

Maven 3.x: The Evolution of Enterprise Java Build Infrastructures

The Maven-related tooling you'll be using in your infrastructure for years to come

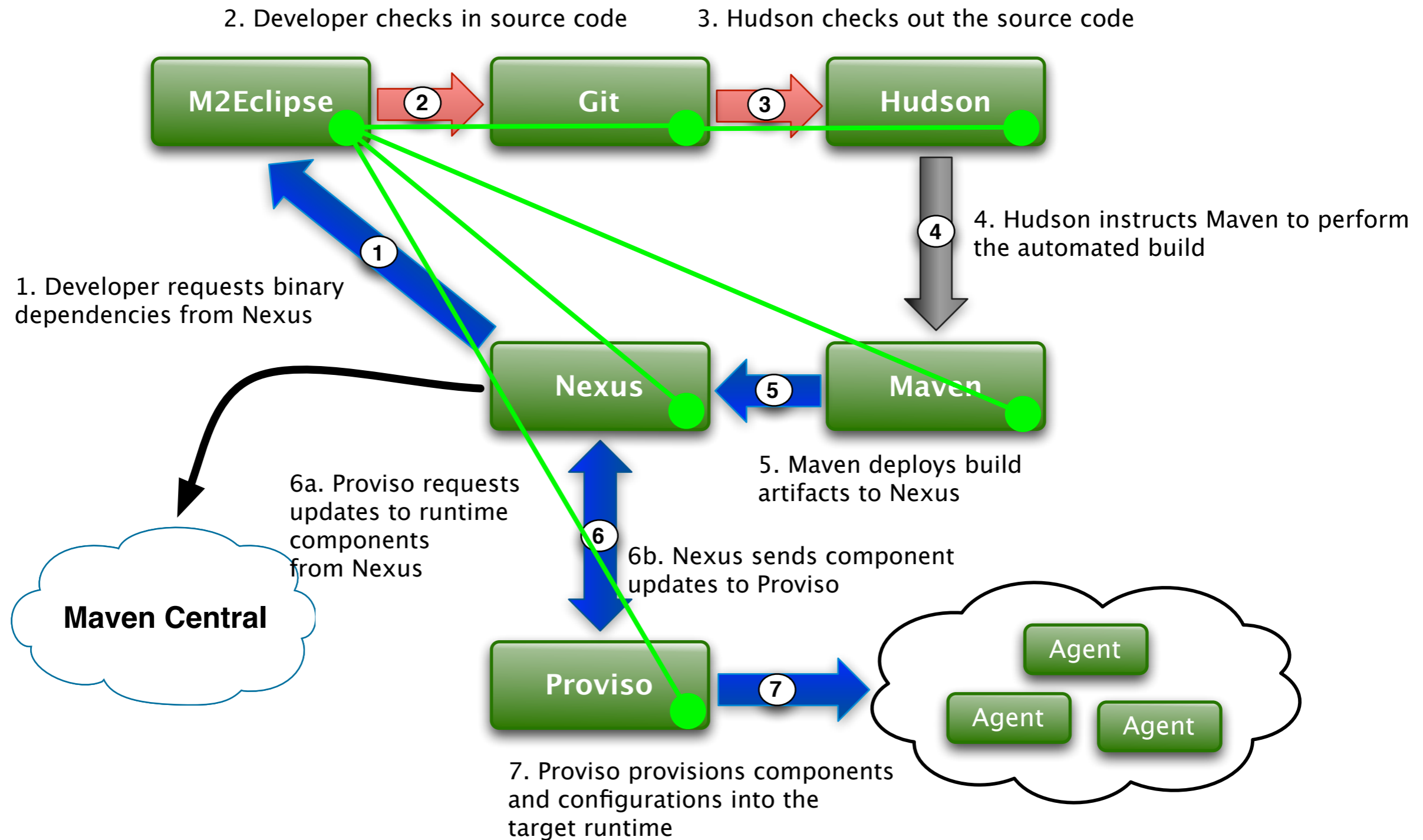
Jason van Zyl

<http://twitter.com/jvanzyl>



Agenda & Session Goals

What we're going to talk about and accomplish this session



Agenda & Session Goals

What we're going to talk about and accomplish this session

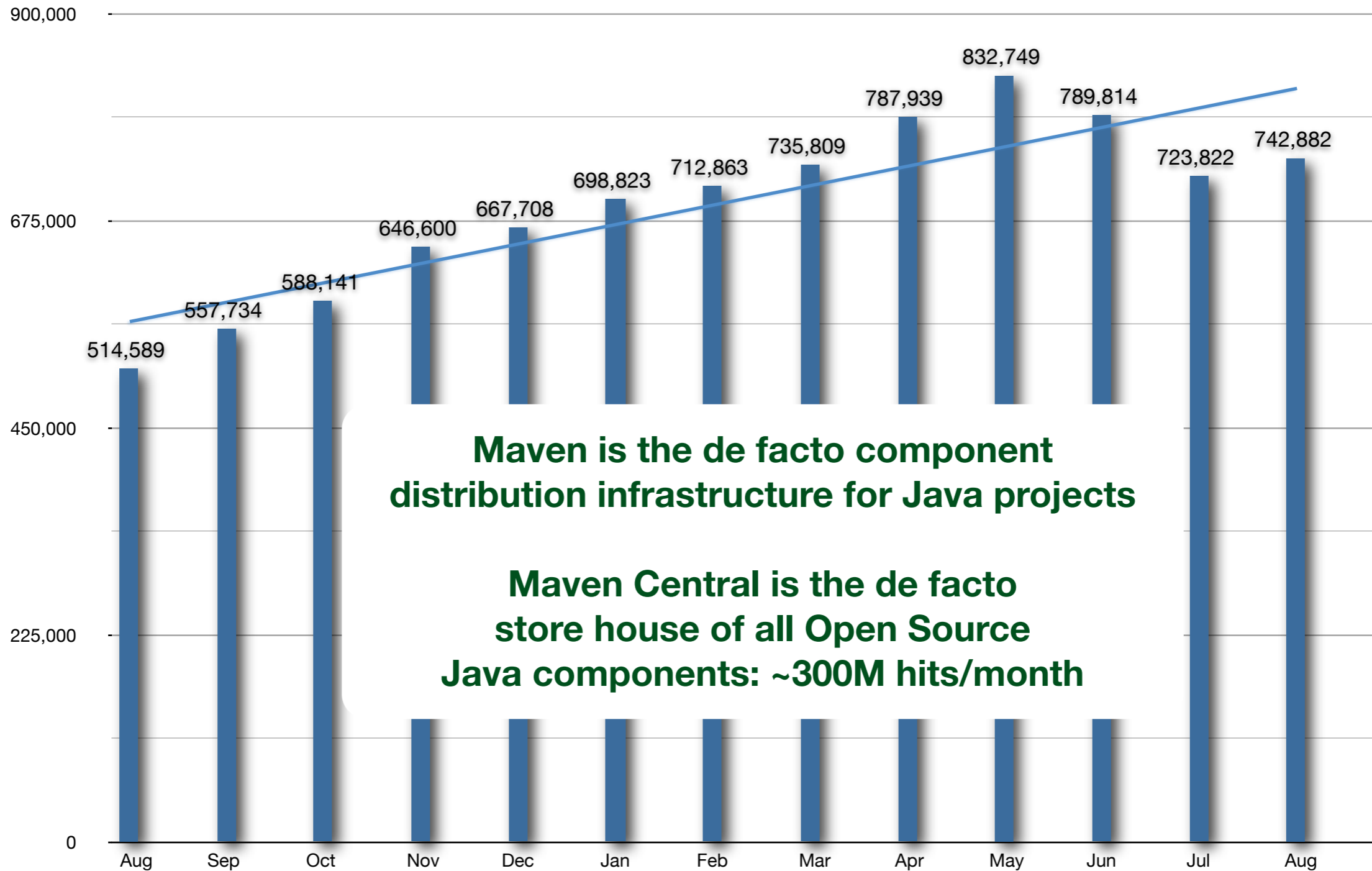
To understand how Maven 3.x and its related tools & frameworks are going to help us get to that ideal delivery infrastructure

Updates on Maven, M2Eclipse & Tycho ...

What's going on at Sonatype & in Maven land?

- Maven 3.0-final has been released. Yay!
- Sonatype is planning features for what we would like to see in Maven 3.1
- M2Eclipse 1.0 will be released after two 4-6 week iterations or so.
 - It's getting better rapidly because most of Sonatype's Eclipse team is working on it.
 - We are moving M2Eclipse to the Eclipse Foundation
 - We've started checking in the code at the Eclipse Foundation
- Tycho passed its project creation review at the Eclipse Foundation, has been provisioned, and will soon begin the parallel IP process
- MavenShell 1.0 will be released December 1st
- PolyglotMaven 1.0 will be released January 1st
- PDE replacement using Tycho hopefully demo-ready in a couple weeks
- We hope to submit our JSR-330 and REST work to the Hudson project soon & possibly working with Oracle to implement Maven 3.x support

Maven Central Unique IPs / Month [Aug2009 to Aug2010]



Maven is the de facto component distribution infrastructure for Java projects

Maven Central is the de facto store house of all Open Source Java components: ~300M hits/month

2008 Total Unique IPs: 1,836,709

2009 Total Unique IPs: 3,978,964

2010 Total Unique IPs: 6,024,701 (end of Aug)

non-Maven < 0.5%
RM < 7%

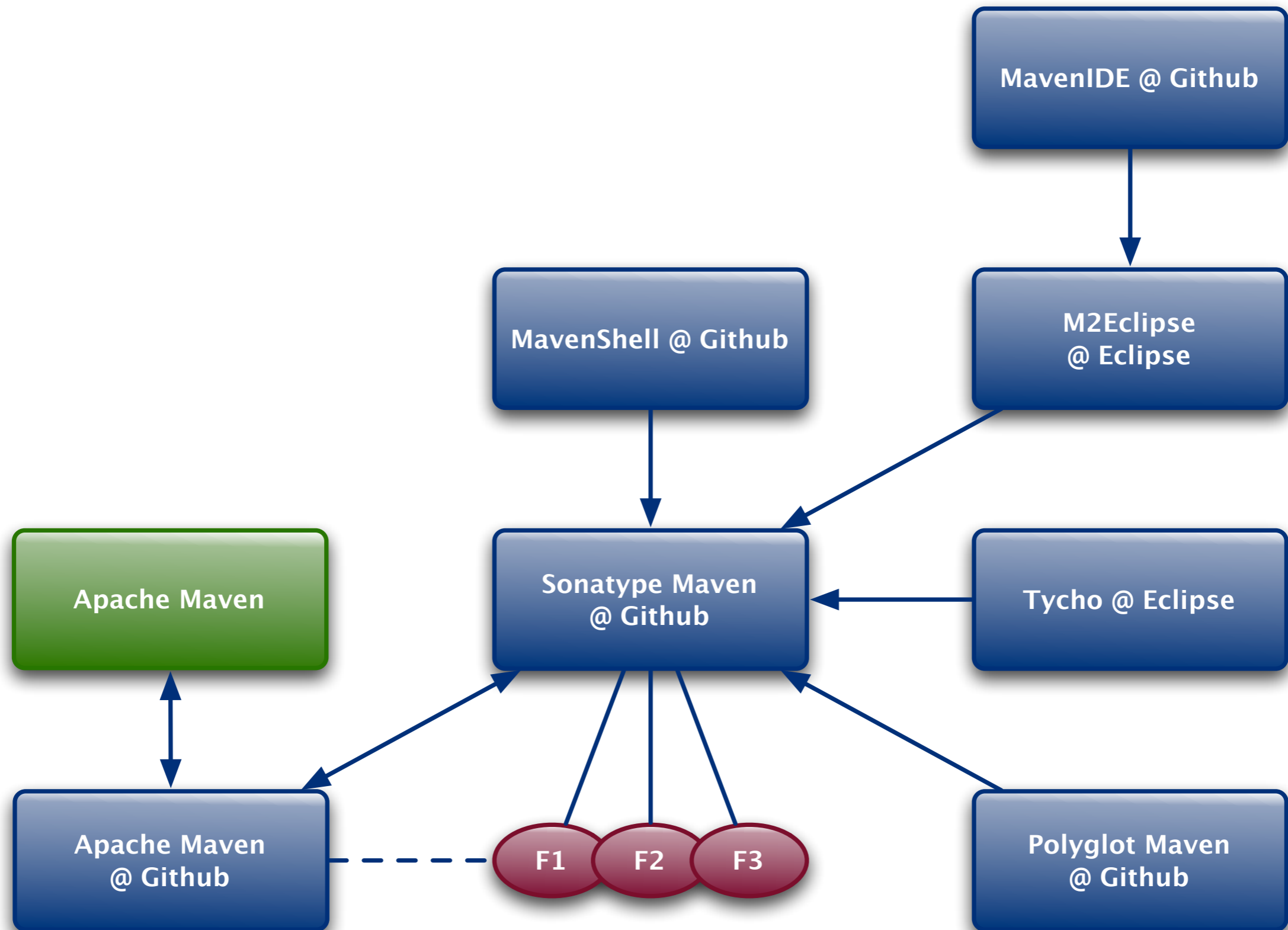
Maven 3.1

What's going on for Maven 3.1

- POM interoperability
- Maven 3.1 Plugin API (SM --> AM)
 - JSR-330
 - Java5 annotations
 - Tight integration with M2Eclipse
 - Build avoidance
- Mixins (SM --> AM)
- Versionless parents in child POMs (SM --> AM)
- **Concurrent-access local repository implementation (SM)**
- **Async HTTP Client connector for Aether (SM)**
- **Async HTTP Client Wagon (SM)**
- **Dynamic Extension API (SM)**
- **Eventing API (SM)**

Sonatype's Maven Plan

Rapidly innovate while being reconciled to the Apache Way



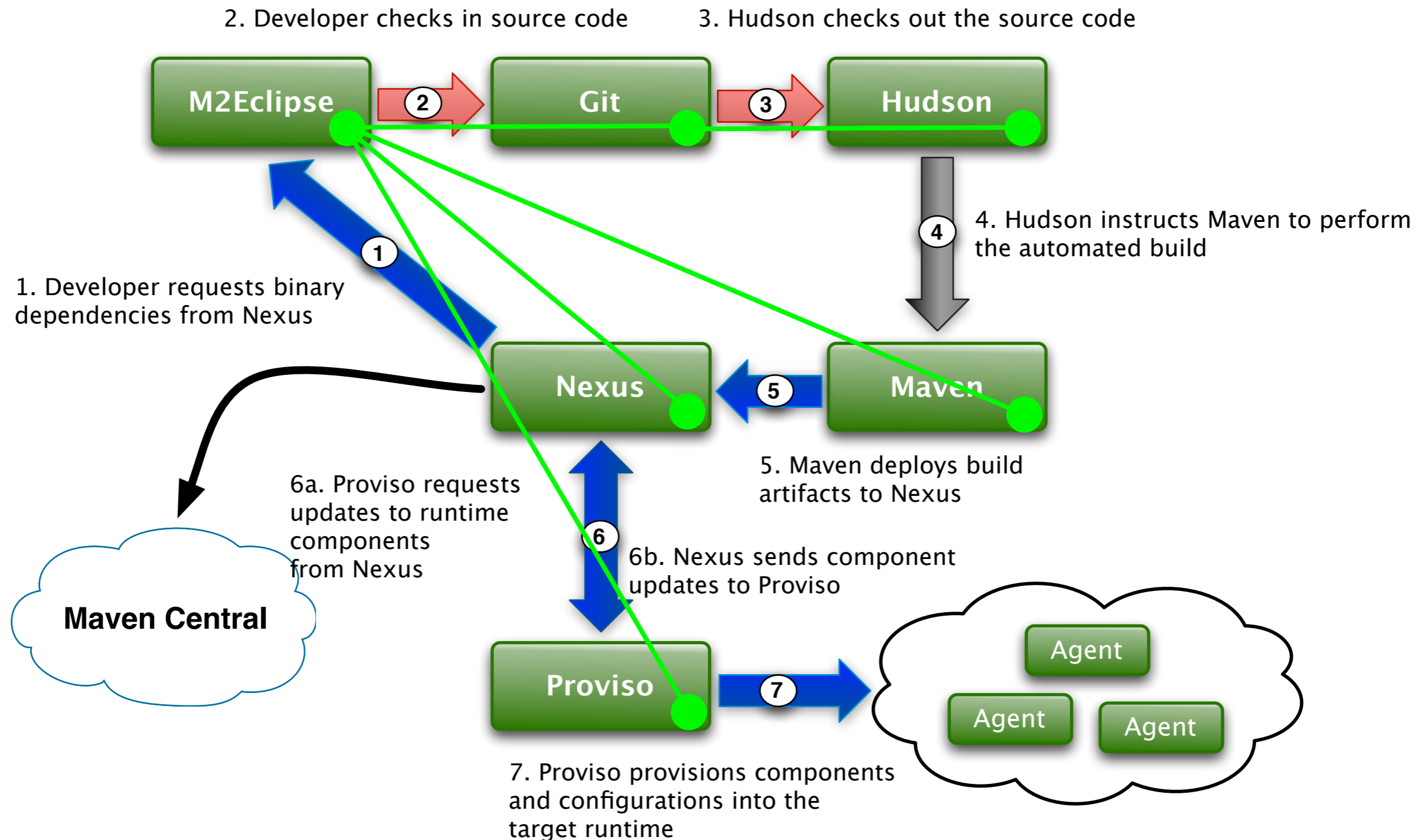
A Maven-focused Hudson

Sonatype is doing a lot of work with Hudson

- **JSR-330-based plugins**
- **REST support using Apache Wink**
- **Global configuration management**
- Apache Shiro integration for security management and SSO
- Slave tool provisioning with Proviso (based on Eclipse p2)
- **Integration with M2Eclipse (Sonatype Professional)**
- Maven 3.x integration (in progress)
- Long-lived workflow using Drools Flow (research)

The Idea Development Infrastructure

Now let's talk about some of the underlying technologies



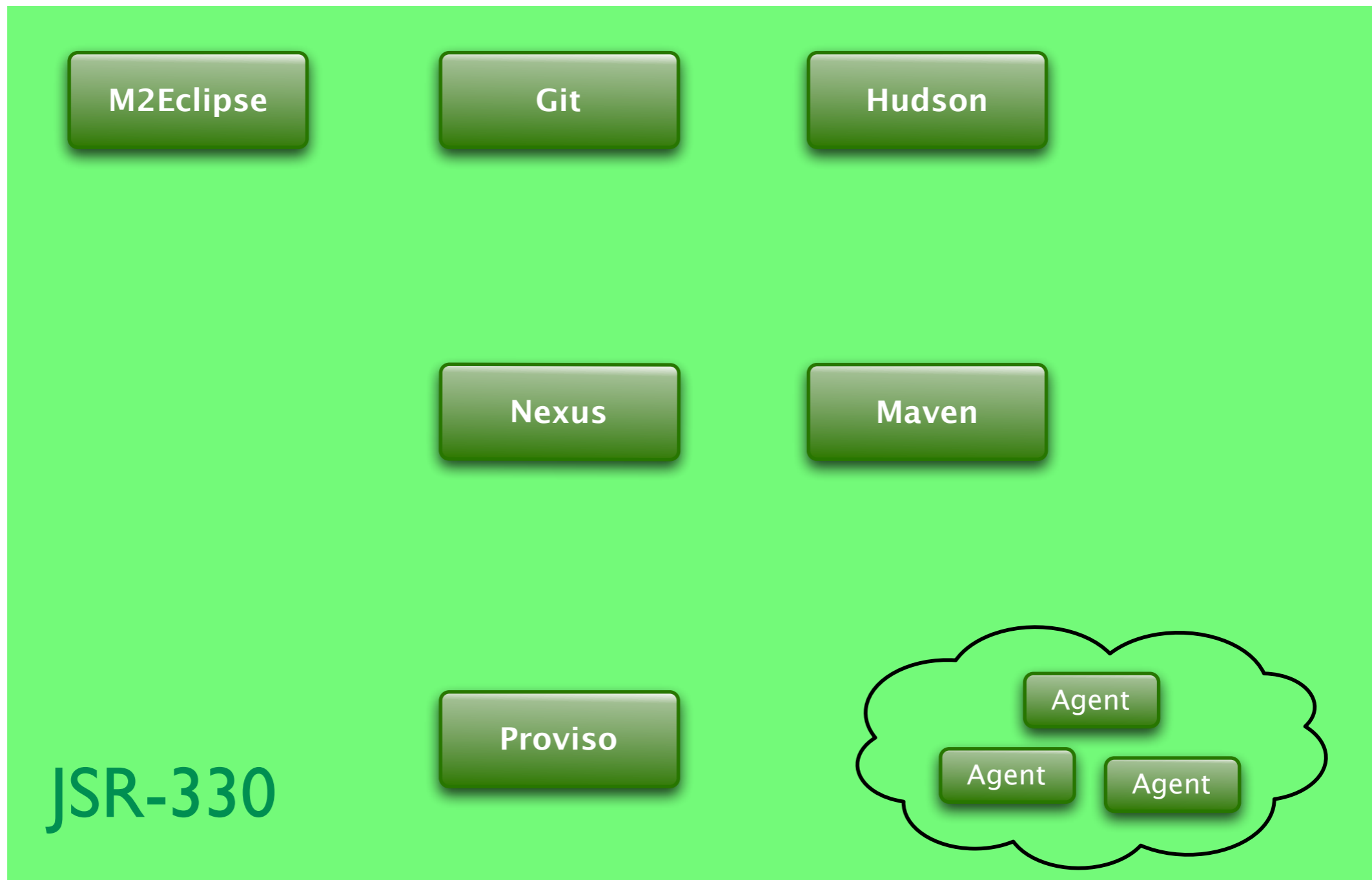
The ideal Maven-focused delivery infrastructure

What does that look like?

- Shared component & plugin model
- Shared transport system
- Shared repository API
- Enriched component metadata
- Enhanced IDE connectivity to the infrastructure
- Provisioning (I think this might have to wait until the next webinar)

Shared component model

For Sonatype this means JSR-330



Moving from Plexus to Guice & JSR-330

Making it all work with Guice

- Requirements
 - Absolutely no code changes for any Maven, Nexus, M2Eclipse component
 - Must support Plexus' classpath/resource scanning
 - Must support Plexus' dynamic component assembly based on discovered metadata
 - Must support Plexus' configuration & converter mechanism
 - When we need changes made to the runtime container, we need those changes to be timely
 - Support for arbitrary lifecycles
 - We need the container to be wed with OSGi -- for us the answer is Peaberry
 - Component graph proxy support: for components and configuration
 - Dynamic language support

Implications of using JSR-330

We bring some sanity to tooling

- Writing plugins in various ways for tools like Maven, Nexus, Hudson, Sonar & Eclipse has a great deal of mental overhead. This burden will be removed.
- The implications for development, testing and delivery are **huge**. They cannot be understated
 - Common development models: how to create JSR-330-based plugins, better component reuse, a common understand of infrastructure tooling
 - Common testing frameworks for JSR-330 e.g Sonatype's REST/UI toolkit
 - Common provisioning models

Sisu Maven Plugin Example

Using the same component model

```
@Goal( "webxml" )
@Phase( GENERATE_RESOURCES )
@RequiresProject
@Threadsafe
public class GenerateWebXml extends SisuMavenMojo {
    @Inject Logger logger;

    @Inject
    private Component component;

    @Inject @Named( "${project}" )
    private MavenProject project;

    @Inject @Named( "${outputDirectory}" ) @DefaultsTo( "${project.build.directory}" )
    private File outputDirectory;

    @Inject
    private List<WebXmlAugmenter> webXmlAugmenters;

    public void execute() throws Exception {
        component.generate( project, webXmlAugmenters, outputDirectory );
    }
}
```

Sisu Hudson Plugin Example

Using the same component model

```
@Named
@Singleton
public class RestPlugin
    extends Plugin
{
    @Inject
    private Logger logger;

    private transient List<ApiProvider> providers;

    private boolean enabled = true;

    @Inject
    public RestPlugin(final List<ApiProvider> providers) {
        assert providers != null;
        this.providers = providers;

        logger.debug("Providers:");
        for (ApiProvider provider : providers) {
            logger.debug( "    {}", provider );
        }
    }
}
```

Sisu Nexus Plugin Example

Using the same component model

```
@Named
@Singleton
@Path( CapabilitiesResource.RESOURCE_URI )
@Produces( { "application/xml", "application/json" } )
@Consumes( { "application/xml", "application/json" } )
public class CapabilitiesResource
    implements Resource {
    public static final String RESOURCE_URI = "/capabilities";

    private final CapabilityConfiguration capabilitiesConfiguration;

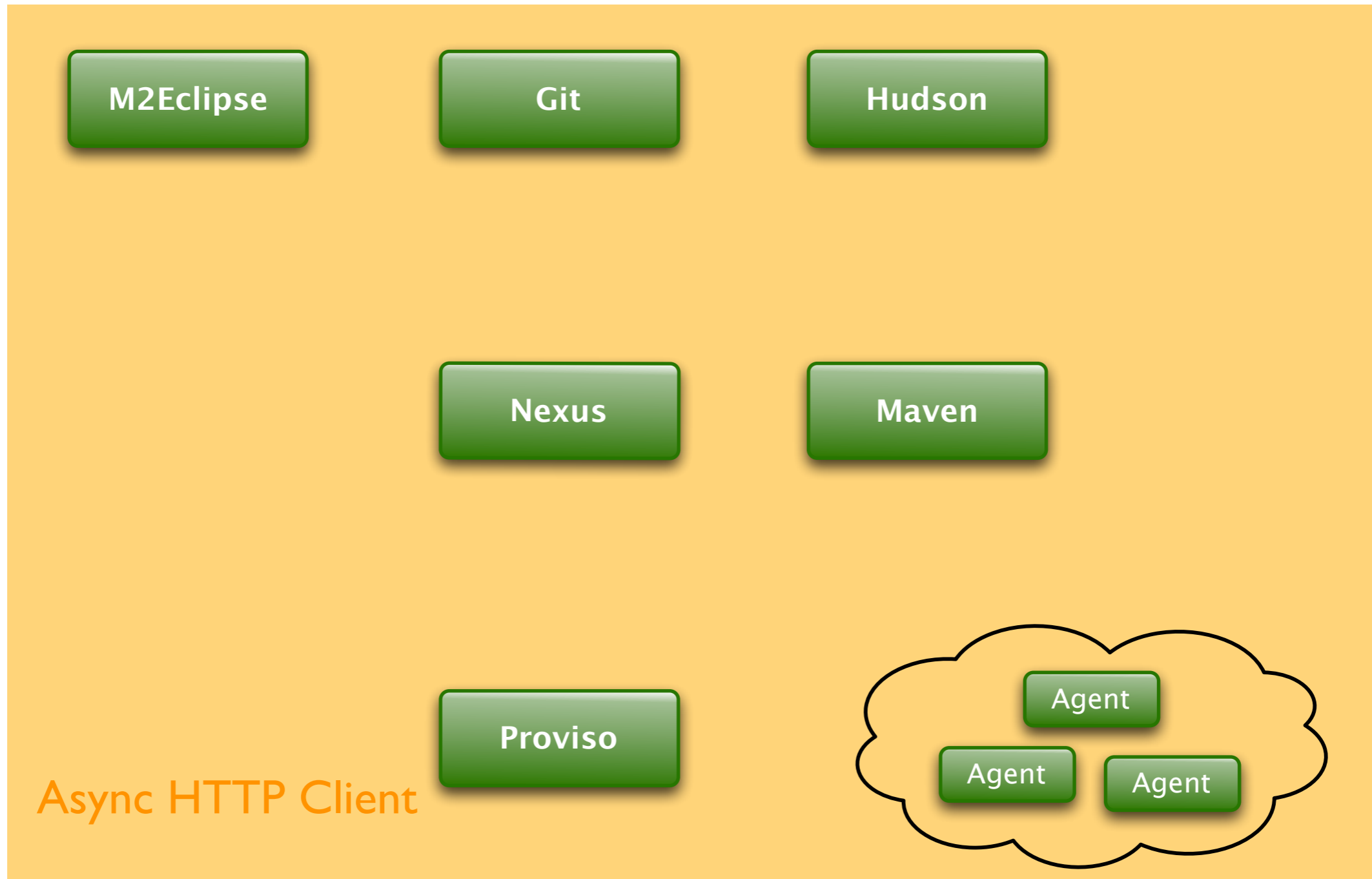
    private final CapabilityDescriptorRegistry capabilityDescriptorRegistry;

    @Inject
    public CapabilitiesResource( CapabilityConfiguration capabilitiesConfiguration,
                                CapabilityDescriptorRegistry capabilityDescriptorRegistry )
    {
        this.capabilitiesConfiguration = capabilitiesConfiguration;
        this.capabilityDescriptorRegistry = capabilityDescriptorRegistry;

        ...
    }
}
```

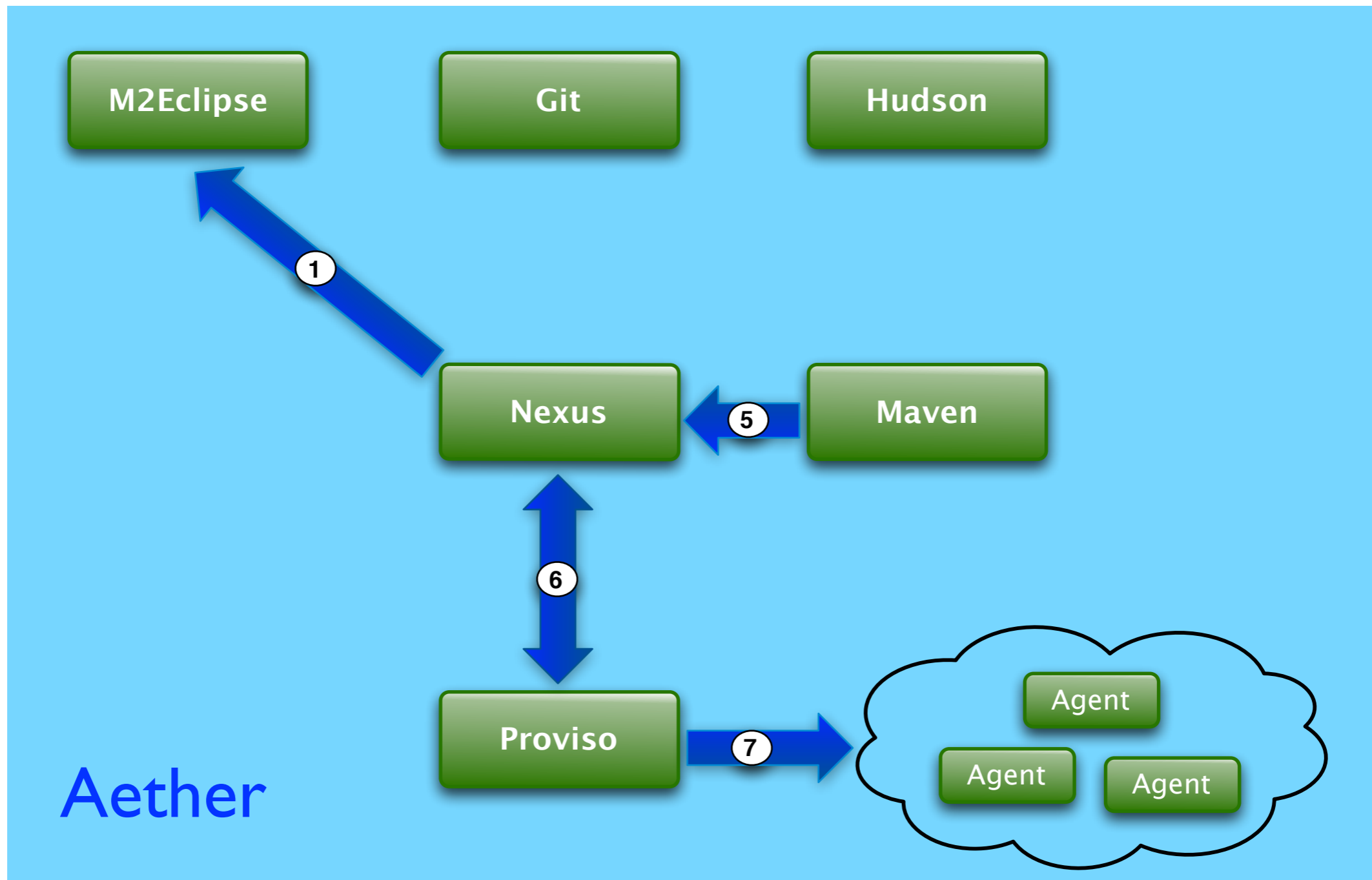

Shared transport system

For Sonatype this means the Async HTTP Client



Shared repository API

For Sonatype this means our new Aether library



Aether

Overhauled Repository Artifact Resolution API

- The artifact resolution code has always been relatively decoupled, but Aether is a completely stand-alone library and has no dependencies on Maven
- SSL support
- DAV support
- Proxies
- NTLM(v2)
- Transport
 - We're using the Async HTTP client being developed by Jean-francois Arcand at Sonatype
- Hoping to collaborate on research with Daniel Le Berre to determine if p2 can be used to do Maven resolution. Ultimately we would like to merge our code into p2 and just use p2

Aether Resolution Example

Easy to embed and simply use as a library

```
public void resolve( String remoteRepository, File localRepository )
    throws DependencyCollectionException, ArtifactResolutionException
{
    Aether aether = new Aether( repoRepository, localRepository );

    AetherResult result = aether.resolve( "com.mycompany.app", "super-app", "1.0" );

    // Get the root of the resolved tree of artifacts
    //
    DependencyNode root = result.getRoot();

    // Get the list of files for the artifacts resolved
    //
    List<File> artifacts = result.getResolvedFiles();

    // Get the classpath of the artifacts resolved
    //
    String classpath = result.getResolvedClassPath();
}
```

Aether Install & Deploy Example

Easy to embed and simply use as a library

```
public void installAndDeploy( String remoteRepository, File localRepository
                             String deployRepository )
    throws InstallationException, DeploymentException
{
    Aether aether = new Aether( remoteRepository, localRepository );

    Artifact artifact =
        new DefaultArtifact( "com.mycompany", "super-core", "jar", "1.0" );
    artifact = artifact.setFile( new File( "jar-from-whatever-process.jar" ) );
    Artifact pom = new SubArtifact( artifact, null, "pom" );
    pom = pom.setFile( new File( "pom-from-whatever-process.xml" ) );

    // Install into the local repository specified
    //
    aether.install( artifact, pom );

    // Deploy to the specified deploy repository
    //
    aether.deploy( artifact, pom, deployRepository );
}
```

Enriched component metadata

What is that and how do we get it?

- First we need to clean up the way artifacts get into Maven Central

Maven Central Quality

Sonatype is working hard to clean up Maven Central

Dashboard › Repository › Home

Browse ▾ Jason van Zyl ▾



<https://docs.sonatype.org/display/Repository/Home>

Added by [Brian Fox](#), last edited by [Juven Xu](#) on Aug 26, 2010 ([view change](#))

In this space you will find up to date documentation on the release process using Nexus at various Forges as well as requirements and tutorials for getting your builds to produce the required artifacts.

Forge Guides

[Sonatype OSS Maven Repository Usage Guide](#)
[Codehaus Maven Repository Usage Guide](#)
[Uploading 3rd-party Artifacts to Maven Central](#)
[Apache Repository Usage Guide](#)
[Choosing your Coordinates](#)
[Central Repository FAQ](#)

Tutorials

[Publish Snapshots](#)
[Stage a Release](#)
[Closing a Staging Repository](#)
[Dropping a Staging Repository](#)
[Releasing a Staging Repository](#)
[How To Generate PGP Signatures With Maven](#)

Requirements

[Central Sync Requirements](#)
[Client System Prerequisites](#)

Setting up your own Forge

[How Repository.Apache.Org is configured](#)
[How Nexus.Codehaus.Org is configured](#)
[Sonatype OSS Administration Guide](#)



Maven Central Requests

Summary

Issues

Popular Issues

Summary

Description

All bundles submitted after February 23rd 2010 must have sources, javadocs and all artifacts accompanied by valid PGP signatures. Your bundles will be immediately rejected if you are missing any of these requirements.

Please **read the instructions before posting requests**. Note that the guide has recently been updated and rsync requests will no longer be processed. The instructions above provide a way to automatically get your artifacts into Maven Central.

URL: <http://maven.apache.org/guides/mini/guide-central-repository-upload.html>

Author: Brian Fox

MAVENUPLOAD

Enriched component metadata

What is that and how do we get it?

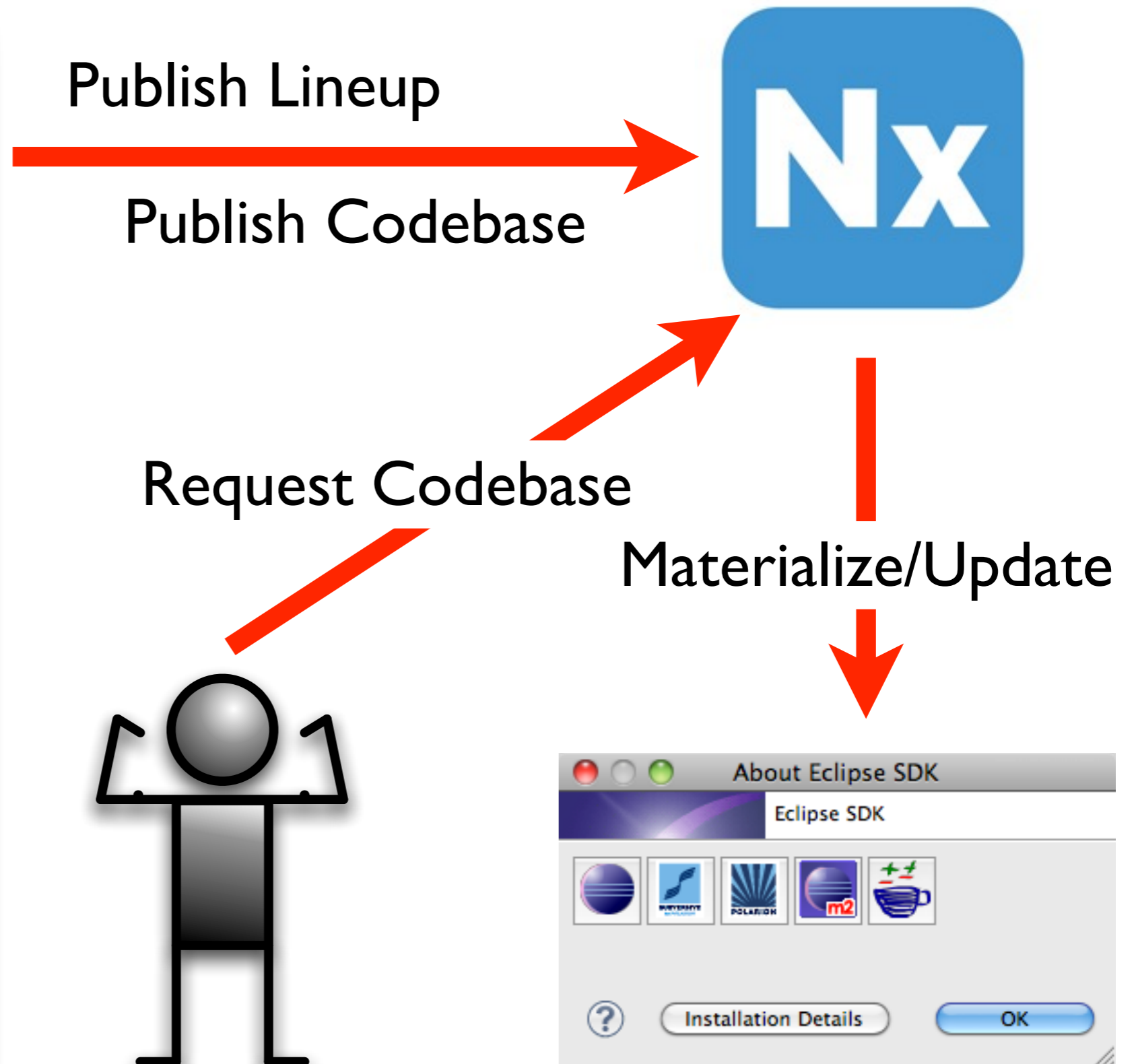
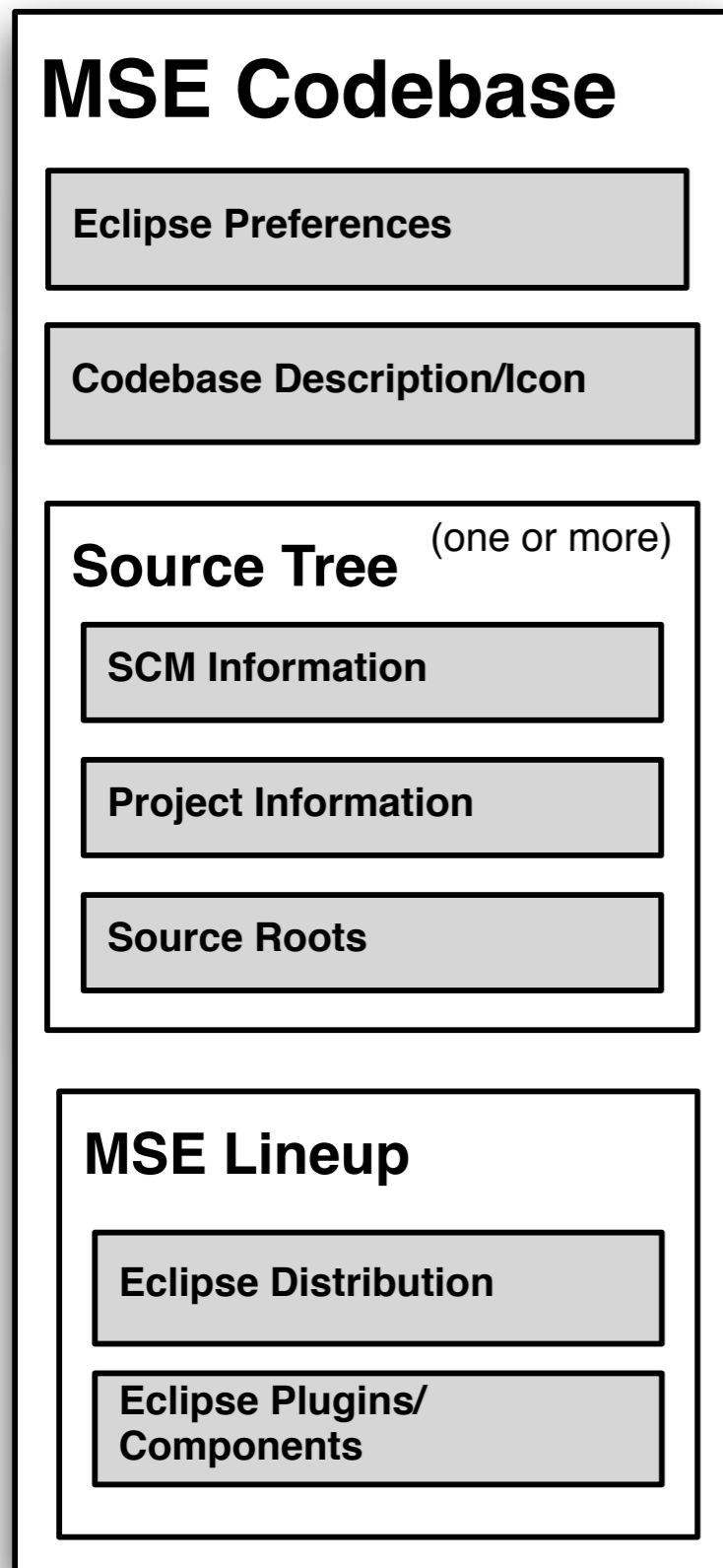
- Project relationships
 - In order to do everything we want below we need to build up a comprehensive graph of all the relationships between projects in Maven Central
- Project statistics
 - Project really want to know how their projects are being consumed on Maven Central
- Quality metadata
 - Test coverage
 - Information provided by many of the existing tools like PMD, Checkstyle, Findbugs
- Provenance metadata
 - License information
 - IP information
- Security metadata
 - Tracking vulnerabilities and determining how it affects your organization

Enhanced IDE connectivity to the infrastructure

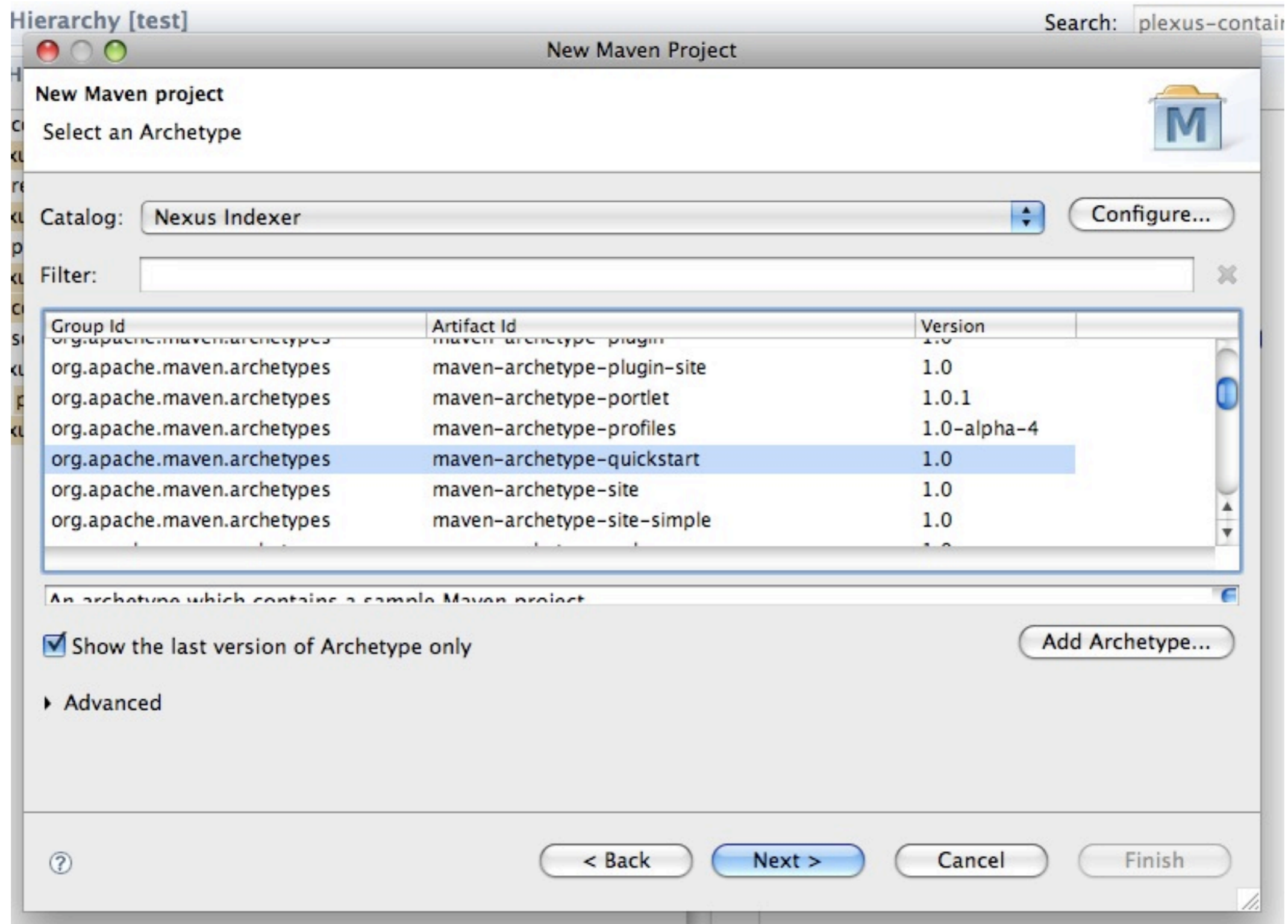
The IDE is the cockpit for a developer and be easy to get into

- Onboarding & updating
 - Getting developers up and running quickly & helping developers update environments and transition to new projects
- Connectivity to
 - Maven
 - SCM
 - Hudson
 - Nexus
 - Proviso
- Connectivity to enhanced Maven Central metadata

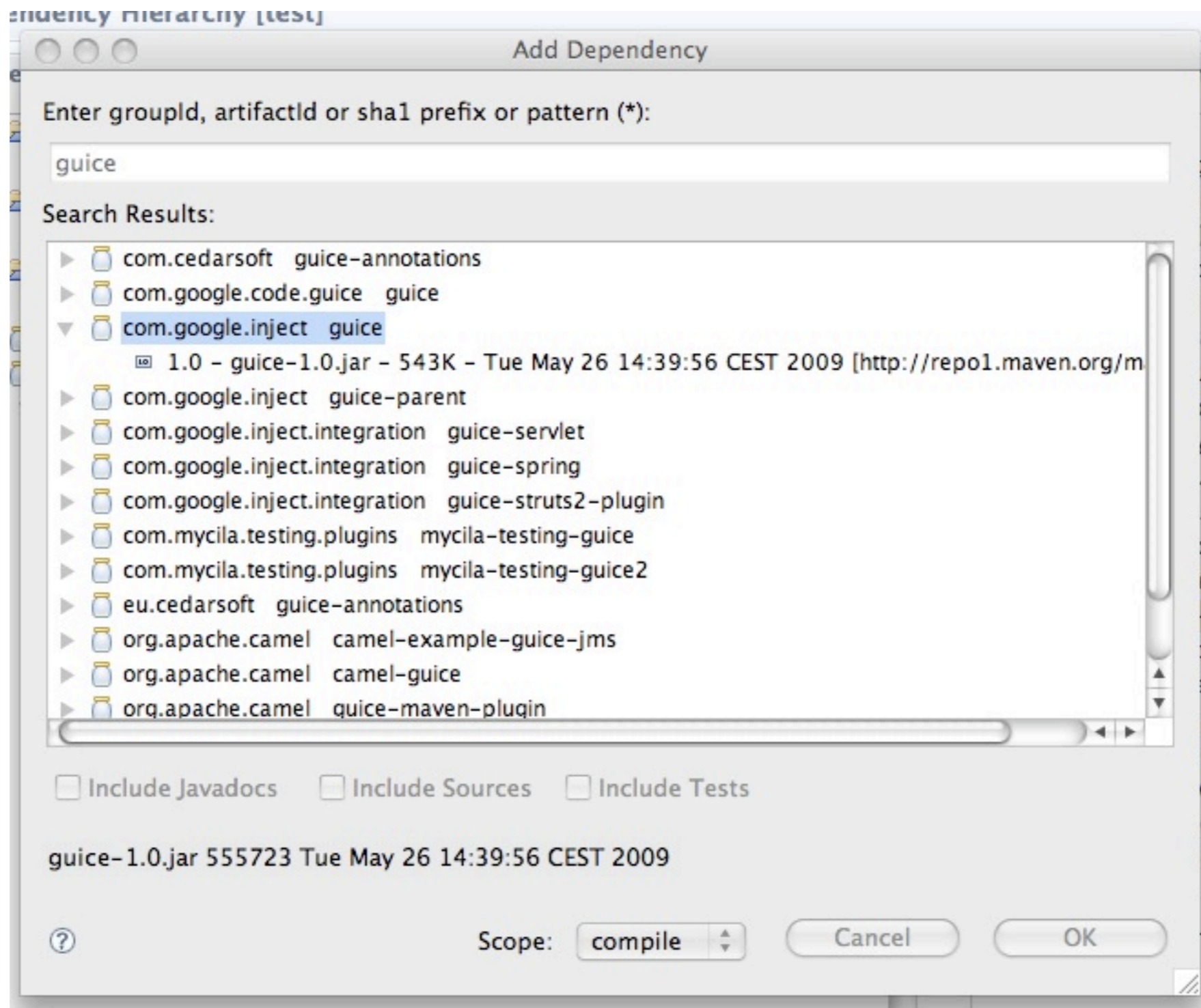
Developer Onboarding & Updating



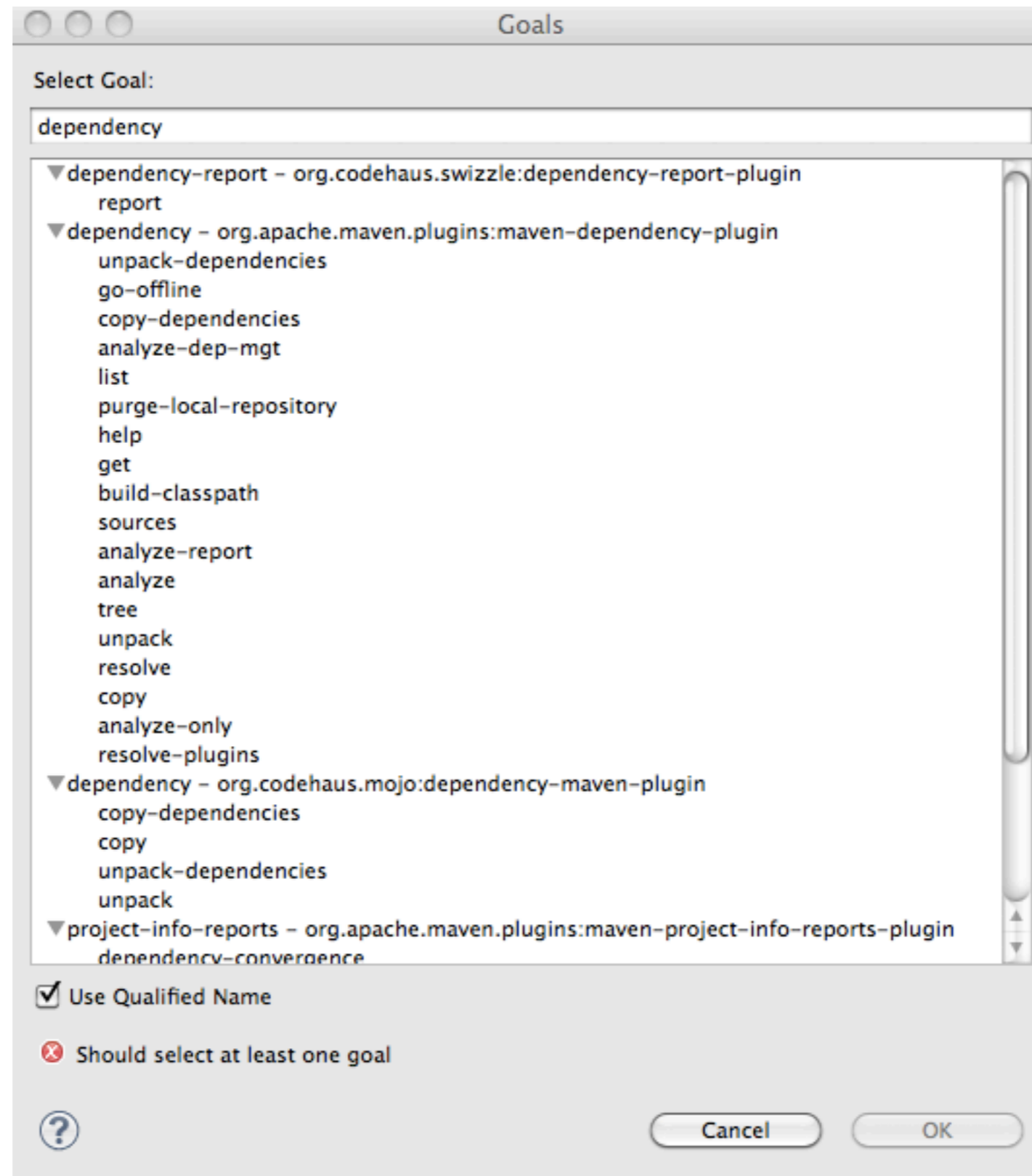
Nexus: Access to Archetypes



Nexus: Access to Artifacts



Nexus: Access to Plugins



Idiom: Access Wikis

Idiom

Idiom Navigator

- Juven Xu
- JVZ
- Licensing
- M2Eclipse
- Management
- Marketing
 - Sonatype Marketing Home
 - Content
 - Creative
 - Events
 - Lead Generation
 - Marketing Legal
 - Partner Marketing
 - Product Marketing
 - Public Relations
 - Web Marketing
- Nexus
- NX
 - Home
 - Automated Integration Tests
 - Change History
 - Download Nexus
 - Getting Started
 - Index
 - Jetty Configuration
 - License
 - Maven Repository View
 - Nexus API
 - Nexus Command Line Tools
 - Nexus Configuration Implem

Jetty Configuration

Nexus API

As we know, Nexus evolved from Proximity2 sources. The module that contains those evolved are "nexus-proxy" module. This doco will focus on Nexus Proxy module only.

h4. Main classes

Nexus main classes and interfaces are shown on the following diagram:

!Nexus Main Classes.jpg!

A few words about the developer's intention with these classes and interfaces:

Nexus core classes (yellow) are "generic", and are not Maven specific. They are "low level" ("file based"). Their methods are usually handling one file in storage and has no side effects regarding other files in storage.

- * ResourceStore - as its name says, this models a "store" (like "generic" FS) and defines methods to read, write, delete, move copy the items in storage. ResourceStore interface is directly implemented in Nexus, it serves only as base interfaces for other specializations.
- * RepositoryRouter - the router's main role is usually to simply choose which repository to handle the request, and if needed, do some postprocessing, or both. They do not introduce many changes to ResourceStore interface.
- * Repository - Repositories are actually the targets of incoming requests. Generally speaking, they remained pretty similar as they were in Proximity: they have LocalStorage and may have RemoteStorage. Repositories, beside the ResourceStore methods, introduces new methods to ResourceStore methods, with same functionality, but that are UID based. These methods provide "low level" (and fast) access to each repository content.
- * RepositoryRegistry - is bookkeeper for repositories. Mainly used by Routers to get the list of repositories for a group, etc.

Edit Preview

Search Versions

Title	Excerpt	Server
-------	---------	--------

Hudson: Access to Build Jobs

The screenshot displays the Hudson web interface within an Eclipse IDE. The main window title is "Plug-in Development - Hudson job named Maven built on server http://localhost:8080 - Eclipse SDK - /Users/jvanzyl/eclipse/workspace-www".

The interface is divided into several sections:

- Package Explorer:** Shows the project structure for "Hudson Job Maven #8". It includes a summary bar with "Runs: 489/489", "Errors: 0", and "Failures: 1". Below this, a tree view shows the package "org.apache.maven.project.PomConstructionTest" with a sub-package "testProfileModules (0.062 s)".
- Job Properties:** A detailed view for the "Maven" job. It states: "Job [Maven](#) is built on Hudson server at <http://localhost:8080> [View job workspace](#)". It also notes: "is a free-style software job", "is currently enabled", and "<No job description set>".
- Hudson Jobs Table:** A table listing all jobs on the server. The table has columns for "S" (status), "W" (workspace), "Job", "Server", and "Last Build".

S	W	Job	Server	Last Build
		Maven	http://localhost:8080	Fri Sep 10 02:46:12 CEST 2010
		Template	http://localhost:8080	Thu Aug 26 19:48:04 CEST 2010

The bottom of the interface shows a "Hudson Jobs" tab and a "Maven" tab, indicating the current view.

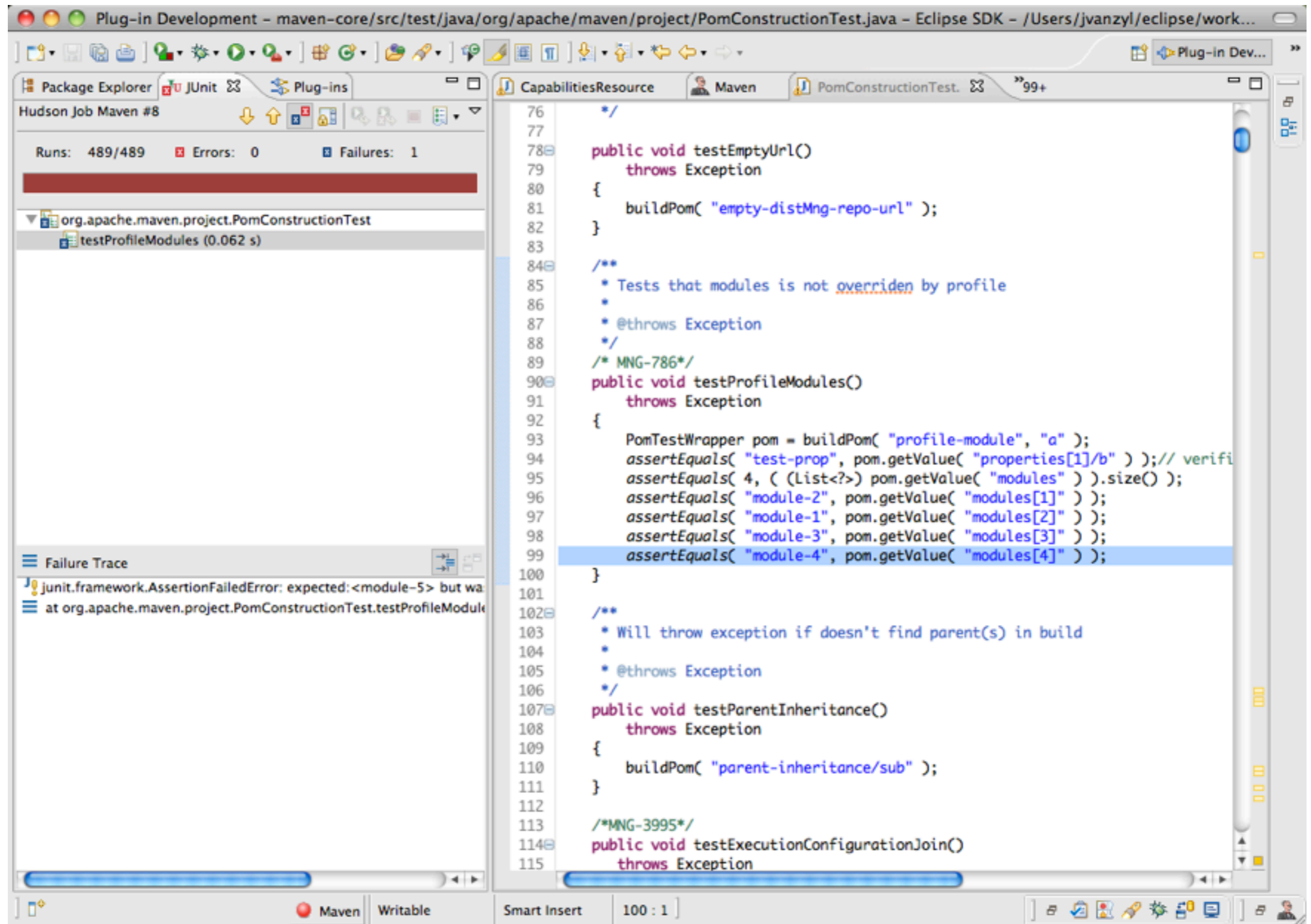
Hudson: Access to Build Job Details

The screenshot displays the Hudson web interface for a build job named 'Maven'. The interface is divided into several sections:

- Job Summary:** Shows 'Runs: 489/489', 'Errors: 0', and 'Failures: 1'.
- Package Explorer:** Displays the project structure, including 'org.apache.maven.project.PomConstructionTest' and 'testProfileModules (0.062 s)'.
- Failure Trace:** Shows the error message: 'junit.framework.AssertionFailedError: expected:<module-5> but wa...'.
- Job properties:** Provides details about the job, such as 'Job Maven is built on Hudson server at http://localhost:8080' and 'is currently enabled'.
- Build properties:** Details the current build, including 'Build #8 failed', 'started by user anonymous', 'started on Fri Sep 10 02:46:12 CEST 2010', and 'took 41 seconds to finish'.
- Build History:** A list of recent builds, with #8 highlighted in red (failed) and #7, #6, #5, and #4 in blue (successful).

At the bottom, there are tabs for 'Summary' and 'SCM Changes', and a status bar showing 'Maven' with a red indicator.

Hudson: Direct Navigation to Test Failures



The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows a project structure with a test case, `testProfileModules`, which has failed. The Hudson Job Maven #8 summary shows 489/489 runs, 0 errors, and 1 failure. The Failure Trace pane shows the error: `junit.framework.AssertionFailedError: expected:<module-5> but was`.

The main editor window shows the Java source code for `PomConstructionTest.java`. The code includes several test methods. The line `assertEquals("module-4", pom.getValue("modules[4]"));` at line 99 is highlighted in blue, indicating it is the source of the failure. The code is as follows:

```
76  */
77
78  public void testEmptyUrl()
79      throws Exception
80  {
81      buildPom( "empty-distMng-repo-url" );
82  }
83
84  /**
85   * Tests that modules is not overridden by profile
86   *
87   * @throws Exception
88   */
89  /* MNG-786*/
90  public void testProfileModules()
91      throws Exception
92  {
93      PomTestWrapper pom = buildPom( "profile-module", "a" );
94      assertEquals( "test-prop", pom.getValue( "properties[1]/b" ) ); // verify
95      assertEquals( 4, ( (List<?>) pom.getValue( "modules" ) ).size() );
96      assertEquals( "module-2", pom.getValue( "modules[1]" ) );
97      assertEquals( "module-1", pom.getValue( "modules[2]" ) );
98      assertEquals( "module-3", pom.getValue( "modules[3]" ) );
99      assertEquals( "module-4", pom.getValue( "modules[4]" ) );
100 }
101
102 /**
103  * Will throw exception if doesn't find parent(s) in build
104  *
105  * @throws Exception
106  */
107 public void testParentInheritance()
108     throws Exception
109 {
110     buildPom( "parent-inheritance/sub" );
111 }
112
113 /*MNG-3995*/
114 public void testExecutionConfigurationJoin()
115     throws Exception
```

Questions & Answers