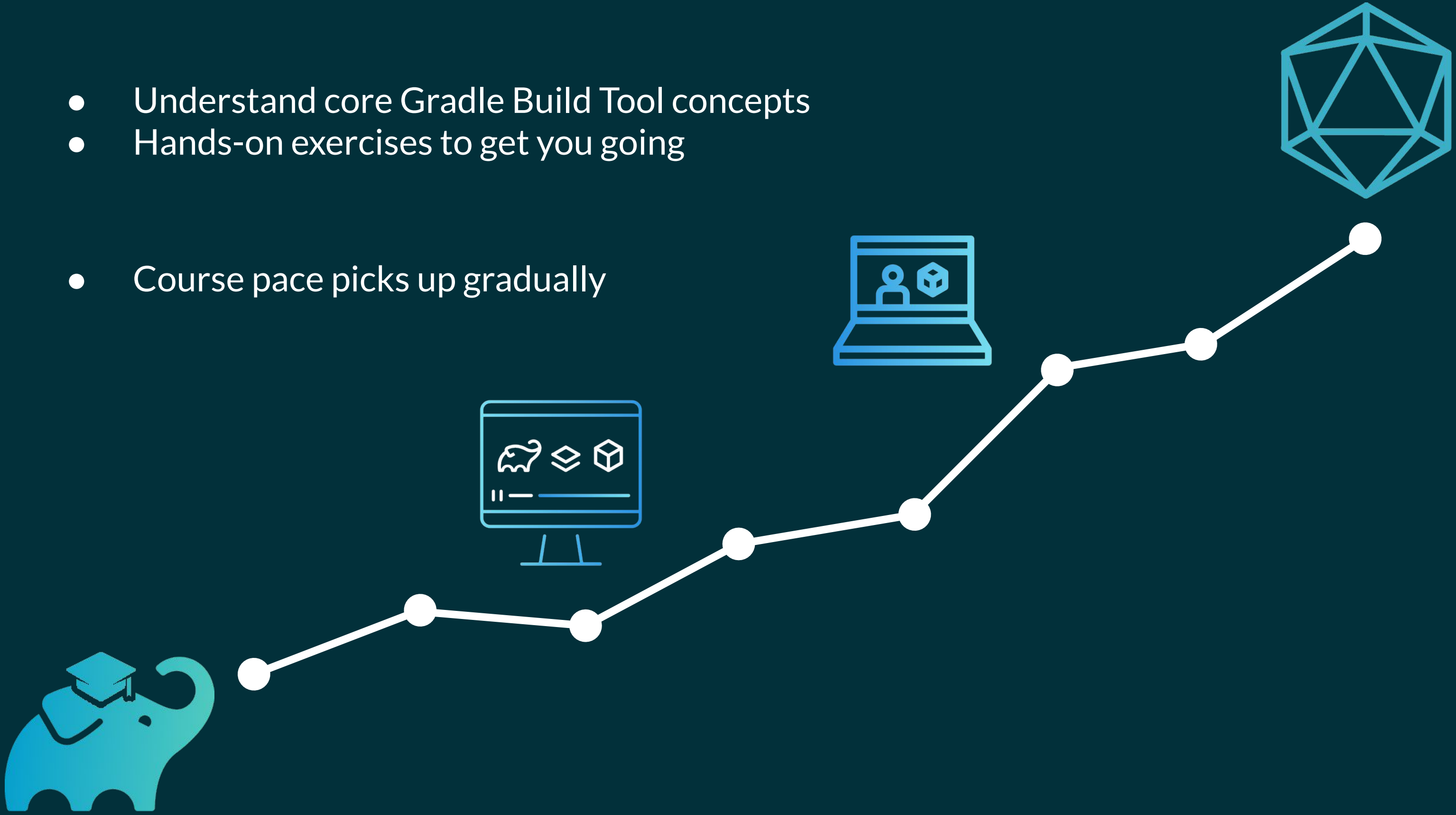
Gradle Training Program

training@gradle.com

Objectives

- Understand core Gradle Build Tool concepts
- Hands-on exercises to get you going

- Course pace picks up gradually



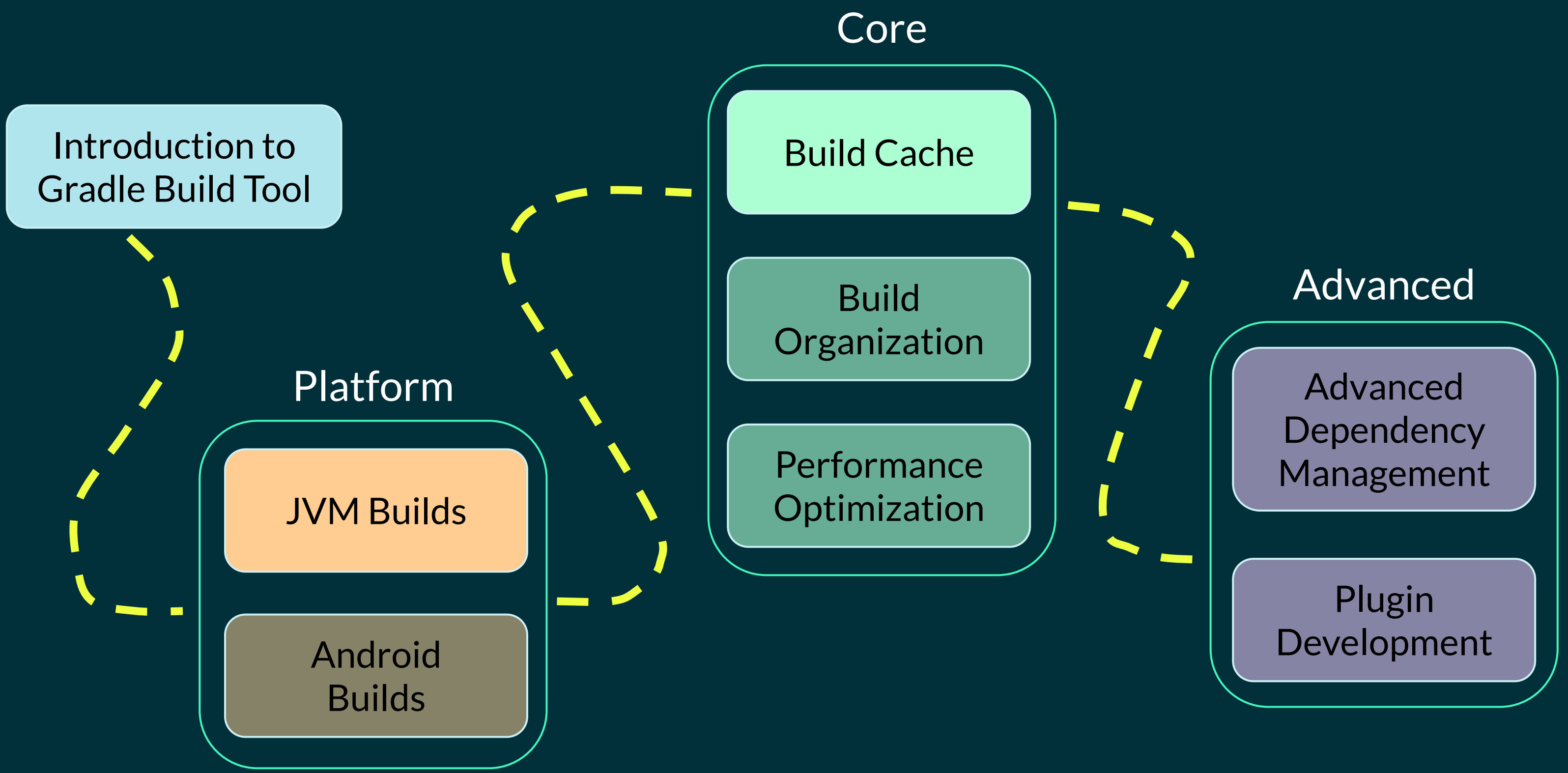
Prerequisites and Notes

- JDK 1.8+ and Gradle Build Tool installed
  - https://gradle.org/install/
- Recommended - IntelliJ
  - IntelliJ community edition used in examples
  - https://www.jetbrains.com/idea/download/
- Basic knowledge of software development
  - Java & Kotlin experience NOT needed
- Hands-on labs
  - READMEs will have instructions

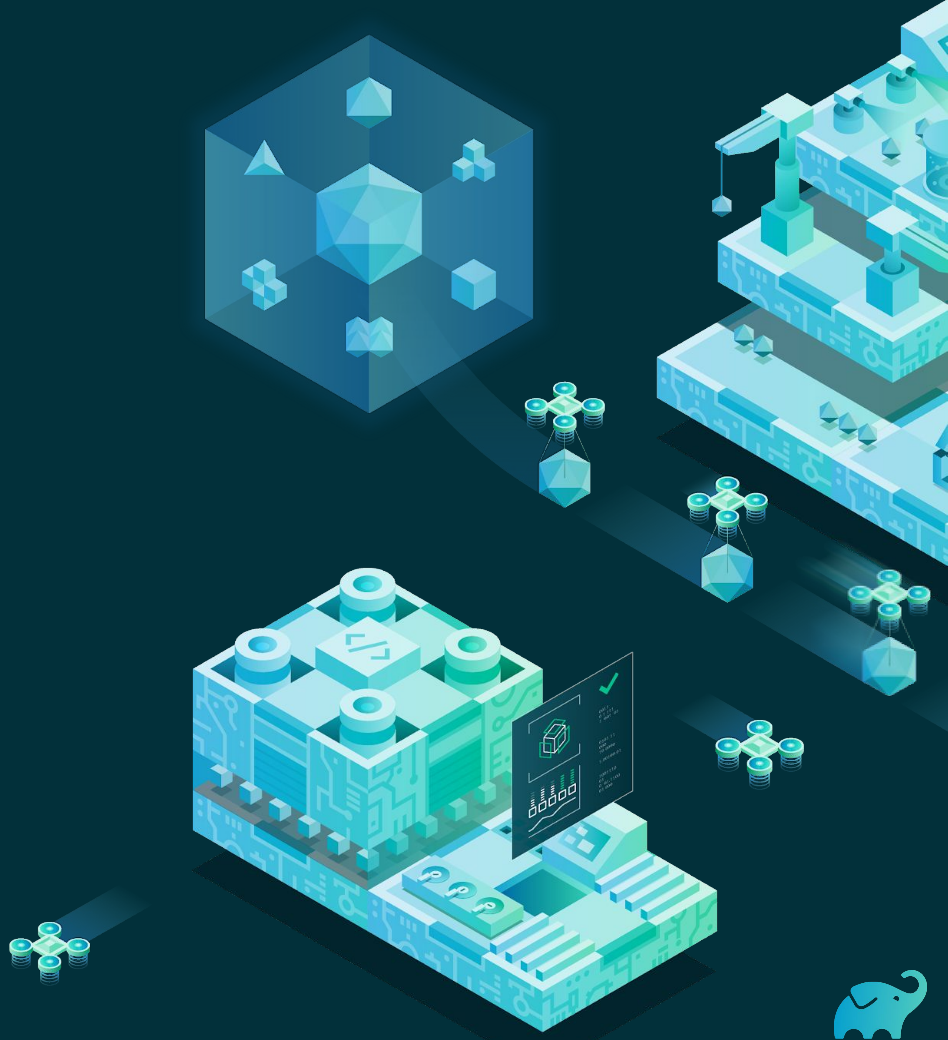
Agenda

- About Gradle Build Tool
- Core concepts
    - Build Configuration
    - Build Lifecycle
    - Plugins
    - Tasks
    - Dependency Management
- Publishing
- Multi-Project Builds

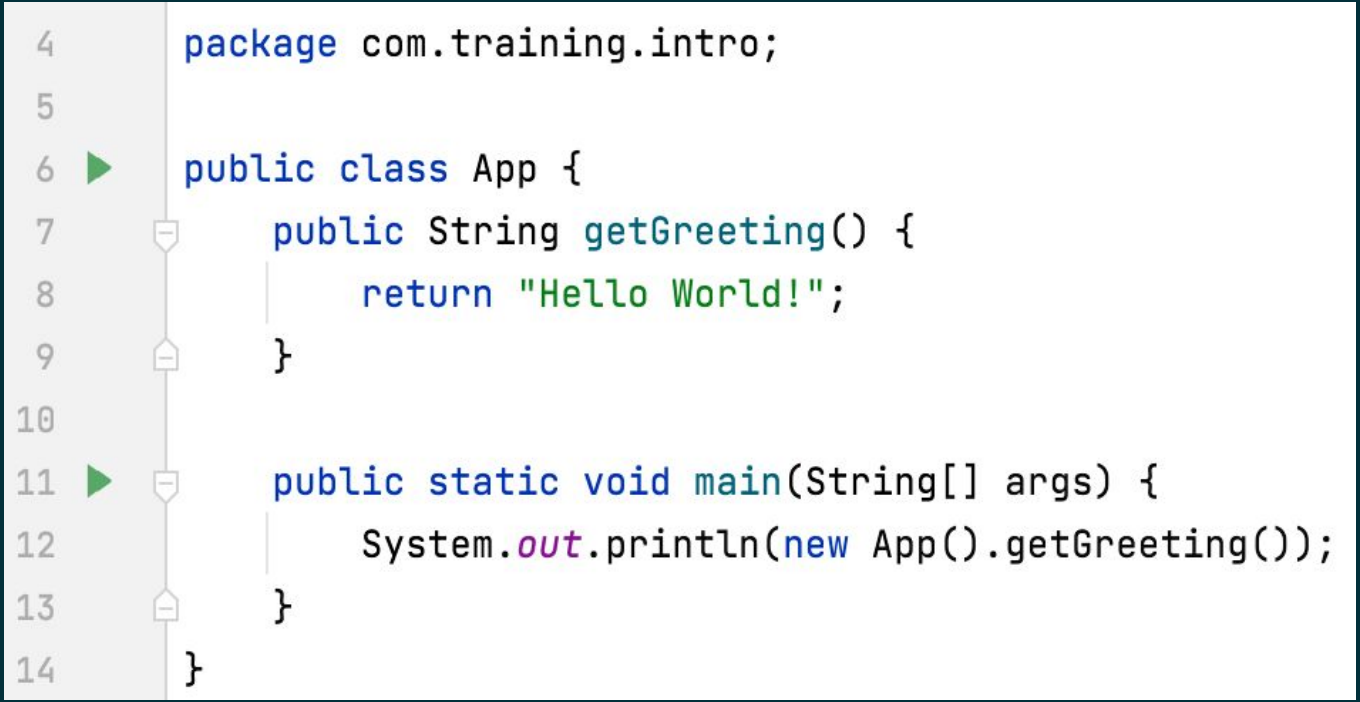
Training Journey - Future Topics



What is Gradle Build Tool?



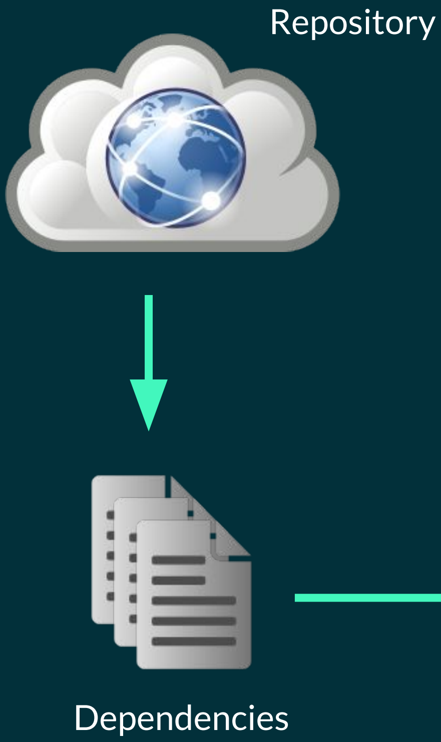
Software Development - Write Code & Tests

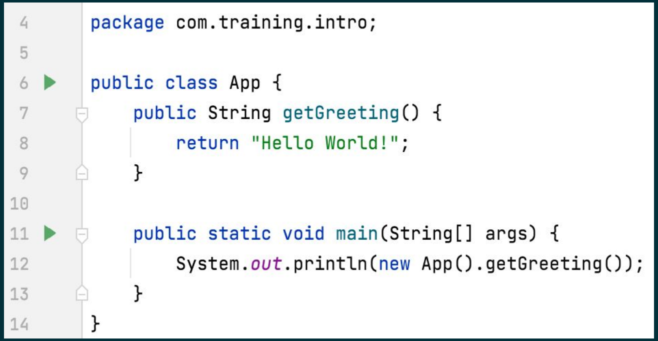


```java
package com.training.intro;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        System.out.println(new App().getGreeting());
    }
}
```

Your Code

Software Development - Dependencies

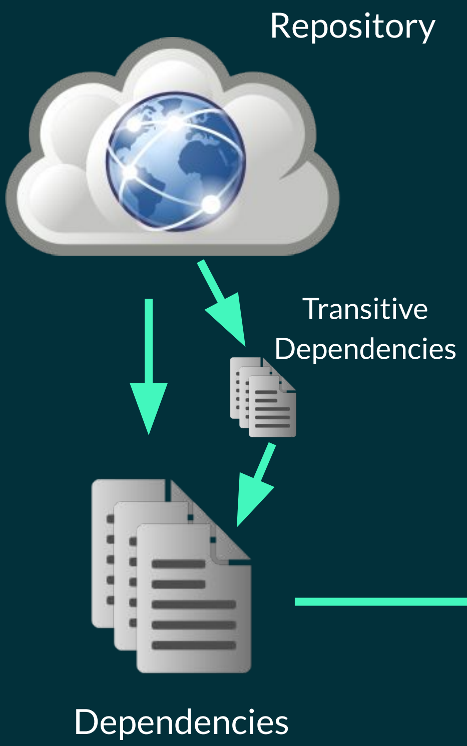


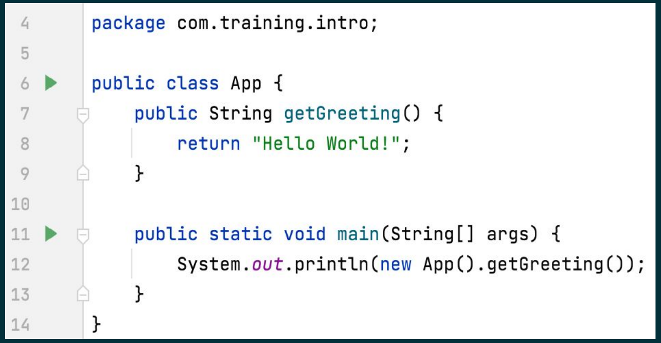


```java
package com.training.intro;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        System.out.println(new App().getGreeting());
    }
}
```

Your Code

Software Development - Dependency Management

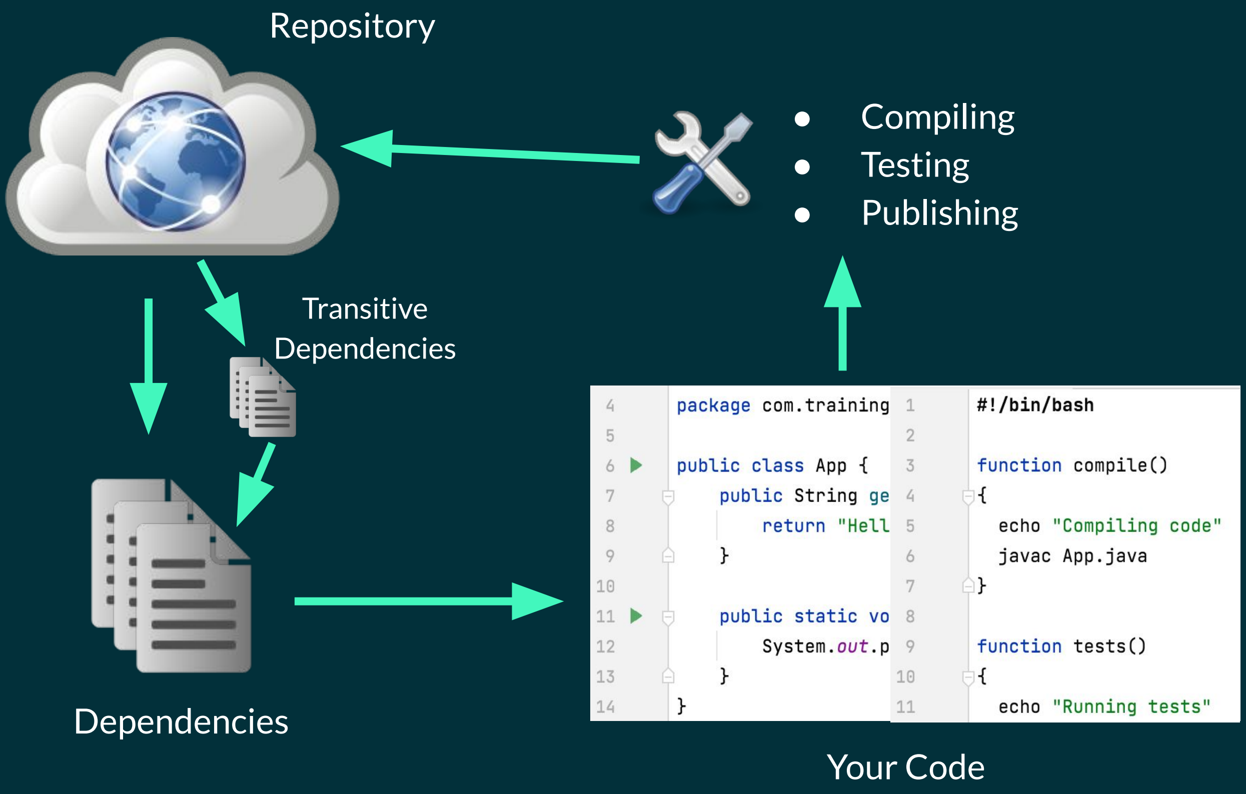




```java
package com.training.intro;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        System.out.println(new App().getGreeting());
    }
}
```

Your Code

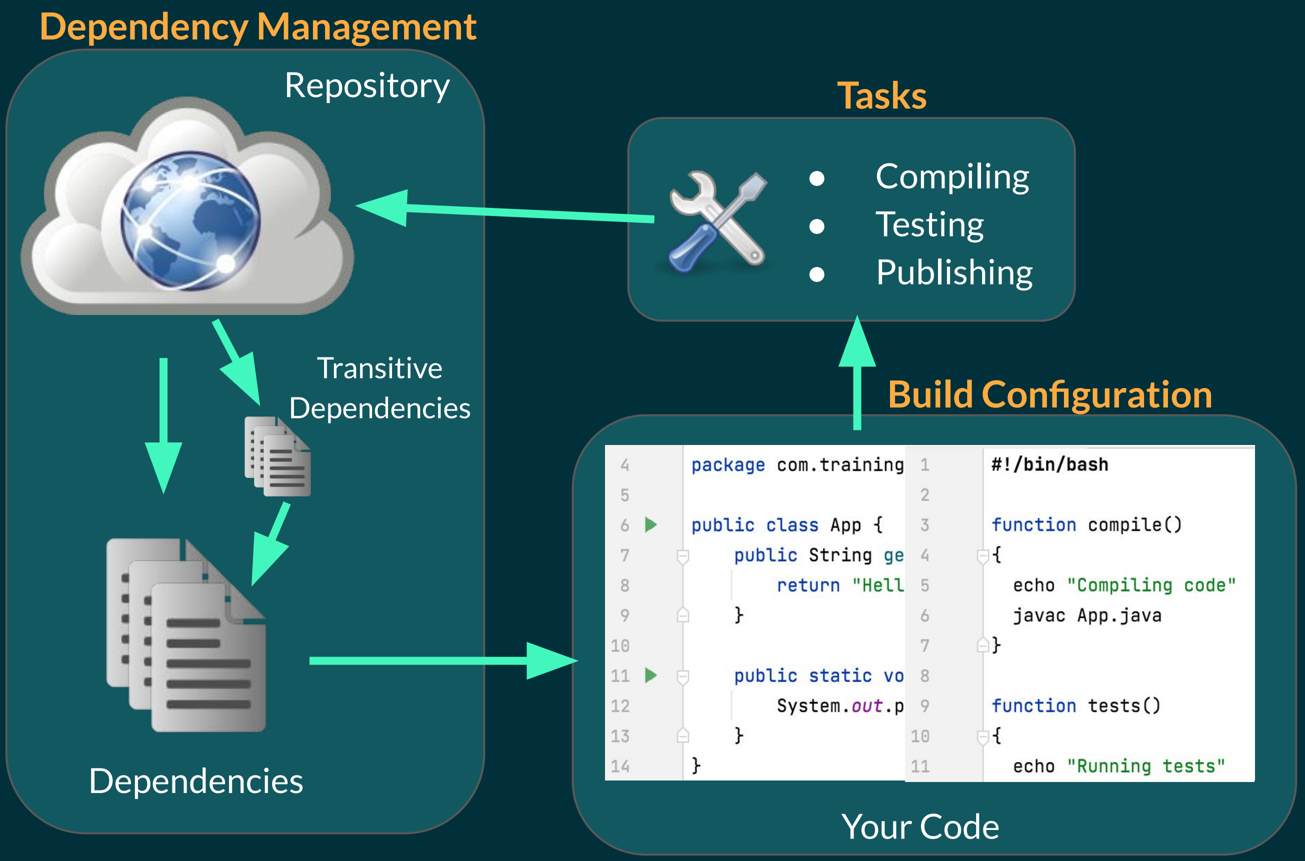
Software Development - Tasks & Build Configuration



```java
package com.training

public class App {
    public String ge
        return "Hell
    }

    public static vo
        System.out.p
    }
}
```

```bash
#!/bin/bash

function compile()
{
  echo "Compiling code"
  javac App.java
}

function tests()
{
  echo "Running tests"
```

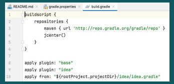
Software Development - Build Management Concepts



```
package com.training

public class App {
    public String ge
        return "Hell
    }

    public static vo
        System.out.p
    }
}
```

```
#!/bin/bash

function compile()
{
  echo "Compiling code"
  javac App.java
}

function tests()
{
  echo "Running tests"
```

Gradle Build Tool Core Concepts



```
buildscript {
    repositories {
        maven { url 'http://repo.gradle.org/gradle/repo' }
        jcenter()
    }
}

apply plugin: "base"
apply plugin: "idea"
apply from: "${rootProject.projectDir}/idea/idea.gradle"
```

Build Configuration

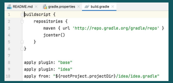
Tasks

Dependency Management

Build Lifecycle

- Gradle is an open-source build automation tool
- Automate tasks:
  - Compiling
  - Testing
  - Publishing
- Comprehensive and flexible dependency management
  - Consistent and reproducible builds

Gradle Build Tool Core Concepts



```
buildscript {
    repositories {
        maven { url 'http://repo.gradle.org/gradle/repo' }
        jcenter()
    }
}

apply plugin: "base"
apply plugin: "idea"
apply from: "${rootProject.projectDir}/idea/idea.gradle"
```

Build Configuration

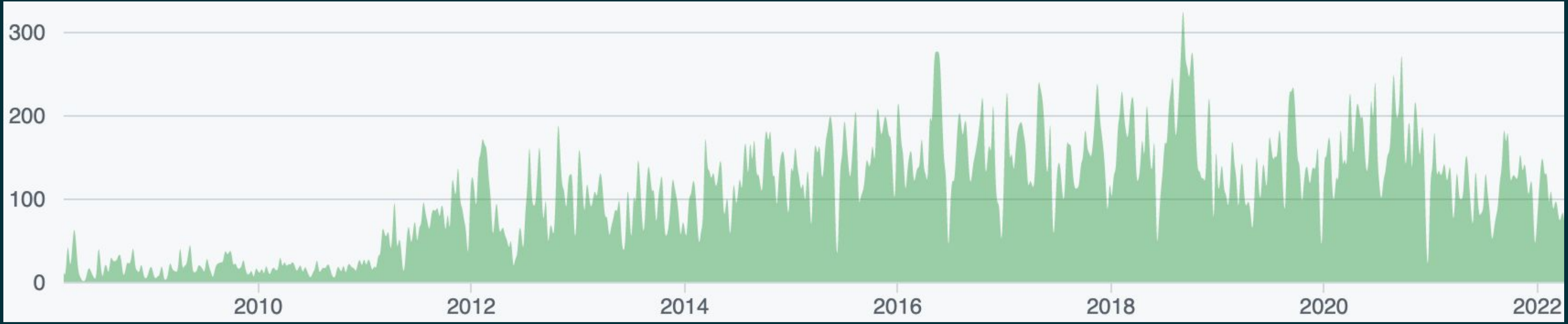
Tasks & Plugins

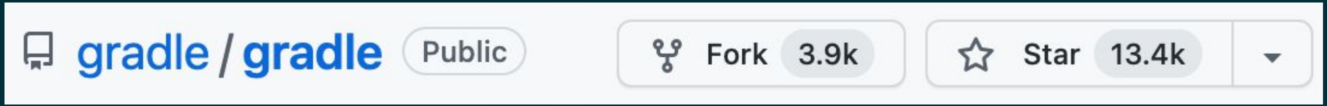
Dependency Management

Build Lifecycle

- Gradle is an open-source customizable build automation tool
- Open-source
- Automate tasks:
  - Compiling
  - Testing
  - Publishing
- Comprehensive and flexible dependency management
  - Consistent and reproducible builds
- Plugin framework

Gradle Community

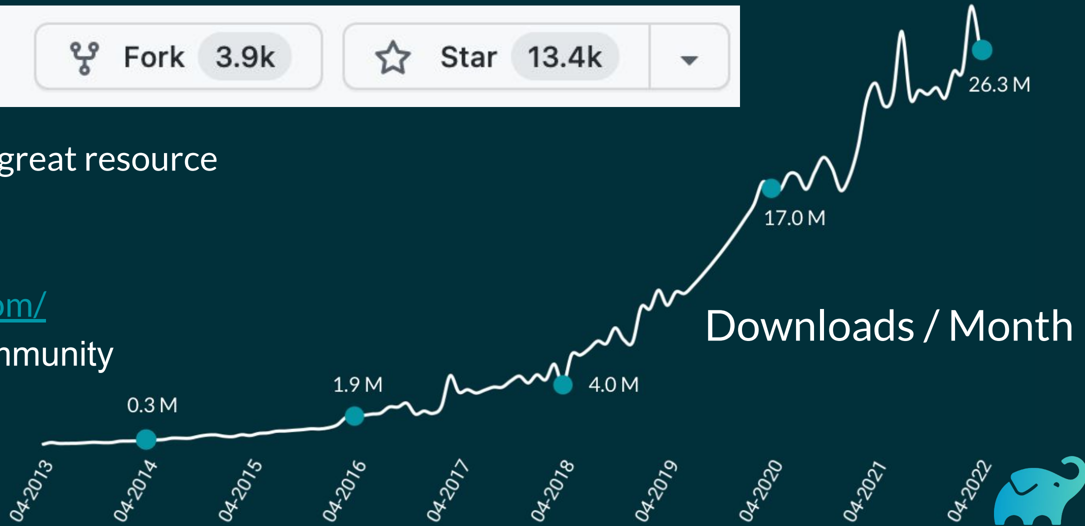



gradle / gradle Public

Fork 3.9k

Star 13.4k

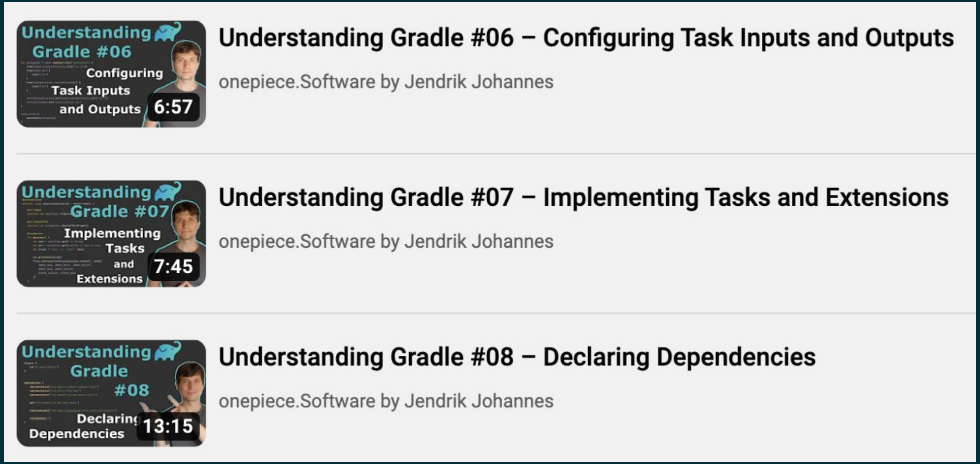
- Official docs and forums always a great resource
  - https://docs.gradle.org/
  - https://discuss.gradle.org/
  - https://newsletter.gradle.com/
- Useful plugins contributed by community
  - https://plugins.gradle.org/



Downloads / Month

Community Video Content

- Grateful to have excellent video content provided by community members
    - Includes tutorials videos
- Recommend onepiece.Software by Jendrik Johannes


Understanding Gradle #06 Configuring Task Inputs and Outputs
6:57
Understanding Gradle #07 Implementing Tasks and Extensions
7:45
Understanding Gradle #08 Declaring Dependencies
13:15

Understanding Gradle #06 – Configuring Task Inputs and Outputs
onepiece.Software by Jendrik Johannes

Understanding Gradle #07 – Implementing Tasks and Extensions
onepiece.Software by Jendrik Johannes

Understanding Gradle #08 – Declaring Dependencies
onepiece.Software by Jendrik Johannes

Checkin Question

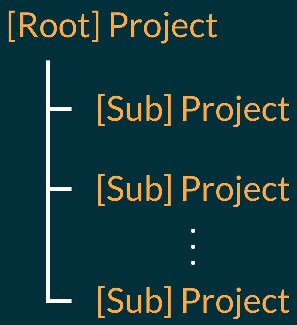
- Dependencies can have their own dependencies. What are these additional dependencies called?
    a. Transition
    b. Transient
    c. Transitive
    d. Recursive

Build Configuration

- About Gradle Build Tool ✓
- Build Configuration
- Build Lifecycle
- Plugins
- Tasks
- Dependency Management
- Publishing
- Multi-Project Builds

Defining Build Configuration

- Configuration consists of Settings & Projects
  - Settings → settings file
    - Root project name
    - List of subprojects
  - Subprojects → build file
    - Plugins
    - Dependencies
    - Tasks
    - Source code
- Can write configuration in either Kotlin or Groovy
  - Configurations are very similar
  - In examples will use Kotlin
    - Kotlin has better IDE support (at least with IntelliJ)
    - Note: Currently Kotlin has some performance issues for large projects



Groovy Example

app/build.gradle

```
plugins {
    id 'application'
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'

    implementation 'com.google.guava:guava:30.1.1-jre'
}
```

settings.gradle

```
rootProject.name = 'demo'
include('app')
```

Kotlin Example

app/build.gradle.kts

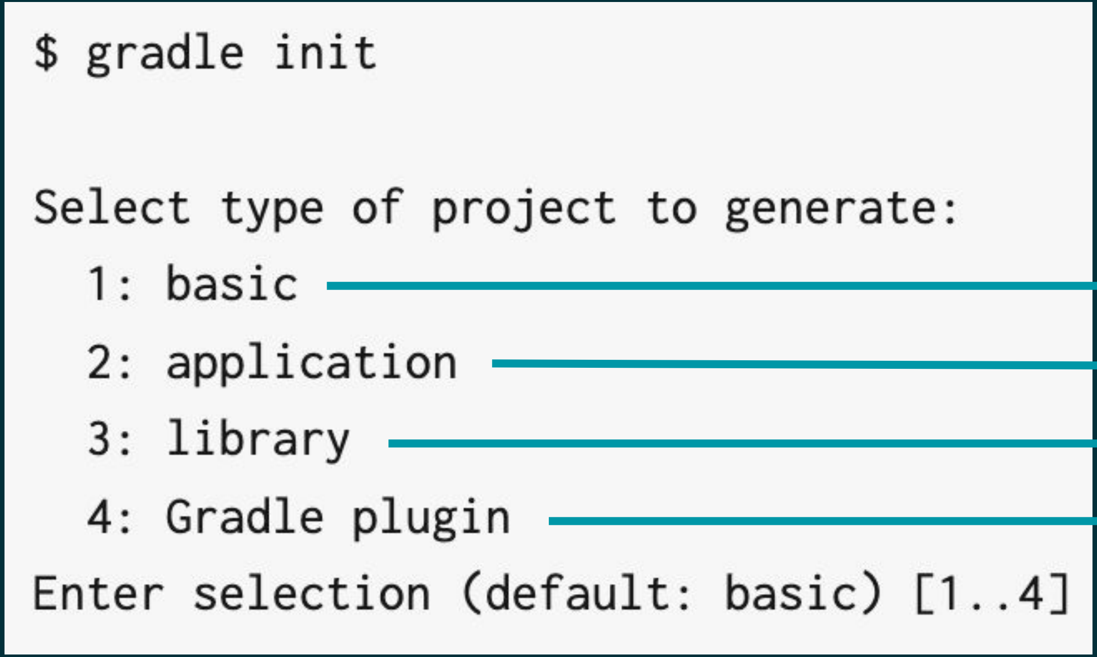
```
plugins {
    application
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter:5.8.1")

    implementation("com.google.guava:guava:30.1.1-jre")
}
```

settings.gradle.kts

```
rootProject.name = "demo"
include("app")
```

Getting Started - Gradle Init

- You can create a new Gradle build project by running:



```
$ gradle init

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4]
```

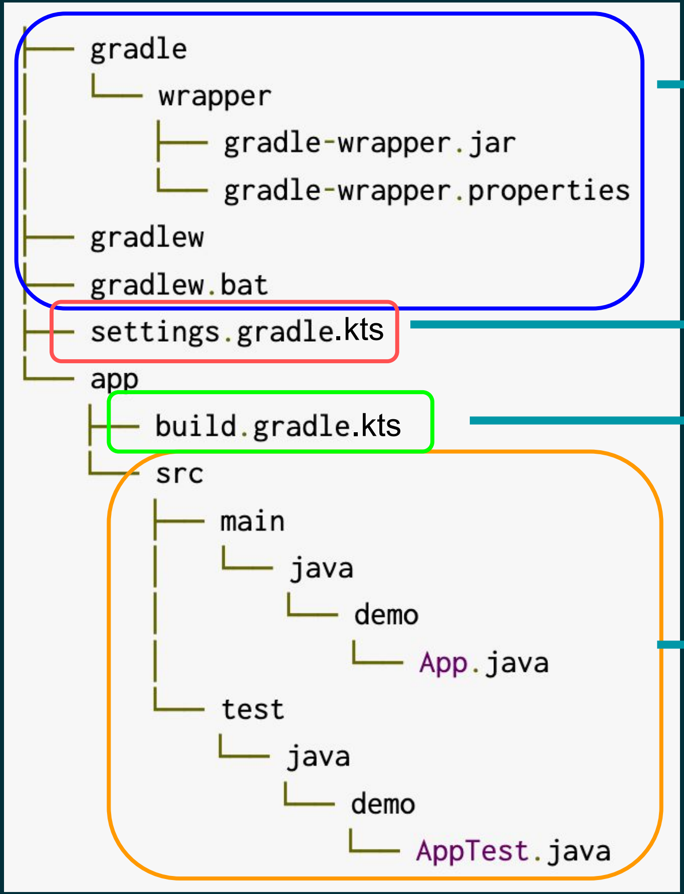
Just provides basic layout

Executable project

Library to be shared

Plugin for Gradle

Inspecting Directory Layout



```
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle.kts
└── app
    ├── build.gradle.kts
    └── src
        ├── main
        │   └── java
        │       └── demo
        │           └── App.java
        └── test
            └── java
                └── demo
                    └── AppTest.java
```

Wrapper - more on this soon

Settings file - list of Subprojects

Build file - Dependencies and Tasks

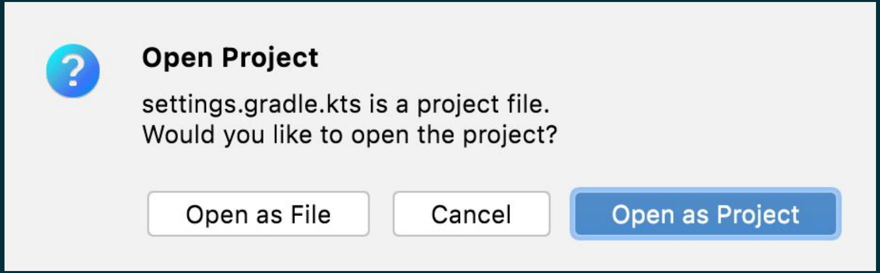
Source layout - notice structure:

- src/main/java
- src/test/java

(more on this later)

Opening with IntelliJ

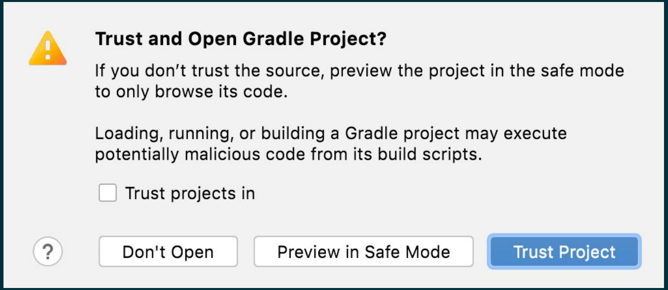
- Open the settings.gradle.kts file



Open Project

settings.gradle.kts is a project file.
Would you like to open the project?

- If asked whether to trust the file, say yes

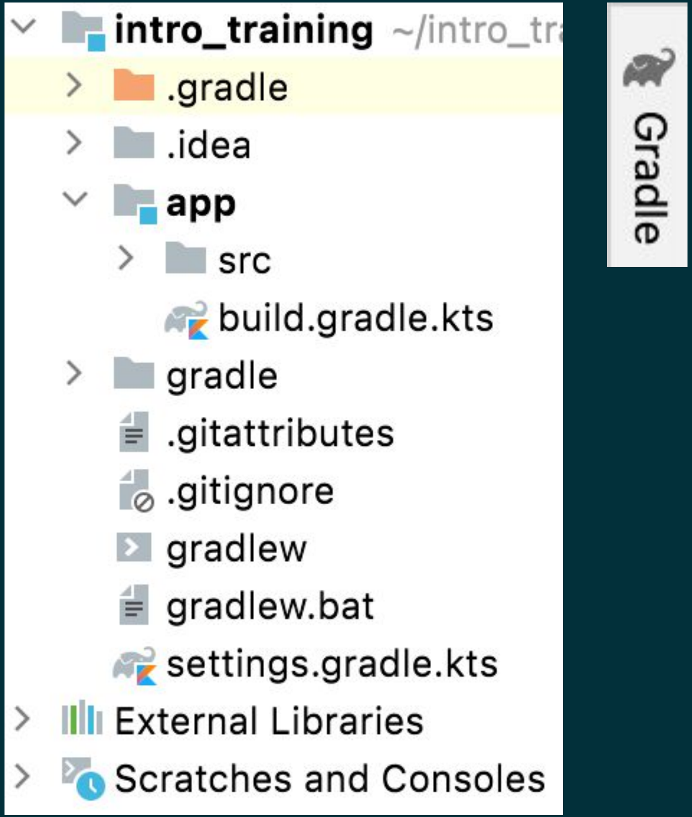


Trust and Open Gradle Project?

If you don't trust the source, preview the project in the safe mode to only browse its code.

Loading, running, or building a Gradle project may execute potentially malicious code from its build scripts.

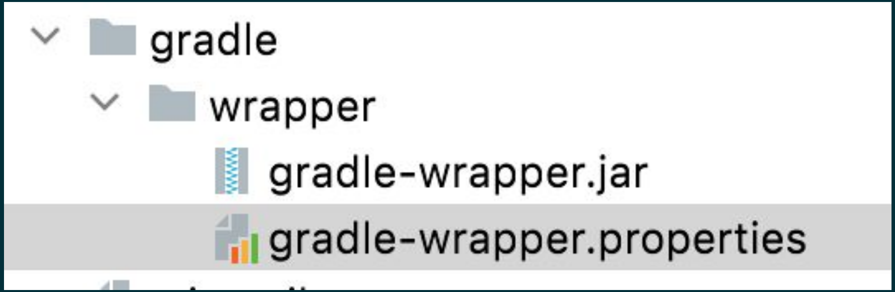
☐ Trust projects in

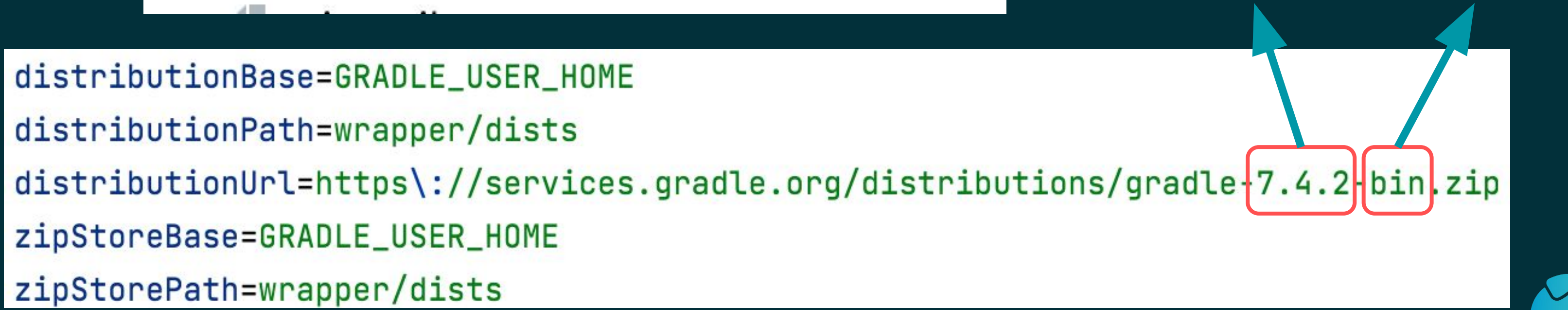


```
intro_training ~/intro_tr
  .gradle
  .idea
  app
    src
    build.gradle.kts
  gradle
  .gitattributes
  .gitignore
  gradlew
  gradlew.bat
  settings.gradle.kts
External Libraries
Scratches and Consoles
```

Gradle Wrapper

- Goal: Consistent and reproducible builds
- Challenge: All team members and build machines use same version of Gradle
- Solution: Wrapper that downloads a specific version of Gradle
- Version specified in gradle/gradle-wrapper.properties

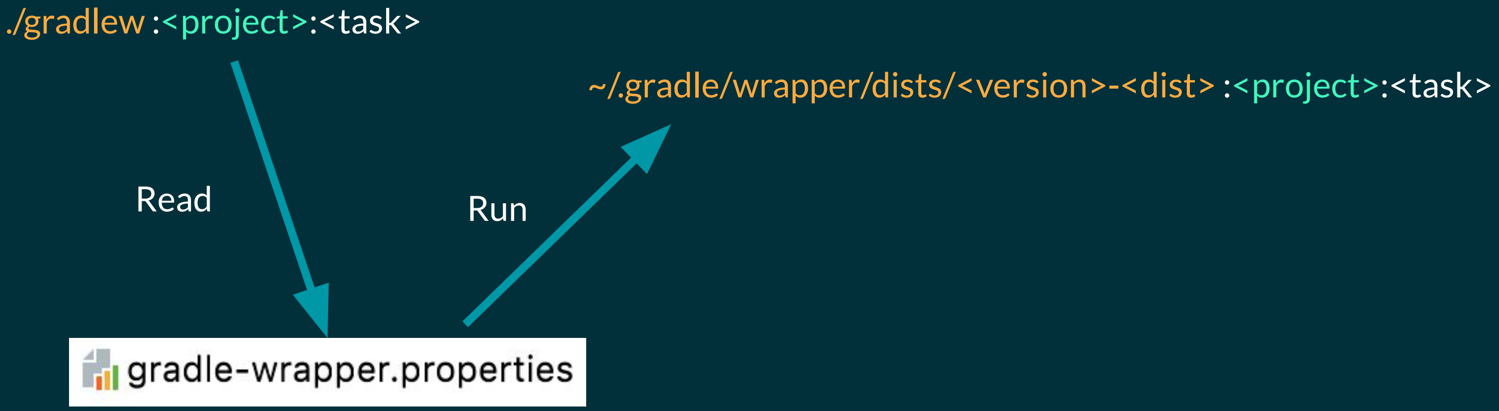




```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.4.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Gradle Wrapper

- Gradle versions downloaded to ~/.gradle/wrapper/dists
- Distribution bin = binary only
  - Smaller in size, good for build machines
- Distribution all = binary + sources
  - Helpful in IDE (usually for Groovy)


./gradlew :<project>:<task>
Read
Run
~/.gradle/wrapper/dists/<version>-<dist> :<project>:<task>
gradle-wrapper.properties

Gradle Wrapper Task

- wrapper task to generate in existing project
- Can specify Gradle version and distribution
- Checked into version control along with Gradle configuration

```
gradle wrapper --gradle-version 7.4.2 --distribution-type all
```

Checkin Question

- What do Kotlin configuration files end with?

  a. .kotlin

  b. .gradle

  c. .ktln

  d. .gradle.kts

Checkin Question

- What command is used to create configuration for a new Gradle project?
    a. gradle go
    b. gradle start
    c. gradle init
    d. gradle initialize

Checkin Question

- What is the main goal of using the Gradle wrapper?
  a. Provides additional task functionality
  b. Helps ensure consistent builds produced by everyone
  c. Creates packaging for your software
  d. Provides command-line auto-complete features

Checkin Question

- What are software pieces managed in a Gradle project called?
    a. Projects
    b. Root project
    c. Component
    d. Subprojects

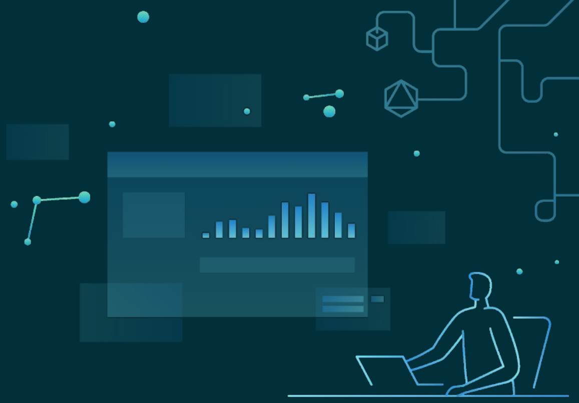
Hands-on Exercise 1

- Introduction to Gradle Build Tool Exercises

- Initializing new Gradle project
- Opening a Gradle project in IntelliJ IDE
- Explore Gradle files

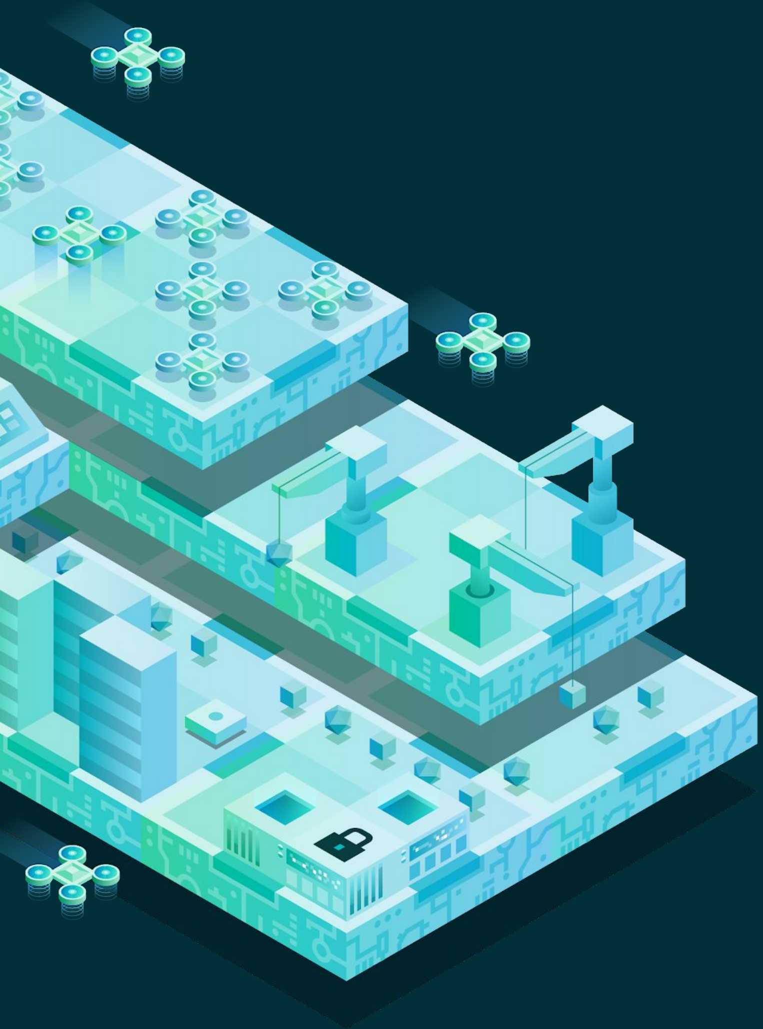
Build Lifecycle

- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle
- Plugins
- Tasks
- Dependency Management
- Publishing
- Multi-Project Builds



Build Lifecycle - Phases

- Steps taken by Gradle when performing tasks
  - Initialization
    - Determine projects -> evaluate settings.gradle.kts
  - Configuration
    - Evaluate all build scripts -> evaluate each build.gradle.kts
    - Build object model
      - Projects and tasks
  - Execution
    - Execute tasks
- Understanding these steps Gradle takes will help with other concepts



Plugins

- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins
- Tasks
- Dependency Management
- Publishing
- Multi-Project Builds

Plugins

- Plugins can be applied to Gradle configurations
- Extend Gradle capabilities
  - Add new configuration model
  - Initialize configuration
  - Add tasks
- Reusable common functionality

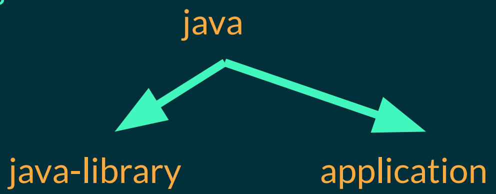
Plugins - Types

- Built-in
  - Shipped with Gradle distribution
  - Plugin reference
- Community
  - Download from plugin repository
  - Need to specify version [if downloading]
  - Plugin portal
- Local
  - Implemented locally

```
plugins {
    id("application") // Built-in
    id("org.other.plugin") version "1.4.0" // Community
}
```

Example - Java Plugins

- java plugin
  - Build configuration for source code locations: SourceSets
    - src/main/java
    - src/test/java
  - Tasks like compileJava and test
- java-library plugin
  - Also applies java plugin
  - Adds “api” dependency configuration - more on this later
- application plugin
  - Also applies java plugin
  - Tasks to run or package an executable application
  - Build configuration for main class
- Note that people only apply either the java-library or application plugin
  - (The java base plugin gets applied automatically)



Java SourceSets

- Configuration added by java plugin to allow logical grouping of source files
  - Similar configuration is likely to be added by plugins for other languages
- Provides way to define source code location
- Default location:
  - src/main/java
  - src/test/java
  - Gradle docs
- Can specify different source locations

```
sourceSets {
    main {
        java {
            srcDir("src/java")
        }
    }
}
```

Checkin Question

- What are the 3 Gradle build lifecycle phases?
    a. Reading, processing & execution
    b. Downloading, processing & execution
    c. Configuration, processing & execution
    d. Initialization, configuration & execution

Checkin Question

- During the initialization phase, which Gradle configuration file is evaluated?

  a. build.gradle.kts

  b. gradle-wrapper.properties

  c. settings.gradle.kts

  d. No file is evaluated during this phase

Checkin Question

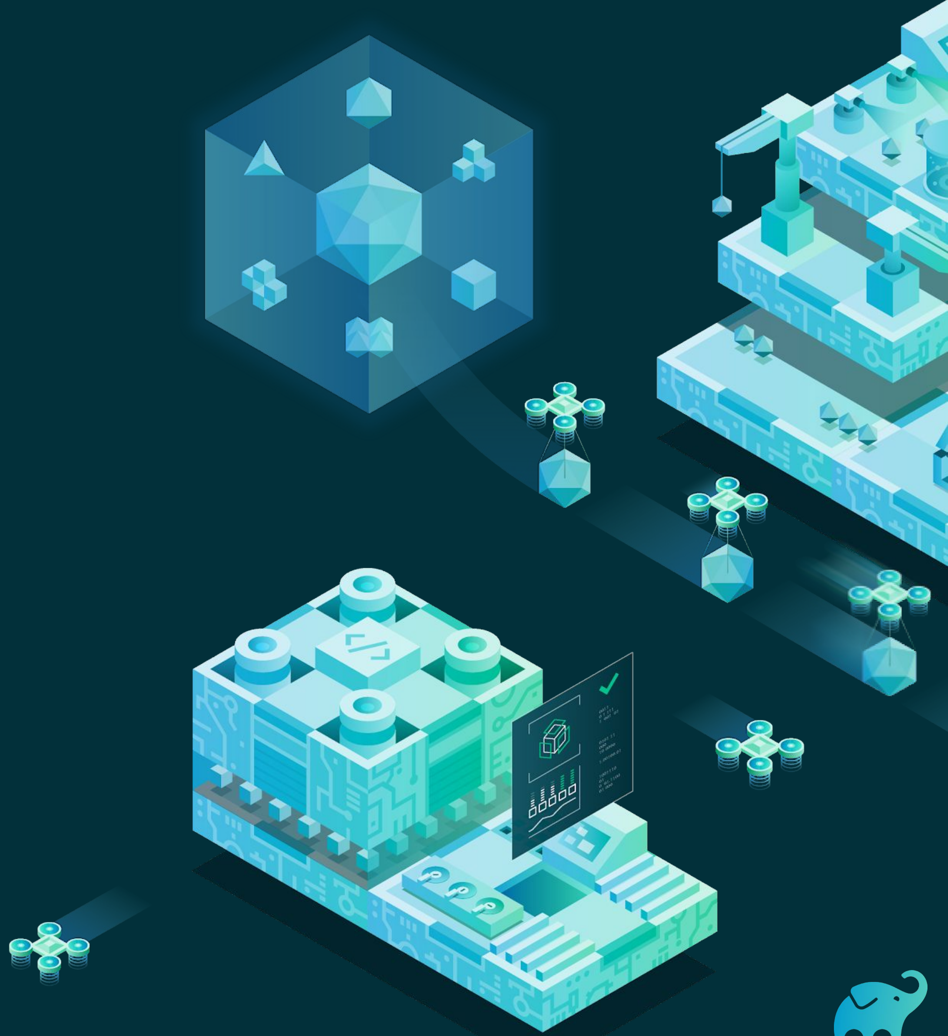
- What is the term used to describe adding a plugin to a Gradle configuration?
    a. add
    b. employ
    c. apply
    d. administer

Checkin Question

- What are the 3 types of plugins in Gradle?
    a. Default, custom & private
    b. Default, community & local
    c. Built-in, custom & local
    d. Built-in, community & local

Tasks

- About Gradle ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins Build Tool ✓
- Tasks
- Dependency Management
- Publishing
- Multi-Project Builds



Tasks

- Tasks are the basic unit of work in Gradle
  - Compile
  - Test
  - Generate docs
- Tasks belong to projects
  - Different projects can have different tasks
- Categories:
  - Built-in
    - init
    - wrapper
  - Provided by Plugins
    - compile
    - test
  - Custom - locally defined

Task Concepts in Gradle

- Inputs → Action → Outputs
  - Inputs read by task
    - Configuration properties
    - Files
    - Can be outputs from other tasks
  - Action
    - What the task does when executing
  - Outputs
    - eg. Files produced by action
    - Often outputs are put in the build directory
      - Convention across many languages, not just Java
      - Exclude from version control
- Dependency & ordering
  - Other tasks that need to run before
  - Tasks that need to run after

```
$ ./gradlew :app:tasks —all
$ ./gradlew :app:compileJava
$ ./gradlew :app:build
$ ./gradlew :app:test
```

Task Concepts in Gradle - eg. Test Task

- Configuration properties → Inputs
  - eg. Compile options, where to store test reports

```
tasks.withType<JavaCompile> {
    options.isDebug = false
}
```

```
tasks.named<Test>("test") {
    reports.junitXml.outputLocation.set(layout.buildDirectory.dir("reports/tests/xml"))
}
```

- Actions → Outputs
  - Run tests and generate report
  - Put in build directory
- Dependency & ordering
  - Depends on compile source & tests
  - See task tree plugins

```
:app:test
+--- :app:classes
|    +--- :app:compileJava
|    `--- :app:processResources
`--- :app:testClasses
     +--- :app:compileTestJava
     |    `--- :app:classes
     |         +--- :app:compileJava
     |         `--- :app:processResources
     `--- :app:processTestResources
```

Tip: Logging Options

- --console=plain or --console=verbose
  - Will show dependent tasks executed
- -q
  - Hide log messages, so that only the output of the tasks is shown
- --dry-run
  - Show what will happen without executing

- Can define logging options in gradle.properties

```
org.gradle.console=verbose
```

Task Outcome Labels

- No label (or EXECUTED) → Task actions were executed
- UP-TO-DATE → Task actions not executed, previous output used
- Gradle uses cached task outputs if inputs were same and task is deterministic
    - This is a huge time-saving feature in Gradle
- More labels in doc page

```
> Task :app:compileJava UP-TO-DATE
> Task :app:processResources NO-SOURCE
> Task :app:classes UP-TO-DATE
> Task :app:compileTestJava
> Task :app:processTestResources NO-SOURCE
> Task :app:testClasses
> Task :app:test
```

Custom Tasks

- register method on tasks property
- Can build upon existing types

```
tasks.register<Zip>("zipTestResult") {
    archiveFileName.set("test-results.zip")
    destinationDirectory.set(layout.buildDirectory)

    from(layout.buildDirectory.dir("test-results"))
}
```

Custom Tasks - Define Dependency

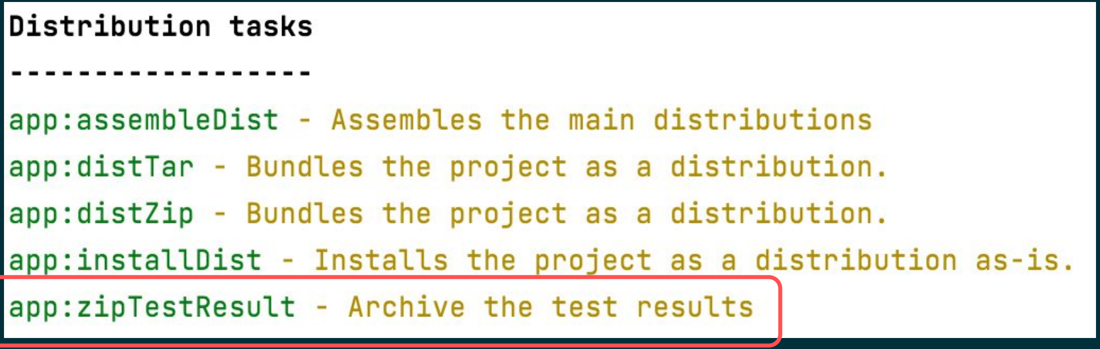
- dependsOn to define dependency
  - Task ordering docs
    - eg. finalizedBy can also define dependency
- Can also specify task dependencies using I/O
  - Inferred task dependencies
  - Linking outputs to inputs

```
tasks.register<Zip>("zipTestResult") {
    dependsOn("test")
    // inputs.files(tasks.test)

    archiveFileName.set("test-results.zip")
    destinationDirectory.set(layout.buildDirectory)

    from(layout.buildDirectory.dir("test-results"))
    // from(tasks.test) { include("**/*.xml") }
}
```

Custom Task Grouping

- ./gradlew tasks --all
- group property
- description property

```
tasks.register<Zip>("zipTestResult") {
    group = "distribution"
    description = "Archive the test results"
    dependsOn("test")

    archiveFileName.set("test-results.zip")
    destinationDirectory.set(layout.buildDirectory)

    from(layout.buildDirectory.dir("test-results"))
}
```



```
Distribution tasks
------------------
app:assembleDist - Assembles the main distributions
app:distTar - Bundles the project as a distribution.
app:distZip - Bundles the project as a distribution.
app:installDist - Installs the project as a distribution as-is.
app:zipTestResult - Archive the test results
```

Passing Parameters

- ./gradlew zipTestResult -PzipName=tr.zip
- providers.gradleProperty()

```
tasks.register<Zip>("zipTestResult") {
    group = "distribution"
    description = "Archive the test results"
    dependsOn("test")

    val zipName = providers.gradleProperty("zipName").orElse("test-results.zip")
    archiveFileName.set(zipName)
    destinationDirectory.set(layout.buildDirectory)

    from(layout.buildDirectory.dir("test-results"))
}
```

Task Configuration - Configure Only if Requested

- Task creation → configuration is read
- register → define task, without creating it
  - configuration not read unless task called
- named → define task configuration, without creating it
  - configuration not read unless task called
- Task configuration avoidance doc
  - Avoid using tasks.create and eager tasks

Avoid These

```
tasks.create("someTask") {
    println("creating someTask")
}

task("anEagerTask") {
    println("eager task!")
}
```

Task Actions - doFirst & doLast

- Define custom task actions
  - Recommend for simple tasks only
  - Try to leverage existing task types
  - Even this example can be done with existing Delete task type

```
tasks.register("deleteTestArchive") {
    doLast {
        layout.buildDirectory.file("test-results.zip").get().asFile.delete()
    }
}
```

Build Scan™

- Example use-case: Share scan with teammate who is helping with debugging
- Free public service to get insights provided by Gradle Inc.
- Uploads Gradle metadata only, no source code
- Can invoke using --scan
    - Need to register email with service
- Can include following snippet in settings.gradle.kts

```
plugins {
   id("com.gradle.enterprise") version "3.6.1"
}
gradleEnterprise {
   buildScan {
       termsOfServiceUrl = "https://gradle.com/terms-of-service"
       termsOfServiceAgree = "yes"
   }
}
```

- Recommend only use for personal projects
    - Gradle Enterprise allows hosting private Build Scan service
        - Provides features for additional use-cases

Checkin Question

- What is the method to define a task so its configuration will not be read unless the task is executed?
    a. create
    b. named
    c. record
    d. register

Checkin Question

- What is the method to define that a task depends on another task executing first?
  a. addDepends
  b. finalizedBy
  c. dependsOn
  d. depsOn

Checkin Question

- What is the method to define that another task be executed after a task has finished?

  a. addDepends

  b. finalizedBy

  c. dependsOn

  d. depsOn

Hands-on Exercise 2

- Introduction to Gradle Build Tool Exercises

- See available tasks
- Run test task and inspect report
- Apply and use community plugin
- Create custom tasks

Dependency Management

- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins ✓
- Tasks ✓
- Dependency Management
- Publishing
- Multi-Project Builds

Dependency

- Pointer to another piece of software needed by a Gradle project
- Types
  - Module
  - Other Gradle project
  - File (not recommended)

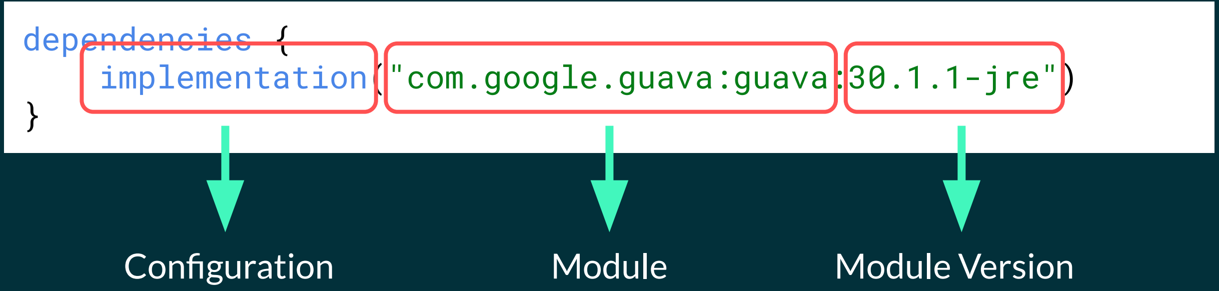
Repository

- Hosts a set of modules
  - Each module can have several releases
- Where to download modules from
- [Maven central]

```
repositories {
    mavenCentral()
}
```

Module Dependency

- Module is a piece of software that evolves over time
  - Typically has multiple releases
- Module uniquely defined by group and library name
- Most common type of dependency



```
dependencies {
    implementation("com.google.guava:guava:30.1.1-jre")
}
```

Groovy Only

```
dependencies {
    implementation group: "com.google.guava", name: "guava", version: "31.1-jre"
}
```

Configuration of Dependencies

- Dependencies grouped together by name for a scope
- Tasks can access the specific dependencies needed

For compiling source

For compiling tests

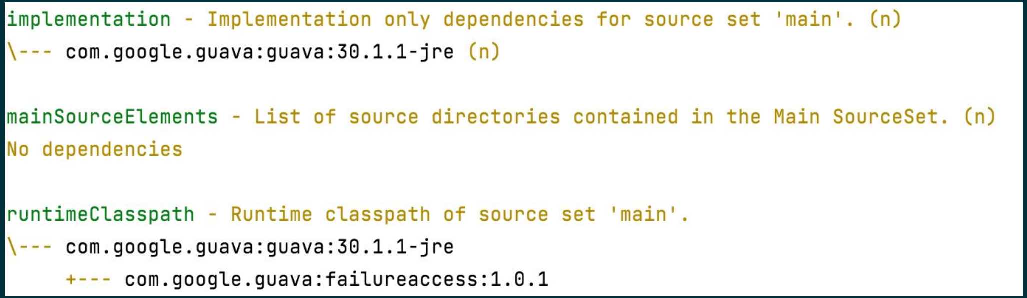
Needed during runtime

```
dependencies {
    implementation("com.google.guava:guava:30.1.1-jre")
    testImplementation("org.junit.jupiter:junit-jupiter:5.8.1")
}
```

Transitive Dependencies

- Dependencies can have their own dependencies
  - For Java these are defined in POM files with artifacts - example
- Gradle will fetch these automatically
- Can see all dependencies:
  - ./gradlew :<subproject>:dependencies [--configuration <name>]
  - Notice other dependency configurations

subproject required



```
implementation - Implementation only dependencies for source set 'main'. (n)
\--- com.google.guava:guava:30.1.1-jre (n)

mainSourceElements - List of source directories contained in the Main SourceSet. (n)
No dependencies

runtimeClasspath - Runtime classpath of source set 'main'.
\--- com.google.guava:guava:30.1.1-jre
     +--- com.google.guava:failureaccess:1.0.1
```

- Explore a dependency in detail:
  - ./gradlew :<subproject>:dependencyInsight --dependency <module>

Dependency Configuration - Most Common

api	Public facing specification, eg. classes used in public interface
implementation	Internally used, eg. computational libraries
runtimeOnly	Required when running application, eg. specific logging library
testImplementation	Required by tests

- api vs implementation Example in Gradle docs
- Further configurations doc

Module Version

- 2.3 or later (if there is a conflict - more on this later)

```
implementation("org.company:some-lib:2.3")
```

- Strictly 2.3

```
implementation("org.company:some-lib:2.3!!")
```

- Latest 2.+ version

```
implementation("org.company:some-lib:2.+")
```

- SNAPSHOT of 2.7

```
implementation("org.company:some-lib:2.7-SNAPSHOT")
```

- Flexibility to allow various specifications

Dynamic and Changing Versions

- Dynamic → Version may change, eg. "latest version"
  - eg. "2.+" may be 2.6 one day then 2.7 the next
- Changing → Artifact for same version may update
  - eg. SNAPSHOT artifact keeps changing
- By default Gradle caches both Dynamic and Changing versions for 24 hours
  - Can configure as shown below

```
configurations.all {
    resolutionStrategy.cacheDynamicVersionsFor(4, "hours")
    resolutionStrategy.cacheChangingModulesFor(10, "minutes")
}
```

- **Note**: Determining versions does impact performance
- Module artifacts are downloaded to ~/.gradle/caches/modules-2/files-2.1/
  - Cached for future builds

Dependency Version Conflict Resolution

- Multiple versions of a dependency requested
    - Either directly or transitively
- Highest version will be chosen unless strict version specified

```
dependencies {
    implementation("com.google.guava:guava:30.1.1-jre")
    implementation("com.google.inject:guice:5.1.0")
}
```

```
+--- com.google.guava:guava:30.1.1-jre
|    +--- com.google.guava:failureaccess:1.0.1
|    +--- com.google.guava:listenablefuture:9999.0-empty-to-avoid-conflict-with-guava
|    +--- com.google.code.findbugs:jsr305:3.0.2
|    +--- org.checkerframework:checker-qual:3.8.0
|    +--- com.google.errorprone:error_prone_annotations:2.5.1
|    \--- com.google.j2objc:j2objc-annotations:1.3
\--- com.google.inject:guice:5.1.0
     +--- javax.inject:javax.inject:1
     +--- aopalliance:aopalliance:1.0
     \--- com.google.guava:guava:30.1-jre -> 30.1.1-jre (*)
```

Documenting a Dependency

- Can document why a dependency was chosen

```
implementation("org.company:some-lib:2.3!!") {
    because("We require the magicFunc in HelperUtils which was removed after 2.3")
}
```

Checkin Question

- What is the point of having dependency configurations?
  a. Makes configuration more readable
  b. Group dependencies by similar size
  c. Group dependencies by scope
  d. We like configurations in our configuration

Checkin Question

- Version 3.+ may resolve to 3.6 one day, and 3.7 the next. What is this kind of version called?
  a. Variable
  b. Runtime
  c. Changing
  d. Dynamic

Checkin Question

- What is the Gradle task to view dependencies of a subproject?

  a. deps

  b. showDeps

  c. showDependencies

  d. dependencies

Hands-on Exercise 3

- Introduction to Gradle Build Tool Exercises

- See dependencies for a project
- Adding dependencies
- Examining dependency version conflict resolution

Publishing

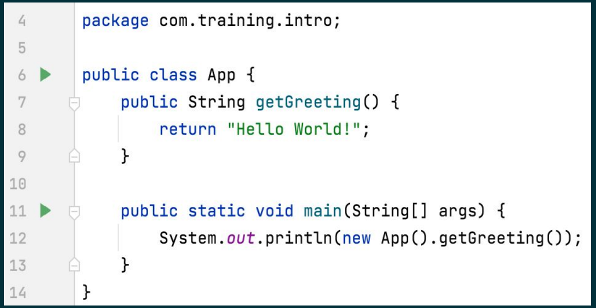
- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins ✓
- Tasks ✓
- Dependency Management ✓
- Publishing
- Multi-Project Builds

Publishing

- Publish software artifacts to repositories
  - For Java projects to Maven/Ivy repositories
- Metadata files automatically generated
  - eg. POM files for Java

```
plugins {
    id("java-library")
    id("maven-publish")
}

publishing {
    publications {
        create<MavenPublication>("library") {
            from(components["java"])
        }
    }
    repositories {
        maven {
            url = uri(layout.buildDirectory.dir("repo"))
        }
    }
}
```

- Build
- Test
- Push / Publish



```
package com.training.intro;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

    public static void main(String[] args) {
        System.out.println(new App().getGreeting());
    }
}
```

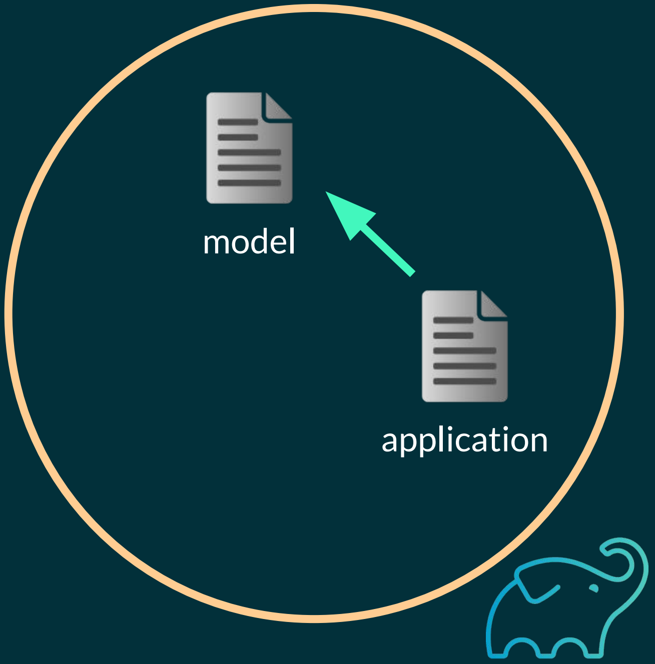
Multi-Project Builds

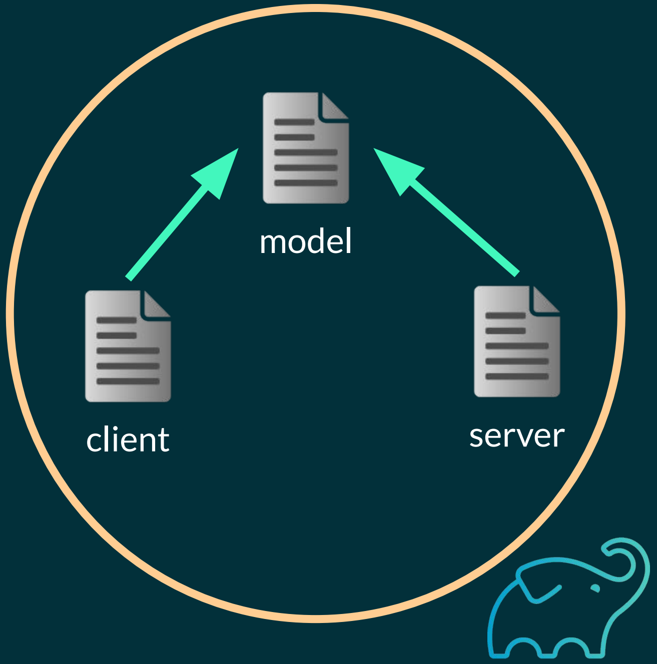
- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins ✓
- Tasks ✓
- Dependency Management ✓
- Publishing ✓
- Multi-Project Builds

What is a Multi-Project Build?

- Large applications are broken into parts
  - Easier to manage
  - Allows sharing small reusable parts
- In Gradle, each part is a subproject, can manage them all together
  - Subprojects can depend on other subprojects

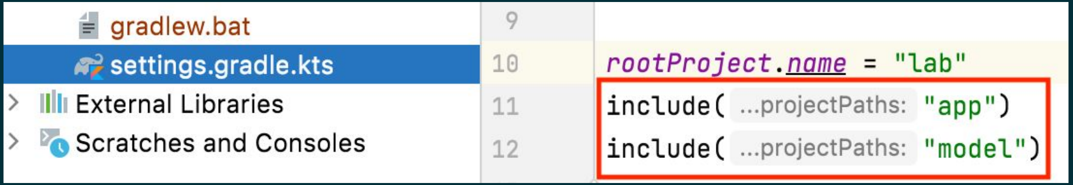
Simple Examples





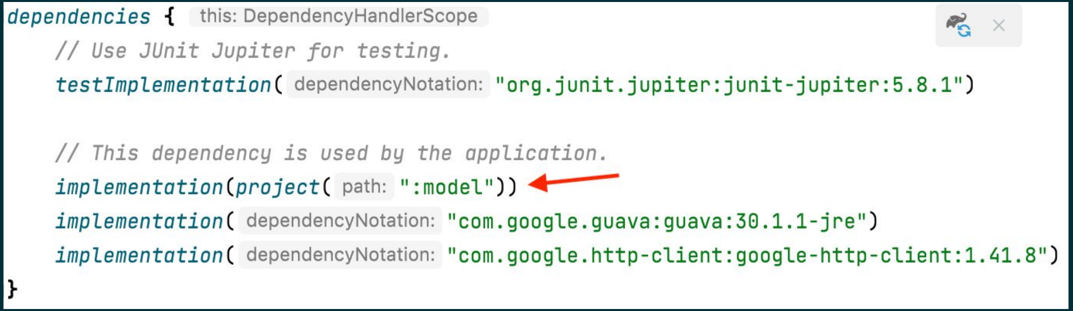
Multi-Project Structure

- Each subproject has its own directory and build.gradle.kts
- Each subproject registered in settings.gradle.kts
- Subprojects can depend on other subprojects
  - implementation(project(":other-subproject"))

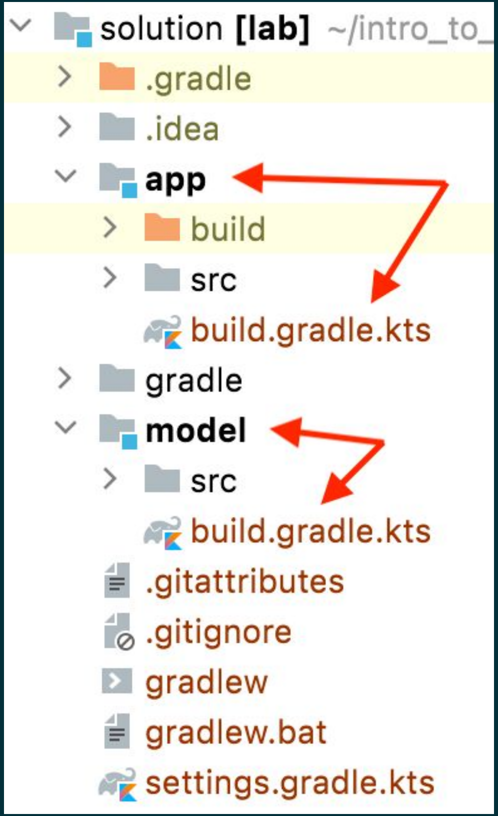


```
gradlew.bat
settings.gradle.kts
External Libraries
Scratches and Consoles
```

```
rootProject.name = "lab"
include( ...projectPaths: "app")
include( ...projectPaths: "model")
```



```
dependencies {  this: DependencyHandlerScope
    // Use JUnit Jupiter for testing.
    testImplementation( dependencyNotation: "org.junit.jupiter:junit-jupiter:5.8.1")

    // This dependency is used by the application.
    implementation(project( path: ":model"))
    implementation( dependencyNotation: "com.google.guava:guava:30.1.1-jre")
    implementation( dependencyNotation: "com.google.http-client:google-http-client:1.41.8")
}
```



```
solution [lab] ~/intro_to_
  .gradle
  .idea
  app
    build
    src
    build.gradle.kts
  gradle
  model
    src
    build.gradle.kts
  .gitattributes
  .gitignore
  gradlew
  gradlew.bat
  settings.gradle.kts
```

Sharing Common Configuration

- Common configuration referenced from shared location
  - Easier to maintain large multi-projects
  - buildSrc → Special subproject
  - Best practice: Use for custom tasks
    - Build files → configuration only
    - Custom tasks → buildSrc

buildSrc/build.gradle.kts

```
plugins {
    `kotlin-dsl`
}

repositories {
    gradlePluginPortal()
}
```

app/build.gradle.kts

```
plugins {
    application
    id("shared-build-conventions")
}

application {
    mainClass.set("com.gradle.lab.App")
}
```

model/build.gradle.kts

```
plugins {
    id("shared-build-conventions")
}
```

buildSrc/src/main/kotlin/shared-build-conventions.gradle.kts

```
plugins {
    java
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter:5.8.1")
    implementation("com.google.guava:guava:30.1.1-jre")
}

tasks.named<Test>("test") {
    useJUnitPlatform()
}
```

Legacy: allprojects & subprojects

- Older projects still use this approach
- Downside: Can’t tell where shared configuration comes from
- Example

Checkin Question

- What is the purpose of the buildSrc subproject?
    a. Define common configuration in one place
    b. Define custom tasks separately, keeping build files as configuration only
    c. Defining different shared configuration for different subprojects
    d. All of the above

Checkin Question

- How do you define the buildSrc subproject in the settings file?

  a. include("buildSrc")

  b. include(":buildSrc")

  c. include("buildSrc/build.gradle.kts")

  d. You don't define buildSrc in the settings file

Hands-on Exercise 4

- Introduction to Gradle Build Tool Exercises

- Add new project to Gradle
- Set one project as dependency of another
- Sharing common configuration between projects

Summary

- About Gradle Build Tool ✓
- Build Configuration ✓
- Build Lifecycle ✓
- Plugins ✓
- Tasks ✓
- Dependency Management ✓
- Publishing ✓
- Multi-Project Builds ✓

Thank you!

Objectives

- Understand core Gradle Build Tool concepts
- Hands-on exercises to get you going

Feedback

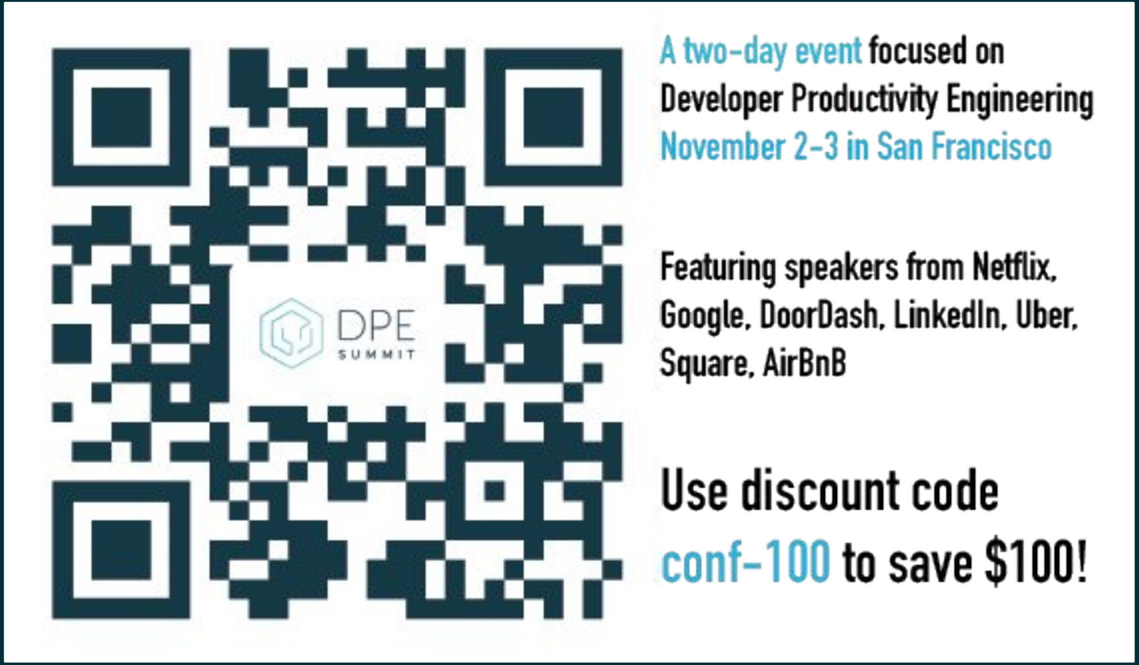
- training@gradle.com

Resources

- https://docs.gradle.org/
- https://discuss.gradle.org/
- https://newsletter.gradle.com/
- https://plugins.gradle.org/
- https://gradle-community.slack.com/

Developer Productivity Engineering Summit

- When - November 2-3
- Where - San Francisco, CA USA
- More Info - https://dpesummit.com/



A two-day event focused on Developer Productivity Engineering November 2-3 in San Francisco

Featuring speakers from Netflix, Google, DoorDash, LinkedIn, Uber, Square, AirBnB

Use discount code conf-100 to save $100!