`Gradle` vs `Maven`

Irei explicar mostrando as diferenças que existe com o `Gradle` sobre o `Maven`. É interessante notar que são duas ferramentas para o processo build do seu projeto em qualquer linguagem da `JVM`. O `Gradle` é uma ferramenta mais robusta e interessante que o `Maven`, e é isso que eu vou apresentar adiante.
Durante anos utilizamos o `Maven` para cuidar da automação do processo de build. Essa ferramenta se tornou um padrão de mercado e ajudou inúmeros times. Porém, não podemos negar que ela também tem suas deficiências. A escolha de `XML` para o descritor de projeto tem seus problemas.

Por que o `Gradle`?

Os scripts do `Gradle` são declarativos, de fácil leitura, e espressivo. Escrever o código em `Groovy` ao invés de `XML`, e seguir a filosofia que o `Gradle` define de **build-by-convention** reduz significativamente o tamanho de um script e é muito mais legível.

Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.mycompany.app</groupId>
   <artifactId>my-app</artifactId>
   <packaging>jar</packaging>
   <version>1.0-SNAPSHOT</version>

   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.11</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

Gradle

```
apply plugin: 'java'
group = 'com.mycompany.app'
archivesBaseName = 'my-app'
version = '1.0-SNAPSHOT'

repositories {
   mavenCentral()
}

dependencies {
   testCompile 'junit:junit:4.11'
}
```

Vamos dar uma olhada no que diferencia o `Gradle` de seus concorrentes: seu conjunto de recursos nativos:

DSL expressiva e API rica

A chave para utilizar todo o poder do `Gradle` é conhecer a fundo sua API, nela podemos encontrar todas as classes e entender bem tudo o que o `Gradle` já nos entrega nativamente para trabalharmos de forma eficiente e rápida. Como toda documentação no mundo `Java`, ela tem sua pagina `HTML` para ser acessada a qualquer momento. Segue o link: Gradle doc;

`Gradle` é `Groovy`

`Ant` e `Maven` são puro `XML`, e `XML` pode ser de primeiro encontro muito simples para escrever e ler, porém, com o tempo pode se tornar muito difícil para dar manutenção e fazer

melhorias. `Gradle` é escrito em `Groovy`, uma linguagem que utiliza o melhor do `Java` para ser o mais simples e dinâmica possível, por esse motivo temos uma facilidade muito grande em escrever nosso arquivo de build em forma de script.

Convenções flexíveis

Uma das grandes ideias de `Gradle` é ser flexível quanto a padrões para seus projetos. Cada projeto `Java` tem uma convenção básica e o `Gradle` sabe exatamente onde fica o código fonte e as classes de teste dessa convenção, e como compilar o código, executar testes unitários, gerar o javadoc, e criar uma release do seu código. Essas tarefas são totalmente integrada no ciclo de vida do build. Se você manter a mesma convenção de projetos `Maven`, ele automaticamente vai saber onde está o código, com isso há um esforço mínimo de configuração de sua parte. Na verdade, o seu build terá somente uma linha.

Gestão de dependências robusta

`Gradle` fornece uma infraestrutura para gerenciar a forma como é resolvida, recuperada, e armazenada as dependências de um projeto. Uma vez que foram baixadas e colocadas em seu cache local, elas são disponibilizadas para o seu projeto. Um requisito fundamental para os builds é a reprodutibilidade. Você já ouviu alguma vez o seu colega de trabalho dizer: "**Mas funciona na minha maquina"**? Os builds tem que produzir o mesmo resultado em diferentes ambientes, independente do conteúdo de seu cache local. Gerenciadores de dependência como `Ivy` e `Maven` ainda não podem garantir totalmente a reprodutibilidade do build. Por que isso acontece? Sempre que uma dependência é baixada e armazenado no cache local, ela não leva em conta a origem do artefato. Em situações em que o repositório de código é modificado para um projeto sem a alteração da sua versão, a dependência em cache é considerada resolvida, mesmo que o conteúdo do artefato seja ligeiramente diferente.

Builds escaláveis

Em algumas empresas, os projetos podem conter centenas de módulos em sua estrutura. Construir e testar pequenas mudanças de código pode consumir muito tempo. Devido a sua clara definição de dependências entre os submodulos do projeto, `Gradle` cuida de fazer o build apenas das partes necessárias.
Para melhorar o desempenho de inicialização, `Gradle` pode ser executado em modo daemon. Invocações subsequentes do build iram fazer a chamada para o processo do `Gradle`, que já vai estar rodando em background.
Como resultado, você vai notar que o build irá executar aproximadamente 25% mais rápido.

Fácil de extender

A maioria das empresas tem uma forma diferente de fazer o build, e uma forma diferente de resolver os mesmos problemas. A maneira mais fácil de implementar lógica personalizada é escrevendo uma task. As tasks podem ser definidas diretamente no script do seu build, sem

dificuldade alguma. Se você sentir que sua task está ficando complexa, você pode criar uma classe pra encapsular o código, tornando a estrutura fácil de entender e sustentável.

Integração com outras ferramentas

Se você está utilizando uma ferramenta como o `Ant`, o `Gradle` não força você a migrar totalmente o seu build, ao invés, ele permite você importar a lógica do seu script `Ant` e reutilizar seus targets. O `Gradle` é 100% compatível com os repositórios `Maven` e `Ivy`.

Orientado pela comunidade e suporte corporativo

`Gradle` é uma solução totalmente open source com a Apache License 2.0. Depois do seu primeiro Release em Abril de 2008, a comunidade abraçou o projeto e deu andamento ao seu desenvolvimento. O código está no GitHub.
O `Gradle` tem a **Gradle Inc.** como empresa que cuida diretamente da padronização do `Gradle` e de suas funcionalidades, contratando os principais commiters da comunidade e entregando soluções de build para todo o tipo de empresa com o `Gradle`.

Build Contínuo

Na versão 2.5, foi lançada uma nova funcionalidade ao `Gradle`, o *continuous build*, essa funcionalidade permite iniciar automaticamente um build em resposta a mudanças no código do projeto.
O `Gradle` não irá terminar o processo de build, em vez disso, vai esperar por arquivos que são processados pelo build mudar. Quando for detectada qualquer alteração, vai voltar a executar a compilação anterior com a mesma seleção de tarefa.

Conclusão

Bom, isso é uma breve introdução ao conceito do `Gradle` e suas diferenças com as outras ferramentas de build. Outras ferramentas resolvem a maioria dos problemas, mas são burocráticas e não são escaláveis, `Gradle` veio para resolver os problemas que não são resolvidos por essas ferramentas e ser elegante, ou seja, é uma evolução para as ferramentas de build, que com toda a certeza atende a todas as necessidades que possam aparecer em um processo de build.

TL;DR

De um ponto de vista prático, Gradle lhe dá todo o poder de uma linguagem de programação dinâmica na hora de compilar, testar e distribuir o seu projeto, enquanto Maven força uma abordagem declarativa com passos praticamente pré-definidos.

Como as ferramentas de build ajudam

No dia-a-dia, essas ferramentas vão evitar muito trabalho repetitivo digitando comandos no console para compilar suas classes, executar testes e distribuir seu programa num JAR ou WAR.
Vão lhe poupar ainda de ter que entender e decorar inúmeros comandos e parâmetros ou depender de umma IDE para fazer o trabalho por você.
Também ajudam a padronizar os seus projetos usando convenções de nomes e diretórios. Isso facilita que a cada novo projeto do qual vai participar, você consiga encontrar as coisas mais facilmente.

Maven

Em geral, Maven é mais fácil de entender e de usar em projetos relativamente simples.
Porém ele acaba não sendo tão interessante em projetos complexos onde há cenários não tão comuns. Por exemplo, se em algum momento dos testes você precisar executar determinadas operações específicas para o seu sistema, tal como restaurar um backup no banco de dados, então vai ter que sair procurando algum plugin ou criar um novo.

Gradle

Gradle é mais flexível neste aspecto, sendo útil em empresas com um ciclo de desenvolvimento complexo.
Entretanto, alguns argumentam que introduzir uma linguagem de programação acaba aumentando muito a complexidade dos builds com o tempo, assim como qualquer software, além de ser desnecessário.
O principal argumento contra o Gradle é que o build deve ser mantido simples e declarativo. Se precisar de algo a mais, use *shell script* ou algo do tipo para complexar a tarefa.

Escolhendo

Na prática, a escolha entre Gradle e Maven acaba dependendo muito de preferência pessoal e do contexto do projeto, da mesma forma que funciona a escolha da linguagem de programação.
Se a sua preocupação é saber qual ferramenta você deve aprender, a resposta é *ambas e um pouco mais*.

A esmagadora maioria dos projetos existentes em empresas usa tecnologias antigas. Comumente, você vai encontrar projetos que são exportados diretamente do Eclipse ou Netbeans, projetos compilados usando Ant e Maven. Raramente as empresas desenvolvem projetos com Gradle ou SBT, a não ser se houver iniciativa individual de uma equipe. Claro que isso varia enormemente de empresa para empresa.