Getting Started with Gradle

Speaker

Sterling Greene (sterling@gradle.com)

Principal Engineer, Gradle Inc

Clone the example project

Agenda

- Gradle Project History
- Gradle Best Practices
- Gradle Basics
- Java Plugin

Gradle

About the project

Gradle

Gradle is a build and automation tool.

Gradle can automate the building, testing, publishing, deployment and more of software packages or other types of projects such as generated static websites, generated documentation or indeed anything else.

- JVM based
- Implemented in Java (core) and Groovy (outer layers)

http://www.gradle.org

Gradle Project

- Open Source, Apache v2 license - Completely free to use
- Source code on Github - github.com/gradle/gradle
- Active user community, centered around discuss.gradle.org
- Frequent releases (minor releases roughly every 4-8 weeks)
- Strong quality commitment
  - Extensive automated testing (including documentation) - builds.gradle.org
  - Backwards compatibility & feature lifecycle policy
  - Thorough design and review process
- Developed by domain experts
  - Gradle, Inc. staff and community contributors

Gradle Documentation

- User Guide
  - 400+ pages, many samples
  - single page HTML, multi page HTML, PDF
  - Search tip: Full text search on single page HTML
- Build Language Reference (gradle.org/docs/current/dsl/)
  - Best starting point when authoring build scripts
  - Javadoc-like, but higher-level
  - Links into Javadoc and Groovydoc
- Gradle Distribution
  - Includes user guide, build language reference, and **many sample builds**
  - `gradle-all` vs. `gradle-bin`

Other Gradle Resources

- discuss.gradle.org
  - Best place to ask questions
  - Expanding knowledge base
- Books, several are free!
- StackOverflow, tagged with gradle

Gradle Evolution

Gradle 1.x

- 13 releases from June 2012 (1.0) to April 2014 (1.12)
- New features in each release
- Better performance
- More extensibility

Gradle 2.0 (July 2014)

- New baseline for backwards compatibility
  - Removed deprecated APIs
  - Upgraded to Groovy 2.x (from 1.8)
  - Support for Java 8
- Performance and extensibility are a major focus for Gradle 2.x
- Most build scripts still work between Gradle 1.x and 2.x
- Plugins can be built to support both 1.x and 2.x

Gradle 2.1 (September 2014)

- Gradle Plugin Portal introduced
  - Searchable location for community Gradle plugins
- Incremental Java compilation

Gradle 2.2 (November 2014)

- Component selection rules
  - Reject versions of particular dependencies based on group and name
- Module replacement rules
  - Provide replacement dependencies (Google Collections -> Guava)
- Sharing configuration files across builds

Gradle 2.3 (February 2015)

- Artifact query resolution
  - Fine grained access to dependency artifacts

Gradle 2.4 (May 2015)

- Fastest Gradle yet
  - 20% to 40% drop in build times
  - Reduced memory consumption for Java 7/8 builds
- Native (C/C++) parallel compilation
  - 50% better build times
- Support for S3 backed Maven/Ivy repositories

Gradle 2.5 (July 2015)

- Continuous build (-t)
  - Automatically rebuild when sources change
- Dependency substitution rules
  - Replace external dependency with a local subproject

Gradle 2.6 (August 2015)

- Play 2.3.x support
- Hot-reload support when in continuous build
- Introduction of Gradle TestKit
  - Test plugins and build logic with functional tests

Gradle 2.7 (September 2015)

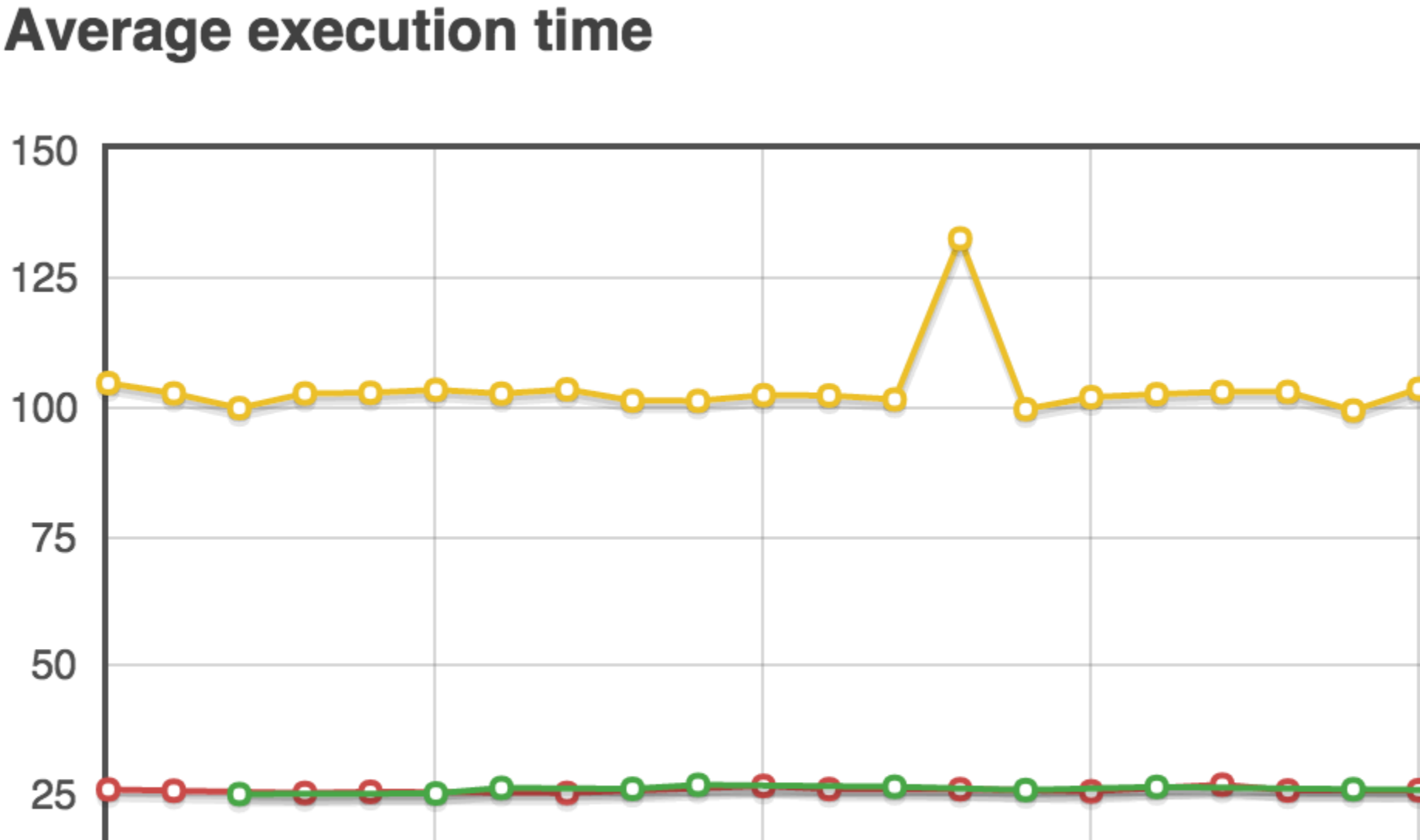
- Preemptive authenication with Maven repositories (e.g., GitHub)
- Play 2.4.x support
- TestKit improvements

Gradle 2.8 (October 2015)

- Groovy 2.4.4 upgrade
- Compiler daemon reuse when in continuous build
- Faster incremental build
  - For very large projects (100k's of sources), dramatic drop in no-op build times

Gradle 2.9 (ETA: November 2015)

- More TestKit improvements
- Even faster incremental build



Running Gradle

Getting Information

Print basic help information:

```
$ gradle help
```

Print command line options:

```
$ gradle -?
```

Print the available tasks in a project:

```
$ gradle tasks
```

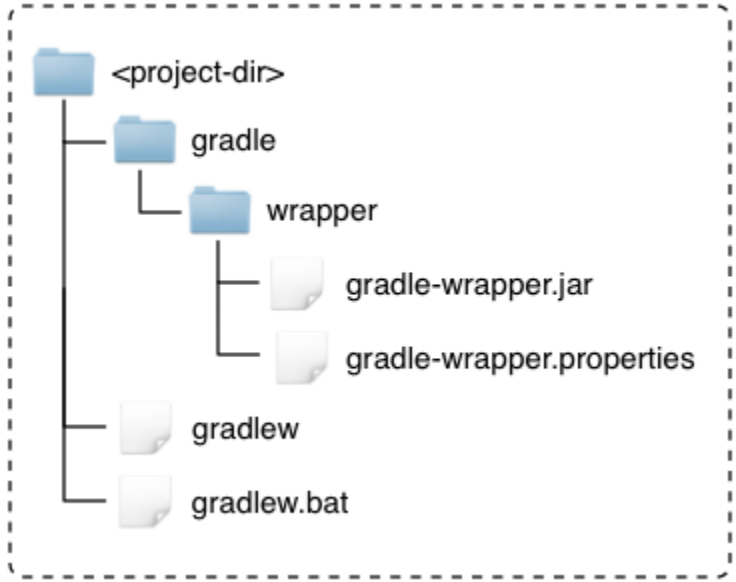
Best Practices for Running Gradle

- Always use the wrapper
- Prefer using the daemon for desktop builds

Gradle Wrapper

A way to bootstrap Gradle installations.

- **gradle-wrapper.jar:** Micro-library for downloading distribution
- **gradle-wrapper.properties:** Defines distribution download URL and options
- **gradlew:** Gradle executable for *nix systems
- **gradlew.bat:** Gradle executable for Windows systems



Usage example:

```
$ ./gradlew build
```

Wrapper task

- Wrapper task is built-in and generates:
  - wrapper scripts
  - wrapper jar
  - wrapper properties

```
$ gradle wrapper --gradle-version=2.6
```

Gradle Build Daemon

Makes builds start faster.

- Enable with
  - `--daemon` command line option
  - `org.gradle.daemon=true` in `gradle.properties`
  - `-Dorg.gradle.daemon=true` in `GRADLE_OPTS`
- Force shutdown with `gradle --stop`

Will be used in the future for more optimizations.

Tasks

Tasks

Tasks are the basic unit of work in Gradle.

- created & configured by the user
- **executed by Gradle**

```
task helloWorld {
    doLast {
        println "Hello World!"
    }
}
```

`task` is a keyword in the Gradle DSL.

All tasks implement the `Task` interface.

Task Actions

Tasks have a list of actions.

```
task hello {
  doLast {
    println "World!"
  }
  doFirst {
    println "Hello"
  }
}
```

Most tasks have one useful main action.

`doLast()` and `doFirst()` can be used to decorate the action.

Quiz

- What would we expect to happen with doLast() if a preceeding action fails?
- What would we expect to happen with doFirst() if the task is skipped?

Quiz (Answers)

- What would we expect to happen with doLast() if a preceeding action fails?

The task fails execution and the rest of the task's actions are not executed.

- What would we expect to happen with doFirst() if the task is skipped?

The doFirst action is never executed.

Abbreviated Task Name Execution

Save your fingers by only typing the bare minimum to identify a task.

```
task myNameIsPrettyLong {
  doLast {
      println "long task!"
  }
}
```

```
> gradle mNIPL
```

It understands camel case.

DSL Syntax and Tasks

```
// << is synonymous with doLast()
// we'll use doLast() from here on
task hello << { println "Hello" }

// access existing task via its name
hello.dependsOn otherTask

// configure existing task via closure
hello {
  dependsOn otherTask
}

// configure new task
task greet {
  dependsOn otherTask
  doLast { println "Hello Gradler!" }
}
```

Task Types

You will usually use tasks of a certain type, that provide useful behavior (e.g. copy files).

```
task copyFiles(type: Copy) {
    // Only configuration (actions are defined by the type)
    from('someDirectory')
    into('anotherDirectory')
}
```

Task is of type `Copy`. Configure it using its API.

If you don't specify a type, you get a `DefaultTask`.

Task Types and API

```
task hello {
    onlyIf { day == "monday" }
    doFirst { println "Hello" }
}
```

The `onlyIf()` method is a method of all tasks (i.e. part of `Task` interface).

```
task copy(type: Copy) {
  from "someDir"
  into "anotherDir"
}
```

The `from()` method here is part of the `Copy` API.

The task API allows the task to be configured.

Quiz

What does each individual line do?

```
//
task whatAmIDoing

//
whatAmIDoing

//
tasks.whatAmIDoing

//
whatAmIDoing {}

//
//
whatAmIDoing << {}
```

Quiz (Answers)

What does each individual line do?

```
// Defines a task called 'whatAmIDoing'
task whatAmIDoing

// Refers to a task called 'whatAmIDoing'
whatAmIDoing

// Refers to a task called 'whatAmIDoing' through the TaskContainer
tasks.whatAmIDoing

// Performs additional configuration on a task called 'whatAmIDoing'
whatAmIDoing {}

// Adds an action to the list of actions to task called 'whatAmIDoing'
// This is the same as whatAmIDoing.doLast {}
whatAmIDoing << {}
```

Implementing Task Types

- POJO extending `DefaultTask`
- Declare action with `@org.gradle.api.tasks.TaskAction`

```
class FtpTask extends DefaultTask {
  String host = "docs.mycompany.com"

  @TaskAction
  void ftp() {
    // do something complicated
  }
}
```

Task Type > Ad-hoc Task

Prefer implementing task types to implementing ad-hoc tasks.

- Avoid global properties and methods
- Separate the imperative from the declarative
- Easy to refactor (e.g. from build script to Jar)
- Easier to utilize other Gradle features

Ad-hoc tasks are OK for small simple tasks.

Task Dependencies

- Tasks can depend on each other
- Semantic relationship (A produces something that B consumes)
- Executed tasks form a directed acyclic graph

```
task foo

// multiple ways to declare task dependencies
task bar(dependsOn: foo) // won't use this syntax from here on
bar { dependsOn foo }
bar.dependsOn foo
```

Build Lifecycle

- Initialization Phase
  - Configure environment (init.gradle, gradle.properties)
  - Find projects and build scripts (settings.gradle)
- Configuration Phase
  - Evaluate all build scripts
  - Build object model (Gradle -> Project -> Task, etc.)
  - Build task execution graph
- Execution Phase
  - Execute (subset of) tasks

A key concept to grasp.

Quiz

What happens here?

```
//
//
//
//
task bar {
    doLast {
        dependsOn foo
    }
}
```

Quiz (Answers)

What happens here?

```
// Dependency configuration happens during execution phase
// so 'bar' would execute without waiting for 'foo'
// In the current version of Gradle, this throws an Exception
// to catch this kind of error.
task bar {
    doLast {
        dependsOn foo
    }
}
```

Java Plugin

Java Plugins

- java-base
  - Provides additional Task types
  - Defines rules for conventions
  - Adds declarative elements to the DSL (e.g. SourceSet)
- java
  - Adds task instances to the project
  - Adds default values to task instances
  - Adds source sets for production and test code
  - Configures the dependency management for Java projects (adds scopes for compile, runtime, ...)

Clean Task

- By default clean deletes the buildDir
- You can specify additional files to delete
- name: 'clean', type: Delete

```
clean {
  delete "fooDir", "bar.txt",
  fileTree("texts").matching { ... }
}
```

Javadoc Task

- Provides all the options of the Javadoc command
- name: 'javadoc', type: Javadoc
- input: sourceSets.main.java, sourceSets.main.compileClasspath

```
javadoc {
  maxMemory = "512M"
  include "org/gradle/api/**"
  title = "Gradle API $version"
}
```

Resources Tasks

- Usually configured via the source set
- Can use the powerful Copy API
- name: processResources, processTestResources
- type: Copy
- input: sourceSets.main(test).resources

Compile Tasks

- Usually configured via the source set
- Provides all the options of the Ant javac task
- name: compile, testCompile, type: JavaCompile
- input: sourceSets.main(test).java , sourceSets.main(test).compileClasspath

```
compileJava {
  options.fork = true
  options.forkOptions.with {
    memoryMaximumSize = "512M"
  }
}
```

Classes Tasks

- Aggregates compile related tasks
- name: classes, testClasses, type: DefaultTask
- dependsOn: compile|testCompile, processResources|processTestResources

Jar Task

- Content of the Jar: production classes
- name: 'jar', type: Jar
- input: sourceSets.main.output

```
jar {
   //you can add more content:
   from sourceSets.main.allJava
   from zipTree("lib/someJar.jar")
}
```

Test Task

- Support for JUnit and TestNG
- Parallel testing
- Custom fork frequency
- Test listeners
- Tests auto-detected in sourceSets.test.output
- name: 'test', type: Test
- input: sourceSets.test.output, sourceSets.test.runtimeClasspath

Test Task Example

```
test {
  jvmArgs "-Xmx512M"
  scanForTestClasses = false //disables auto-detection
  include "**/tests/special/**/*Test.class"
  exclude "**/Old*Test.class"
  forkEvery = 30
  maxParallelForks = guessMaxForks()
}

def guessMaxForks() {
  int processors = Runtime.runtime.availableProcessors()
  Math.max(2, processors.intdiv(2))
}
```

Test Task Listeners

```
test {
  beforeTest { desc ->
    // do something
  }
  afterTest { desc, result ->
    // do something
  }
  afterSuite { desc, result ->
    // do something
  }
}
```

Multi-Project Builds

Multi-Project Builds

- Flexible directory layout
- Configuration injection
- Project dependencies & partial builds

Configuration Injection

- **ultimateApp**
  - api
  - services
    - webservice
  - shared

```
subprojects {
  apply plugin: "java"
  dependencies {
    testCompile "junit:junit:4.7"
  }
  test {
    jvmArgs "-Xmx512M"
  }
}
```

Filtered Injection

- **ultimateApp**
  - api
  - services
    - webservice
  - shared

```
configure(nonWebProjects()) {
  jar.manifest.attributes Implementor: "Gradleware"
}

def nonWebProjects() {
  subprojects.findAll { !it.name.startsWith("web") }
}
```

Project Dependencies

- ultimateApp
  - **api**
  - services
    - webservice
  - shared

```
dependencies {
  compile "commons-lang:commons-lang:2.4"
  compile project(":shared")
}
```

Partial Builds

- ultimateApp
  - **api**
  - services
    - webservice
  - shared

```
>gradle build
>gradle buildDependents
>gradle buildNeeded
```

Name Matching Execution

- **ultimateApp**
  - api
  - services
    - webservice
  - shared

```
>gradle build
>gradle classes
>gradle war
```

Task/Project Paths

- For projects and tasks there is a fully qualified path notation:
  - : (root project)
  - :clean (the clean task of the root project)
  - :api (the api project)
  - :services:webservice (the webservice project)
  - :services:webservice:clean (the clean task of webservice)

```
>gradle :api:classes
```

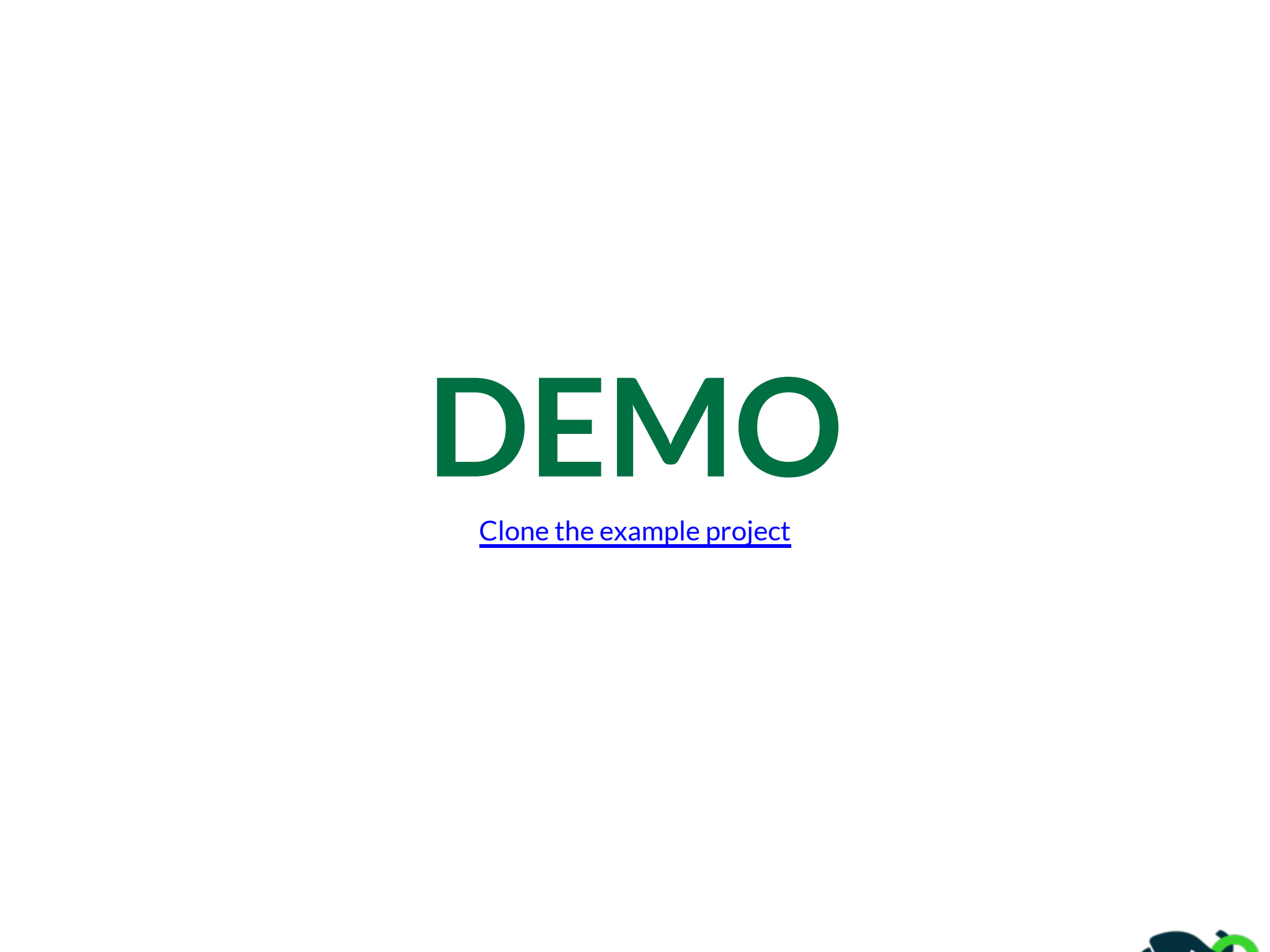
Defining a Multi-Project Build

- settings.gradle (location defines the root)
- root project is implicitly included

```
//declare projects:
include "api", "shared", "services:webservice"

//Everything is configurable:

//default: root dir name
rootProject.name = "main"

//default: 'api' dir
project(":api").projectDir = file("/myLocation")

//default: 'build.gradle'
project(":shared").buildFileName = "shared.gradle"
```

DEMO

Clone the example project



Thank You!

- Thank you for attending!
- Questions?
- Feedback?
- Gradle Home
- Get more help!