PRESENTED BY RED HAT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

JBoss Maven Repository

Paul Gier and John Casey
Red Hat
June 25, 2010

Good News/Bad News

- Good – The JBoss.org Maven Repo Refactoring is Complete
- Bad – No Maven Repository for JBoss Products
- Good – We're working on it!

Presentation Goals

- Learn more about Maven
  - repositories
  - dependency management
- Learn about the JBoss.org Maven repository
- Understand the process used to create JBoss products
- Learn about the upcoming JBoss product Maven repository

Overview

- Maven Repositories and Settings
- The JBoss.org Maven Repository
- Managing Project Dependencies
- Plans for the JBoss Product Maven Repository

What is a Maven repository?

- File server
  - Local file system
  - httpd
- Follows standard layout conventions
  - Directory paths match groupId, artifactId, version
- Release vs. Snapshot repository
  - Releases are never deleted
  - Snapshots are periodically cleaned up

Why Use a Repository?

- Provides a central location for build artifacts
  - Find and re-use artifacts
  - Manage storage from a single place
- Manage dependencies across multiple projects
- Make component integration easier

How is the repository used?

Configuring Maven Repositories

- The Super POM

The Super POM – Central Maven Repository

```
<repositories>
  <repository>
    <id>central</id>
    <name>Maven Repository Switchboard</name>
    <layout>default</layout>
    <url>http://repo1.maven.org/maven2</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
    <id>central</id>
    <name>Maven Plugin Repository</name>
    <url>http://repo1.maven.org/maven2</url>
    <layout>default</layout>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <releases>
      <updatePolicy>never</updatePolicy>
    </releases>
  </pluginRepository>
</pluginRepositories>
```


Repositories in Project POMs

- Add the JBoss.org repository to the build

```
<repositories>
  <repository>
    <id>jboss-public-repository-group</id>
    <name>JBoss Public Maven Repository Group</name>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    <layout>default</layout>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```


Maven Settings

- Global Settings (M2_HOME/conf/settings.xml)
- User Settings (~/.m2/settings.xml)
- Repositories can be added in profiles

```
<profile>
  <id>jboss-public-repository-group</id>
  <repositories>
    <repository>
      <id>jboss-public-repository-group</id>
      <name>JBoss Public Repository Group</name>
      <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
  </repositories>
</profile>
```


Pop Quiz

- You have a project with one repository (Jboss.org) configured in your project POM.
- Using a default Maven install, how many total repositories will Maven download from?

Answer

- ?
- Maven will first add the configured repository and central
- Maven will automatically use repositories found in dependencies POMs
- Good – These repositories don't require configuration
- Bad – Could cause unpredictable build results

Four ways a repository can get into the build

- The super POM
- The project POM(s)
- Dependency POMs
- Maven Settings (settings.xml)

Maven Mirror Settings

- Located in settings.xml
- More control over repositories
- Example mirror configuration:

```
<mirror>
  <id>superfast</id>
  <mirrorOf>cetral</mirrorOf>
  <name>Fast mirror of central</name>
  <url>http://superfast.org/maven2</url>
</mirror>
```


Maven Mirror Settings cont.

- Controlling multiple repositories

```
<mirror>
  <id>jboss-public-repository-group</id>
  <mirrorOf>*</mirrorOf>
  <name>JBoss.org Public Repository Group</name>
  <url>http://repository.jboss.org/nexus/content/groups/public/</url>
</mirror>
```

```
<mirror>
  <id>jboss-public-repository-group</id>
  <mirrorOf>*,!jboss-deprecated</mirrorOf>
  <name>JBoss.org Public Repository Group</name>
  <url>http://repository.jboss.org/nexus/content/groups/public/</url>
</mirror>
```


Overview

- Maven Repositories and Settings
- The JBoss.org Maven Repository
- Managing Project Dependencies
- Plans for the JBoss Product Maven Repository

Rebuilding the JBoss.org Repository

- Old Repository
  - Releases (repository.jboss.org/maven2)
  - Snapshots (snapshots.jboss.org/maven2)
- Problems
  - Deployment over SVN
  - Manually Uploading Thirdparty Jars
  - Manual Search Indexing
  - Bad Artifacts
  - No Staging

Goals for the New JBoss.org Repository

- Sort out the good from the bad
- Split the repository into several parts
  - JBoss.org projects
  - Copied artifacts
  - Bad artifacts
- Improved Deployment/Releases
- Automated Validation/Cleanup
- Use a repository manager

Maven repository managers

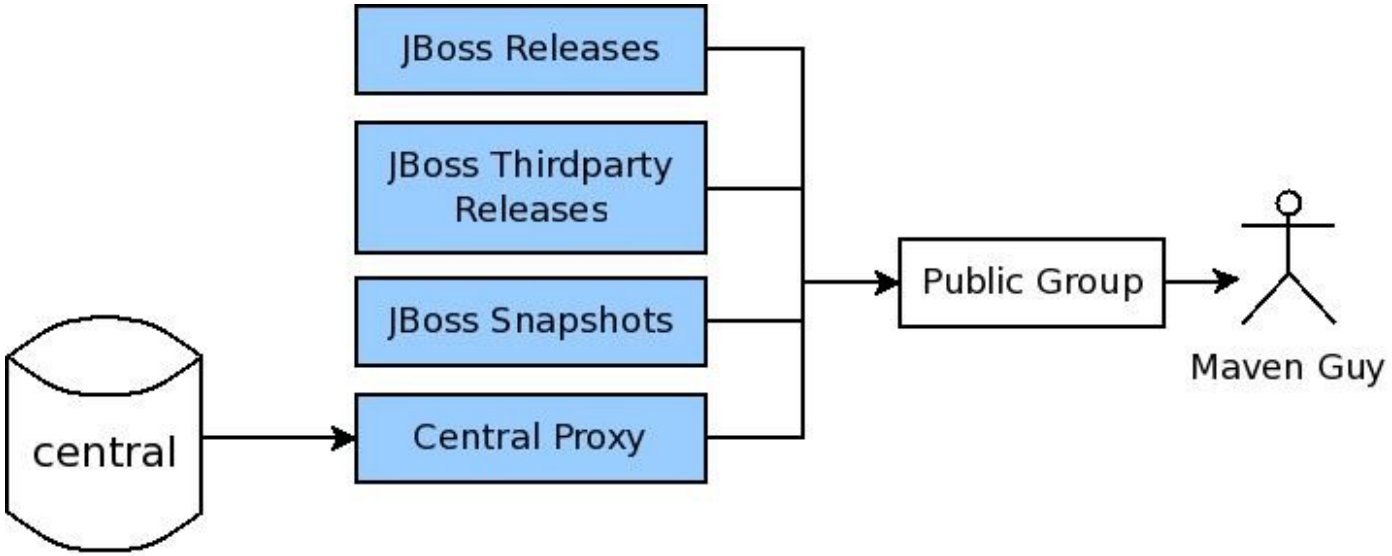
- Features of a Maven repository manager
  - Staging
  - Proxy repositories
  - Repository groups
  - POM validation
- Common repository managers
  - Archiva
  - Artifactory
  - Nexus

Rebuilding the Jboss.org repository

- Sort artifacts into one of several types
  - JBoss Releases
  - JBoss Snapshots
  - JBoss Thirdparty Releases
  - Thirdparty Artifacts copied from another repository
  - Broken artifacts
    - Artifacts that were copied but renamed
    - Artifacts that were changed but not renamed
  - Proxy Repositories (central, java.net, etc.)

Simplify Configuration Using Groups

- Repository Groups provides a unified view of several repositories



Speed Bumps

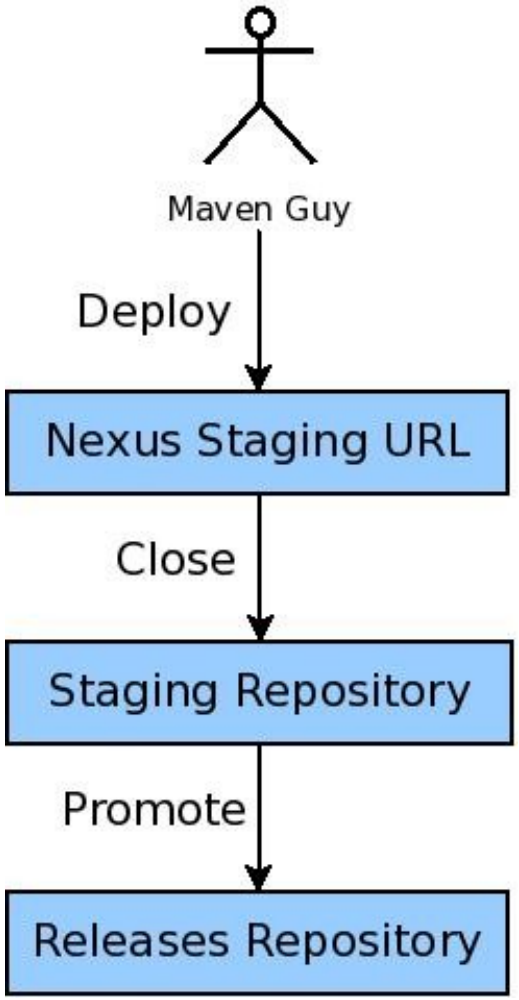
- Complex use of settings.xml
- Performance issues
  - Legacy authentication system
  - https
- All hope is not lost!
  - The new system allows flexibility

Improving the Repository

- Use public group for all development
- Use mixed http/https
- Use repositories in POMs during development
  - Stricter settings.xml during QA and release
- Don't require authentication for downloads

Improved Release Process

- Deploy to staging URL
- Automatic temporary staging repository
- Drop or promote



Lessons Learned

- Repositories in POMs vs. Settings
  - Convenience vs. Reliability
- Repository Managers
  - Provide more power and flexibility
  - Add complexity
- Keep development simple and releases reliable

Overview

- Maven Repositories and Settings
- The JBoss.org Maven Repository
- Managing Project Dependencies
- Plans for the JBoss Product Maven Repository

Maven Dependencies

- Fairly simple...
  - GroupId, artifactId, version and forget it...right?
- So, why worry?
  - Incorrect dependencies == painful to use!
  - One of the few materials exported by your project
- What do I need to know?
  - Dependency Scoping
  - Inheritance and Reuse
  - Conflicts and Exclusions

What are You Producing? Classifiers

- 1:1 POM:artifact is the rule for most cases
- Classifiers used for artifacts *derived* from main output
- “attached” to main POM / artifact pair for install / deploy
- Uses <classifier/> element when used as dependency
- Transitive dependencies may not work correctly
- Examples: javadocs, source jars, distro archives

What are You Producing? Types

- Roughly equivalent to file extensions
- Loosely related to POM <packaging/> element
  - Multiple <packaging/> types may produce “jar” artifacts
- Determines dependency-handling rules
  - Example: transitive deps not resolved for “war” type
- Custom type definitions injected via build extensions
- Uses <type/> element when used as dependency

How are You Using It? Dependency Scoping

- Many dependencies required for compiling, running
  - `<scope>compile</scope>` (default, can be left off)
- Some dependencies are for testing
  - `<scope>test</scope>`
- Others required only to run the application
  - `<scope>runtime</scope>`
- Some are assumed to be provided by the platform
  - `<scope>provided</scope>`
  - Example: javamail API

Dependency Scoping (cont'd.)

- Scopes make dependencies available to certain parts of the build process
  - Example: “test” scope available for unit testing
- Scopes can imply other scopes
- Runtime scope used to resolve transitive deps

Dependency Scoping: Example

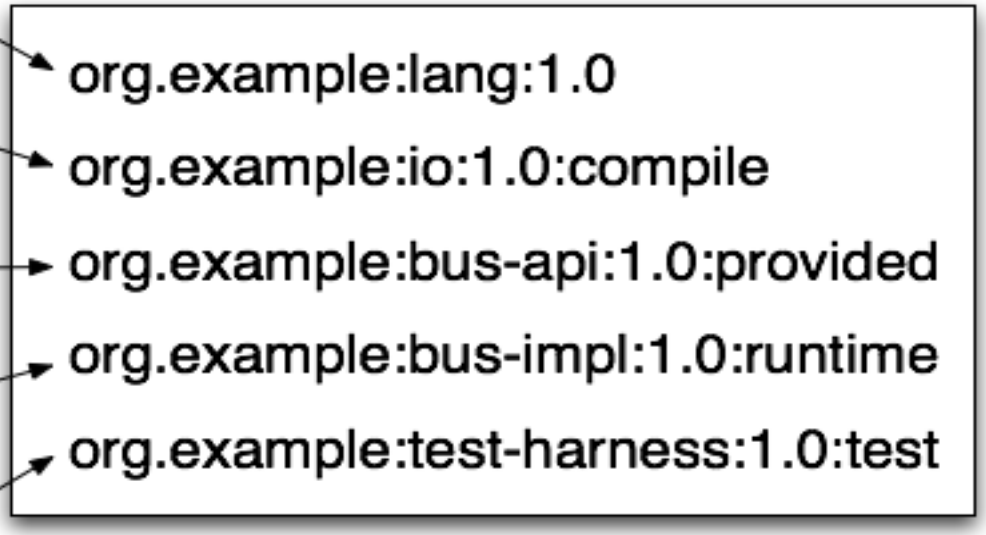
```
<dependencies>
  <dependency>
    <groupId>org.example</groupId>
    <artifactId>lang</artifactId>
    <version>1.0</version>
  </dependency>

  <dependency>
    <groupId>org.example</groupId>
    <artifactId>io</artifactId>
    <version>1.0</version>
    <scope>compile</scope>
  </dependency>

  <dependency>
    <groupId>org.example</groupId>
    <artifactId>bus-api</artifactId>
    <version>1.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>org.example</groupId>
    <artifactId>test-harness</artifactId>
    <version>1.0</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.example</groupId>
    <artifactId>bus-impl</artifactId>
    <version>1.0</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```



Dependency Scoping: Compile

- Default is “compile”
- Platform APIs required to compile code
- Dynamically loaded classes NOT required
- Test code is separate, so test dependencies not required

```
org.example:lang:1.0
org.example:io:1.0:compile
org.example:bus-api:1.0:provided
org.example:bus-impl:1.0:runtime
org.example:test-harness:1.0:test
```


Dependency Scoping: Test

- Tests need access to ALL dependencies
- Platform APIs, runtime deps may be required in testing environment

```
org.example:lang:1.0
org.example:io:1.0:compile
org.example:bus-api:1.0:provided
org.example:bus-impl:1.0:runtime
org.example:test-harness:1.0:test
```


Dependency Scoping: Runtime

- Runtime scope == artifacts to be distributed with application
- Platform APIs excluded (provided scope)
- Used during transitive resolution

org.example:lang:1.0

org.example:io:1.0:compile

org.example:bus-api:1.0:provided

org.example:bus-impl:1.0:runtime

org.example:test-harness:1.0:test

Review: Transitive Dependencies

application/pom.xml:

```
<dependencies>
   <dependency>
      <groupId>org.example</groupId>
      <artifactId>io</artifactId>
      <version>1.0</version>
   </dependency>
</dependencies>
```

io/pom.xml:

```
<dependencies>
   <dependency>
      <groupId>org.example</groupId>
      <artifactId>lang</artifactId>
      <version>1.0</version>
   </dependency>
</dependencies>
```

mvn dependency:tree

```
...
[INFO] org.example:application:jar:1.0
[INFO] +- org.example:io:jar:1.0:compile
[INFO] |  \- org.example:lang:jar:1.0:compile
```

direct dependency

transitive dependency

Transitivity and Dependency Scoping

application/pom.xml:

org.example:bus-api:1.0:compile

io/pom.xml:

org.example:lang:1.0:compile
org.example:reflect:1.0:runtime

bus-api/pom.xml:

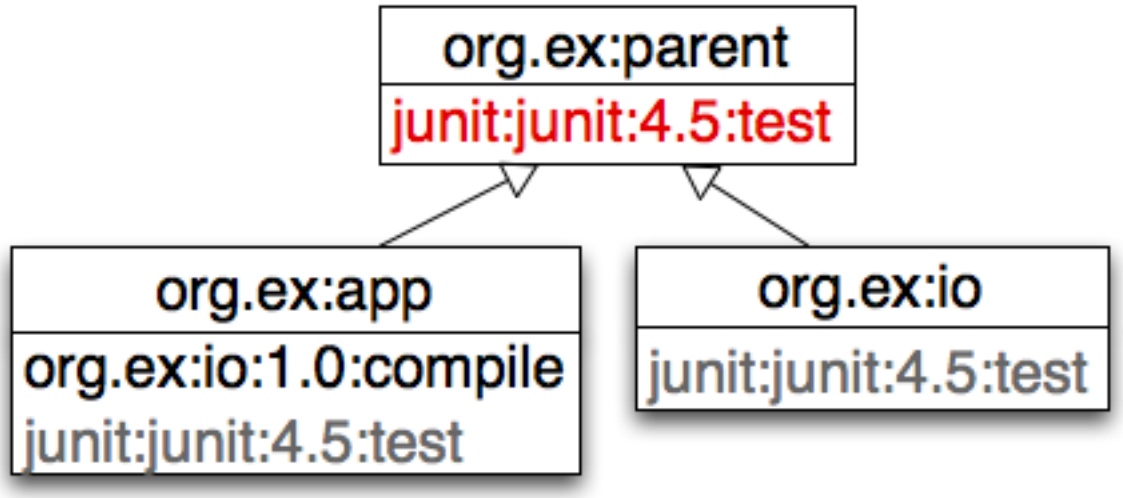
org.example:io:1.0:compile
org.example:mailer:1.0:provided

mvn dependency:tree

```
...
[INFO] org.example:application:jar:1.0
[INFO] +- org.example:bus-api:jar:1.0:compile
[INFO]    \- org.example:io:jar:1.0:compile
[INFO]       +- org.example:lang:jar:1.0:compile
[INFO]       \- org.example:reflect:jar:1.0:runtime
```


Common Dependencies

- How can we reuse common dependencies?
- Parent POMs are ideal for consolidating dependencies
- If all children need a dependency, simply declare in the parent:



Common Dependencies: DependencyManagement

- What if only SOME children use a dependency?
- Declare in <dependencyManagement/>, and reference in child POMs:

Parent:

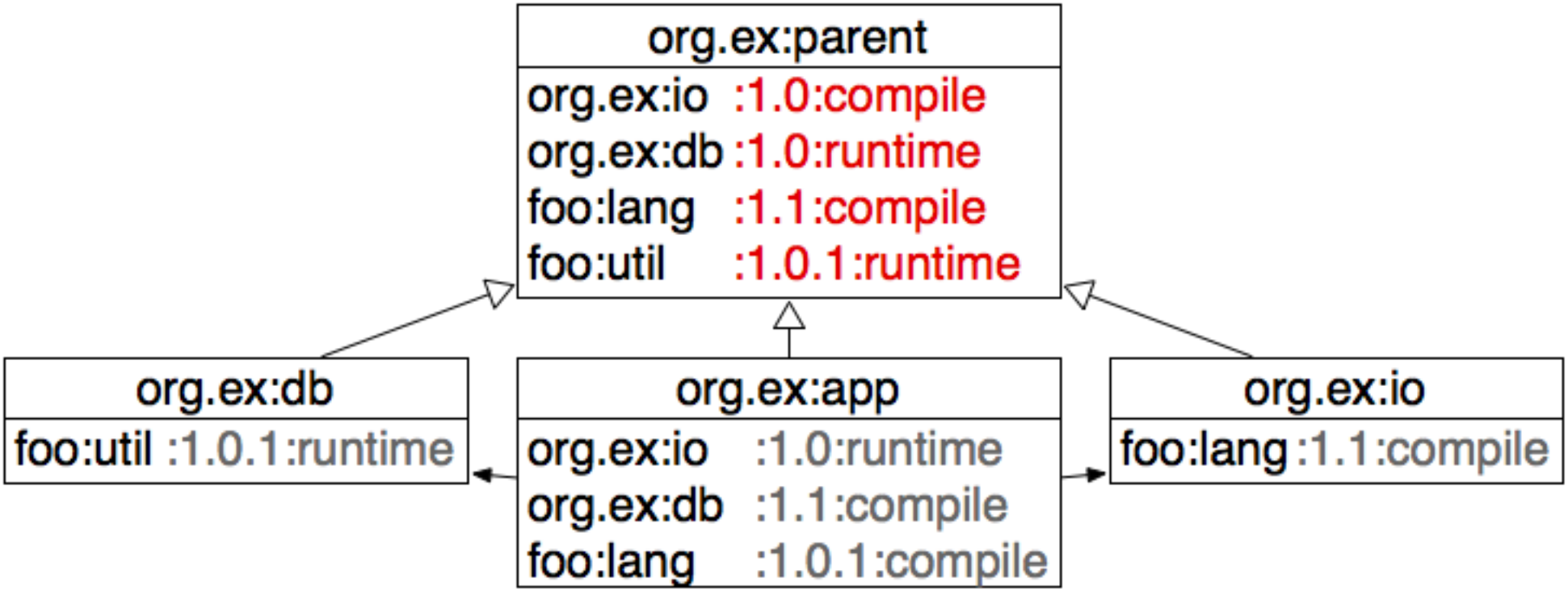
```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>foo</groupId>
      <artifactId>util</artifactId>
      <version>1.1</version>
      <scope>runtime</scope>
  </dependencies>
</dependencyManagement>
```

Child:

```
<dependencies>
  <dependency>
    <groupId>foo</groupId>
    <artifactId>util</artifactId>
  </dependency>
</dependencies>
```

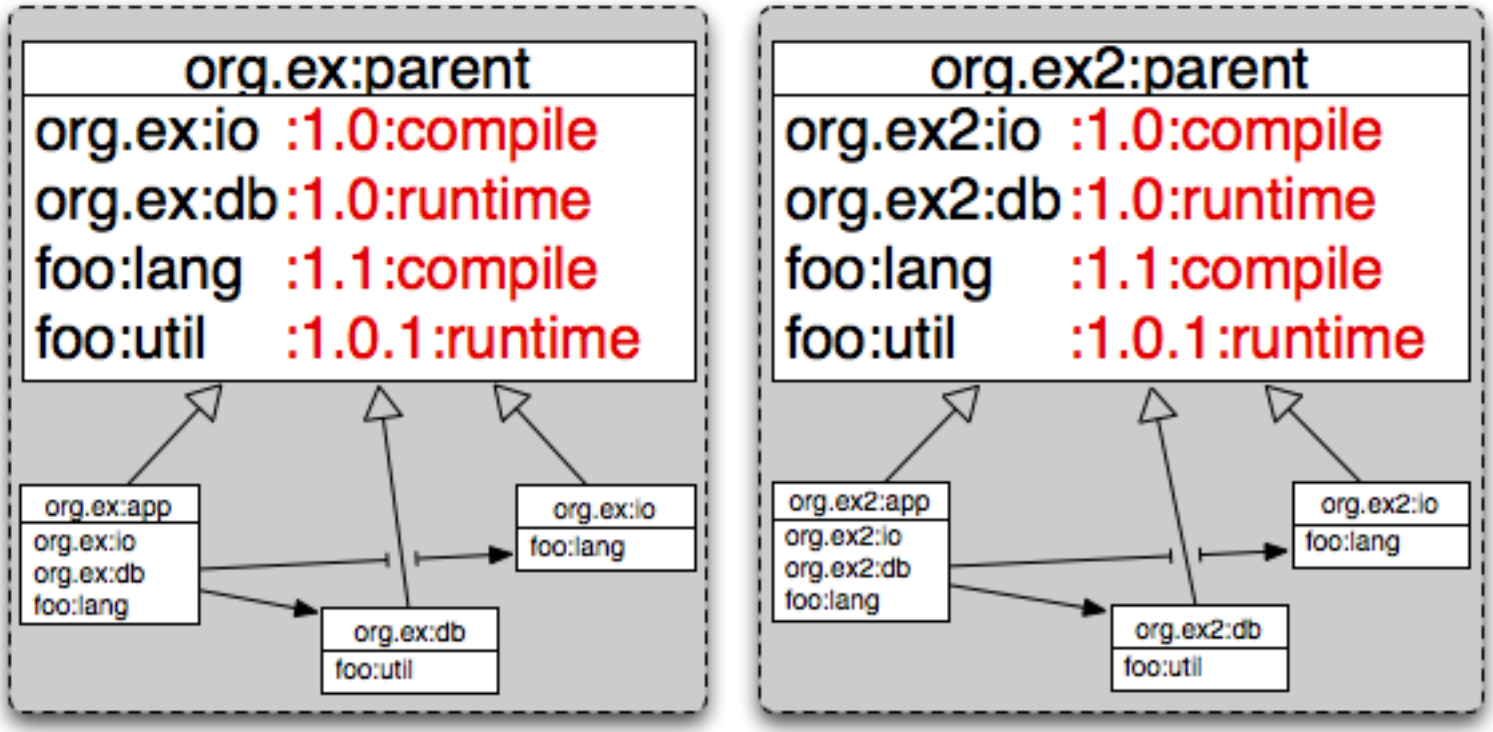

Common Deps: DependencyManagement (cont'd).

- Reference managed dependencies as needed in children:



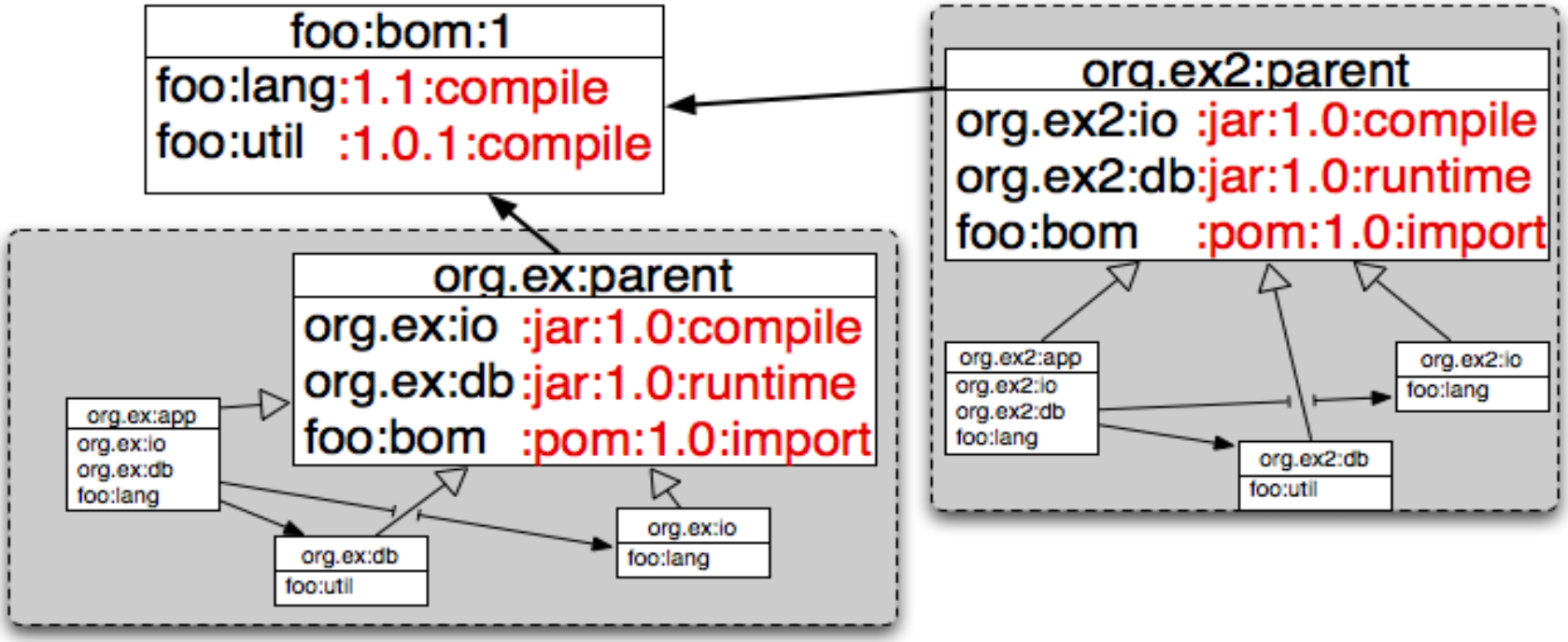
Sharing Managed Dependencies

- What if more than one application needed the same groups of dependencies?



Sharing Managed Dependencies: The BOM

- Bill-of-Materials POMs (BOMs)
- Shared <dependencyManagement/> section in a POM



Sharing Managed Dependencies: The BOM

- BOM <dependencyManagement/> copied to referencing POM
- Reference using “import” scope, “pom” type:

BOM:

```
<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>foo</groupId>
   <artifactId>util</artifactId>
   <version>1.1</version>
   <scope>runtime</scope>
 </dependencies>
</dependencyManagement>
```

Referencing POM:

```
<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>foo</groupId>
   <artifactId>bom</artifactId>
   <version>1.1</version>
   <type>pom</type>
   <scope>import</scope>
 </dependencies>
</dependencyManagement>
```


When Things Go Wrong

- Polluters living upstream. Example:
  - One of the projects you use in your application declares a dependency on Junit with “compile” scope
- Version conflicts. Example:
  - Your application uses version 1.2 of a logging library, and version 1.1 of some client library
  - The client library uses version 1.0 of the same logging library

application/pom.xml:

```
org.ex:logging:1.2
org.ex:client:1.1
```

client/pom.xml:

```
org.ex:logging:1.0
```

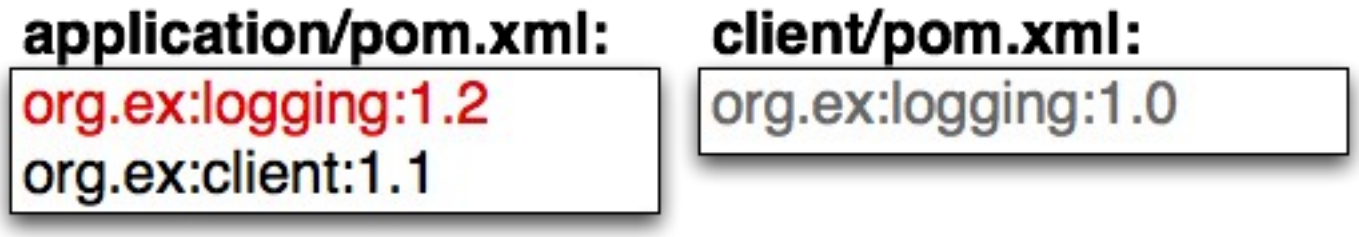

When Things Go Wrong: Dependency Pollution

- Add an exclusion to ban Junit coming from that dependency:

```
<dependency>
  <groupId>org.ex</groupId>
  <artifactId>io</artifactId>
  <version>1.0</version>
  <exclusions>
    <exclusion>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```


When Things Go Wrong: Version Conflicts

- Maven will automatically prefer the closest declaration:



Overview

- Maven Repositories and Settings
- The JBoss.org Maven Repository
- Managing Project Dependencies
- Plans for the JBoss Product Maven Repository

Putting these Principles to Use...

- Plan is to publish JBoss product artifacts as a Maven repository
  - Allow developers to build against commercial versions of artifacts
  - Harness product build process to populate
- Build in the advantages of JBoss products

Building a JBoss Product: Goals

- Audit Trail
  - Certification of build process and resulting product
  - Preserve unbroken chain of custody from source code to running software
- Ownership of Code
  - If a bug is reported, we can fix it and rebuild

Building a JBoss Product: Certifying Results

- Preserve information about every step from source to binary
  - Inputs, Outputs, Logs
- Secure all steps and machinery used
  - Build-system interconnections authenticated
  - Network traffic restricted
  - Storage secured
- Build output (RPMs, etc.) signed
- Yum connections signed / secured

Building a JBoss Product: Tooling

- Modified version of Koji
  - Isolated, cleanroom build environment
  - Each environment contains EXACT build requirements for that project
  - Tracks build input, output, logs
  - Generates RPM and related Maven repository fragment
- Wraps existing builds provided by project where possible

Product Repository: Goals

- Automatically publish EXACT copy of any artifact included in a product
- Maintain compatibility with community projects and repository
- Minimize pain for users switching over from community projects and repository

Product Repository: Coordinate Design

- Preserve groupId's, artifactId's
  - Changing these would require exclusions to ban corresponding community artifacts
- Coordinate must be different if the artifact is rebuilt
  - Checksum, signatures will be different
- Provide visual clue to differentiate product artifacts from community ones
  - Add “-redhat-#” to the end of artifact versions
  - “#” signifies the rebuild index of that artifact

Product Repository: Usage Design

- Rely on Bill-of-Materials (BOM) POMs to group related JBoss artifacts
  - BOMs available for both community projects and product
  - Users reference BOM to “import” JBoss artifact versions
  - Change BOM version to switch from community to commercial artifacts
- REQUIRES MAVEN 3

Product Repository: Process

- Aggregate Koji build output
  - Remember those repository fragments?
- Indexed for searching by tools such as Nexus
- Hosted with UI to make consumption easier

Product Repository: Ease of Use

- Documentation
  - Searching for dependencies
  - Configuring Maven for the product repository
- Maven archetypes using product repository
- Integration with JBoss Tools (JBoss Developer Studio)
- Painless switchover from community to product artifacts
  - Switch to product repository URL
  - Switch to corresponding product BOM

Help Us Test!

- We're currently working on a prototype product repository
- Sign up to be part of the beta testing
- Questions?

FOLLOW US ON TWITTER

www.twitter.com/redhatsummit

TWEET ABOUT IT

#summitjbw

READ THE BLOG

http://summitblog.redhat.com/