Gestão de Configuração e Mudanças de Software

Allan Lima

Arquitetura, Design e Implementação de Sistemas para Internet
Pós Graduação
Faculdade 7 de Setembro

arglbr@gmail.com

nov/2014

Introdução ao Maven

Parte 1

O que é Maven?

- Uma tentativa de definir Maven:
  - Ferramenta para simplificar o processo de build.
- `Build`?
  - Tarefas rotineiras que levam a construção ou montagem de um software a partir do código fonte.
- Tarefas rotineiras?
  - Compilar, executar testes, empacotar aplicação, etc.
- Outra tentativa de definir Maven:
  - Um conjunto de padrões usados para gerenciar e descrever projetos em java.

Quais os benefícios do Maven?

- Modelo que pode ser aplicado aos projetos Java.
- A idéia é que o modelo traga mais transparência, mais reuso, mais facilidade de manutenção e entendimento.
- Fornece uma abstração que a maioria dos desenvolvedores estão familiarizados:
  - Semelhante a abstração do automóvel: se você aprendeu a dirigir em um modelo A de carro, então poderá facilmente dirigir um modelo B.

Quais os benefícios do Maven?

- Abordagem declarativa:
  - POM — Project Object Model.
  - A tarefas são delegadas para o POM e para os plugins.
  - Os desenvolvedores podem usar as tarefas (encapsuladas pelos plugins) sem necessariamente entender como elas funcionam internamente

Princípios do Maven

- CoC - Convenções sobre a configuração
- Reuso de lógicas de *builds*
- Execução declarativa
- Organização coerente de dependências

Princípio 1: CoC - Convenção sobre a configuração

- Estratégia de "propriedades *defaults*" para a maioria das tarefas (podem ser alteradas quando conveniente) economiza tempo.
- Convenções primárias:
  - Estrutura de diretórios padrão para projetos
    - Código fonte, recursos (xml, properties), saída de arquivos gerados, documentação etc
  - Cada projeto gera um único resultado: `jar`, `war`, `ear`...
  - Padrões de nomes
    - Para diretórios: `my-app/src/main/java`
    - Para arquivos gerados (*outputs*): `commons-logging-1.2.jar`

Princípio 2: Reuso de lógicas de *builds*

- Toda a lógica de *build* é encapsulada pelos *plugins*
- Um *plugin* para
  - Para compilar o código fonte
  - Para executar os testes de unidade
  - Para empacotar a aplicação (`jar`, `war`, `ear`)
  - Para gerar `javadocs`
  - etc

Princípio 3: Execução declarativa

- Tudo no *maven* é orientado de forma declarativa no `POM` e nas configurações específicas dos *plugins*. Exemplo do `POM`:

```
1 <project>
2     <modelVersion>4.0.0</modelVersion>
3     <groupId>br.com.curso</groupId>
4     <artifactId>dv-generator</artifactId>
5     <version>0.0.1-SNAPSHOT</version>
6     <name>DV Generador</name>
7     <description>Gerador de digito verificador</description>
8     <dependencies>
9         <dependency>
10            <groupId>junit</groupId>
11            <artifactId>junit</artifactId>
12            <version>4.4</version>
13            <scope>test</scope>
14        </dependency>
15    </dependencies>
16 </project>
```

Princípio 3: Execução declarativa

- Ciclo de Vida de *build*:
  - Consiste de uma série de fases onde cada uma pode executar uma ou mais ações (goals) relacionada a essa fase
  - Uma fase pode incluir outras fases:

A fase `compile` executará:

- `validate`
- `initialize`
- `generate-sources`
- `process-sources`
- `generate-resources`
- `compile`

Princípio 4: Organização de dependências

- Conceitos conectados:
  - Dependências
  - Artefatos
  - Repositórios

```xml
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.curso</groupId>
    <artifactId>dv-generator</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>DV Generador</name>
    <description>Gerador de digito verificador</description>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Princípio 4: Organização de dependências

- Uma dependência é uma **referência** a um artefato específico em um repositório
- Para o `maven` disponibilizar a dependência ele precisa saber em qual repositório procurar usando as **coordenadas** da dependência: `groupId`, `artifactId` e `version`
- `Maven` busca a dependência de um **repositório remoto** e copia para um **repositório local**
- Sempre que uma dependência não se encontrar em um repositório local, `maven` tenta buscar a dependência em repositórios remotos

Repositórios Locais

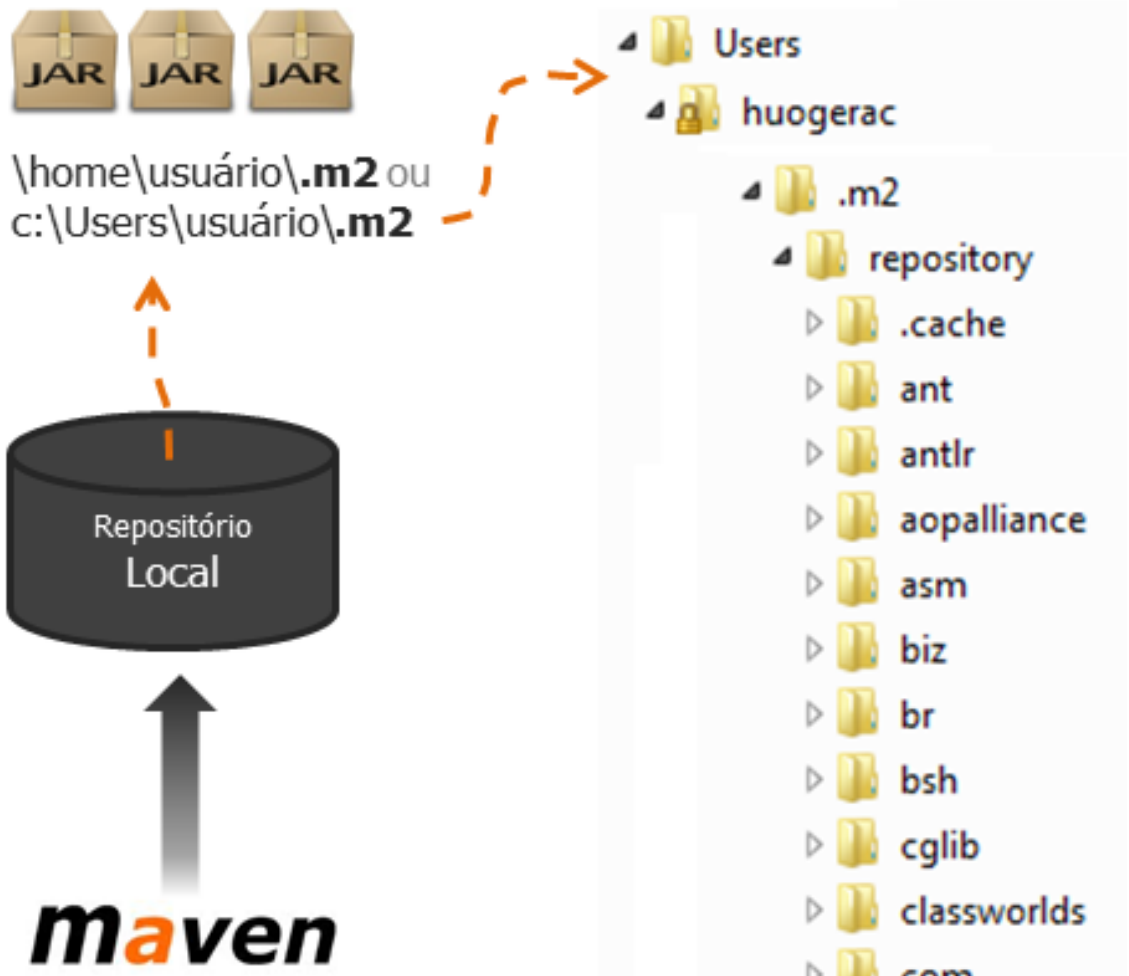

Figure: Fonte: `http://blog.billcode.com.br/2011/09/repositorios-maven-em-5-minutos.html`

Repositórios Internos

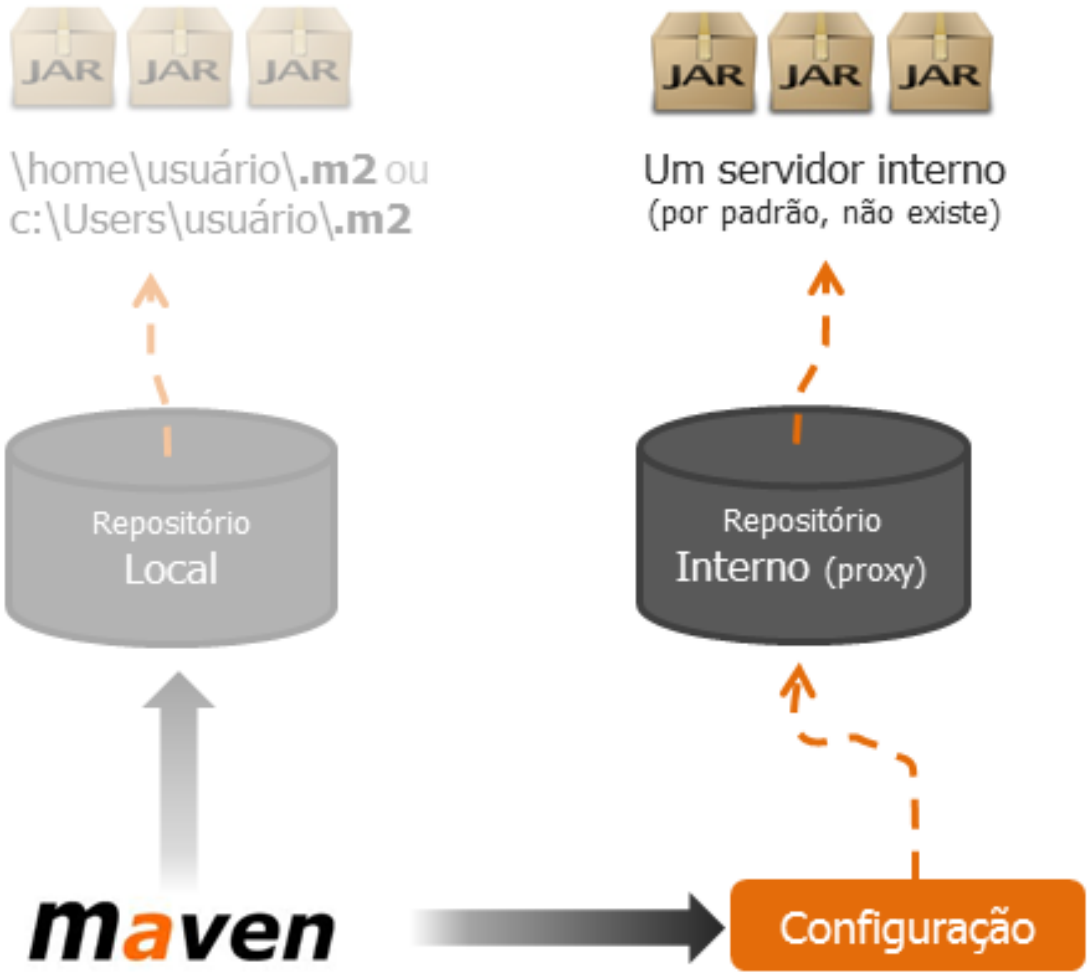

Figure: Fonte: `http://blog.billcode.com.br/2011/09/repositorios-maven-em-5-minutos.html`

Repositórios Remotos

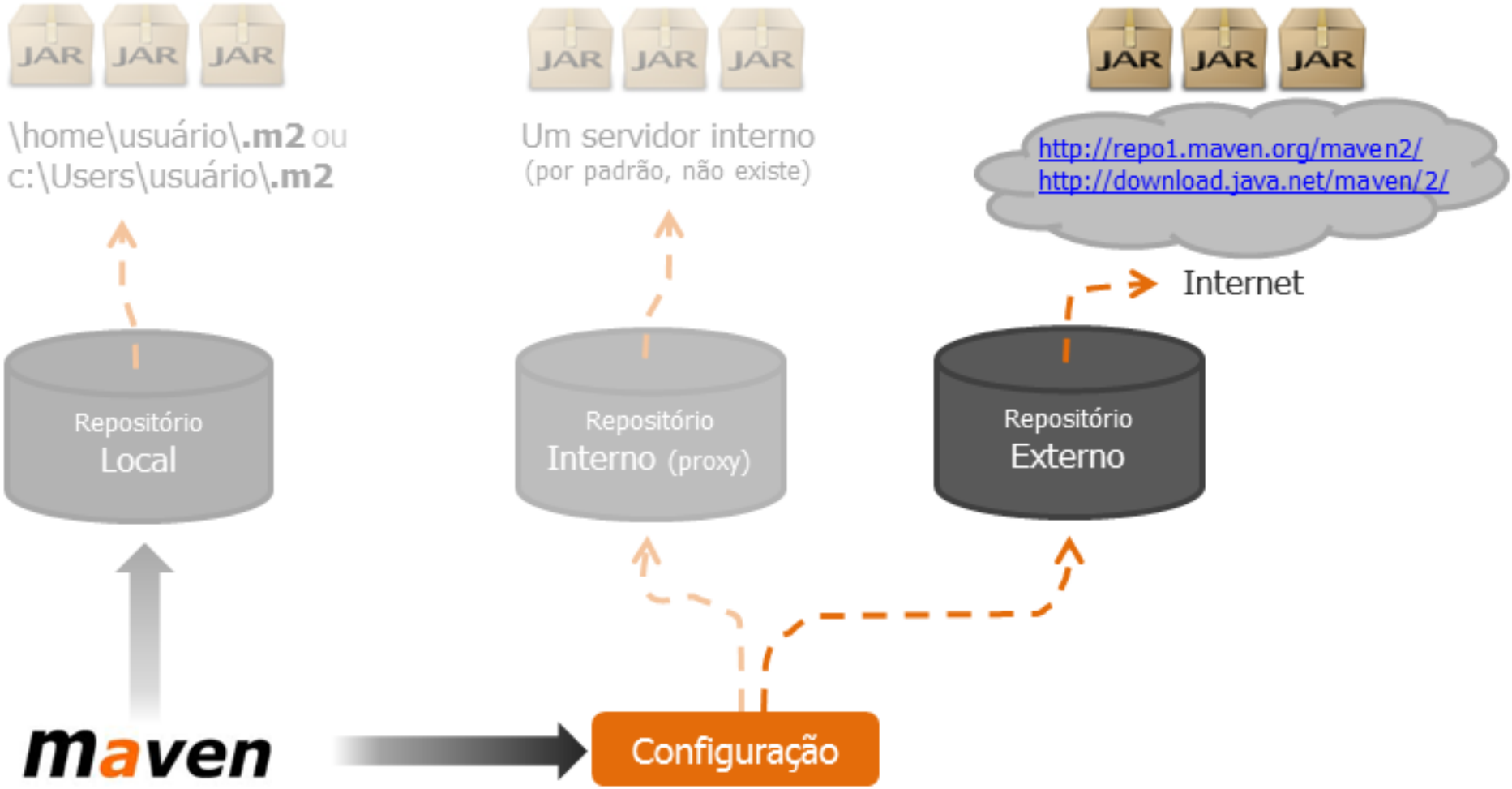

Figure: Fonte: `http://blog.billcode.com.br/2011/09/repositorios-maven-em-5-minutos.html`