APACHE GERONIMO

Apache Geronimo What's New in v1.1?

Aaron Mulder
CTO, Chariot Solutions
Committer, Apache Geronimo

Agenda

- Geronimo Overview & History
- Geronimo 1.0 (Jan 2006)
- Geronimo 1.1 Improvements (June 2006)
  - J2EE application deployment
  - Admin console
  - Little G
  - Plugins
  - Server cloning
- Summary, Q&A

Geronimo Overview & History

What is Apache Geronimo?

- J2EE certified application server
- Completely open source, under the Apache License v2
- Built from best-of-breed open source components (Tomcat, Jetty, ActiveMQ, OpenEJB, Howl, etc., etc., etc.)
- Developed by an open community at Apache
- Production distributions and support available, both large and small

Why Geronimo?

- JBoss and JOnAS encumbered by
  - *GPL licensing
  - Not entirely open community
- Tomcat has only web features
  - Even if you don't buy into EJB, the world needs more (messaging, transactions, etc.)
- The app server market is losing innovation (vs. e.g. Ruby on Rails)

History

- Started quite a while ago
  - Initial J2EE certification took an altogether unreasonable amount of time
- First passed the TCK in June 2005
- 1.0 release in January
- 1.1 in feature freeze now, expected release end of May / early June

Community

- 28+ committers from 8+ companies
- Extremely active mailing lists (user/dev)
- Many companies providing stacks including Geronimo and/or 24/7 support
- Multiple companies building applications on Geronimo
- Sites as big as Ebay deployed on Geronimo

Geronimo 1.0

Geronimo 1.0

- J2EE feature release
  - Web, EJB, JMS, J2CA, JTA, ...
- J2EE certified bundles with either Tomcat or Jetty as the web container
- Initial web-based administration console
- Initial tool support (XDoclet, Eclipse, Maven, debuggers, etc.)
- Initial performance profiling

And the Benefits

- Server runs, redeploys, and serves apps without bumping up JVM memory args
- Administration console for configuration
  - web ports, database pools, security realms...
- Remote management and deployment
- Clean startup with useful progress and port/URL information
- Free documentation, published schemas

Under the Covers

- Modular architecture
  - Easy to create custom distributions with custom feature sets
- Easy to develop Geronimo services
  - GBeans, inspired by Spring, have few requirements (just some metadata)

Limitations of 1.0

- Dependency tracking too restrictive
- Limited clustering support
- J2EE distributions + samples are quite heavyweight for many purposes
- Only J2EE certified on Java 1.4
  - Plus misleading startup stack trace on Java 5
- May be easy to build add-ons, but not as convenient to distribute them

Geronimo 1.1

Geronimo 1.1 Status

- In feature freeze now
- Expecting release in next few weeks
- Big step forward compared to 1.0
- No* problems running on Java 5
- Unfortunately, including some changes affecting compatibility
- Let's talk about what's in 1.1...

Geronimo Modules

Digression: Modules

- A Geronimo server is composed of many modules
- Each module may include core server functionality (web container, thread pools, etc.) or an application (the admin console)
- Each module must have a unique ID
- New deployments are also modules

Digression: Module IDs

- Module IDs are inspired by Maven
- They have 4 components:
  - Group (geronimo)
  - Artifact (webconsole-jetty)
  - Version (1.1)
  - Type (ear)
- Everything except the Artifact is typically optional, though enough must be provided to make it unique

Using Module IDs

- Every module can declare a Module ID in its Geronimo deployment plan
  - Otherwise, it is assigned a default one where the artifact name is the file name
- When adding dependencies to a module (either other modules or JARs), you use the target's Module ID to refer to it
- You use the Module ID to identify a module to the deploy tool (stop, undeploy)

Module “Type”

- The modules distributed with Geronimo have a type of **car**, which is a convention of modules packaged as part of the Geronimo build
- Application modules should be **jar**, **war**, **ear**, **rar**
- Dependency JARs should usually have type **zip** or **jar**
- Services would usually be **jar** or **car**

J2EE Application Configuration and Deployment

Changes to Deployment Plans

- In 1.0, a deployment plan included a “configId” and “parentId” and various “dependency” and “import” elements
- In 1.1, we replace this with one “moduleId” to specify the ID for a module, and a list of “dependency” elements that handle either JAR or module dependencies

Before and After

```
<web-app configId="MyWebApp"
         parentId="geronimo/jetty/1.0/car">
  <dependency>
    <uri>log4j/log4j/1.2.8/jar</uri>
```

```
<web-app>
  <environment>
    <moduleId>
      <artifactId>MyWebApp</artifactId>
    </moduleId>
    <dependencies>
      <dependency>
        <artifactId>jetty</artifactId>
        <type>car</type></dependency>
      <dependency>
        <artifactId>log4j</artifactId></dependency>
```

What's the Difference?

- Can omit ID version, group, etc.
  - Less information you need to configure
  - Lets you accommodate or restrict upgrades
  - Means dependencies on Geronimo modules can be valid for updated Geronimo versions
- Combines parent, import, and dependency into one element
  - Streamlined configuration

Explicit Dependencies

- We encourage you to put dependencies in the repository and list them explicitly
- However, there's also a “shared library” directory you can enable in your plan
  - Copy ZIP/JAR files to var/shared/lib
  - Add a dependency on artifact “sharedlib”
  - However, that disables exporting this module as a plugin, because we can't tell what the dependencies really are

Using Shared Libraries

```
<web-app configId="MyWebApp">
  <dependency>
    <uri>log4j/log4j/1.2.8/jar</uri>
  </dependency>
  <dependency>
    <uri>commons-io/commons-io/1.1/jar</uri>
```

```
<web-app>
  <environment>
    <moduleId>
      <artifactId>MyWebApp</artifactId>
    </moduleId>
    <dependencies>
      <dependency>
        <artifactId>sharedlib</artifactId>
      </dependency>
```

Hot Deploy Directory

- Various improvements to the hot deploy directory
  - Handles service module deployments consisting of only a Geronimo plan
  - When a module is deployed via the hot deploy dir and undeployed some other way, the file is deleted from the hot deploy dir
  - The hot deploy directory redeploys on startup if a newer file was copied in while the server was down

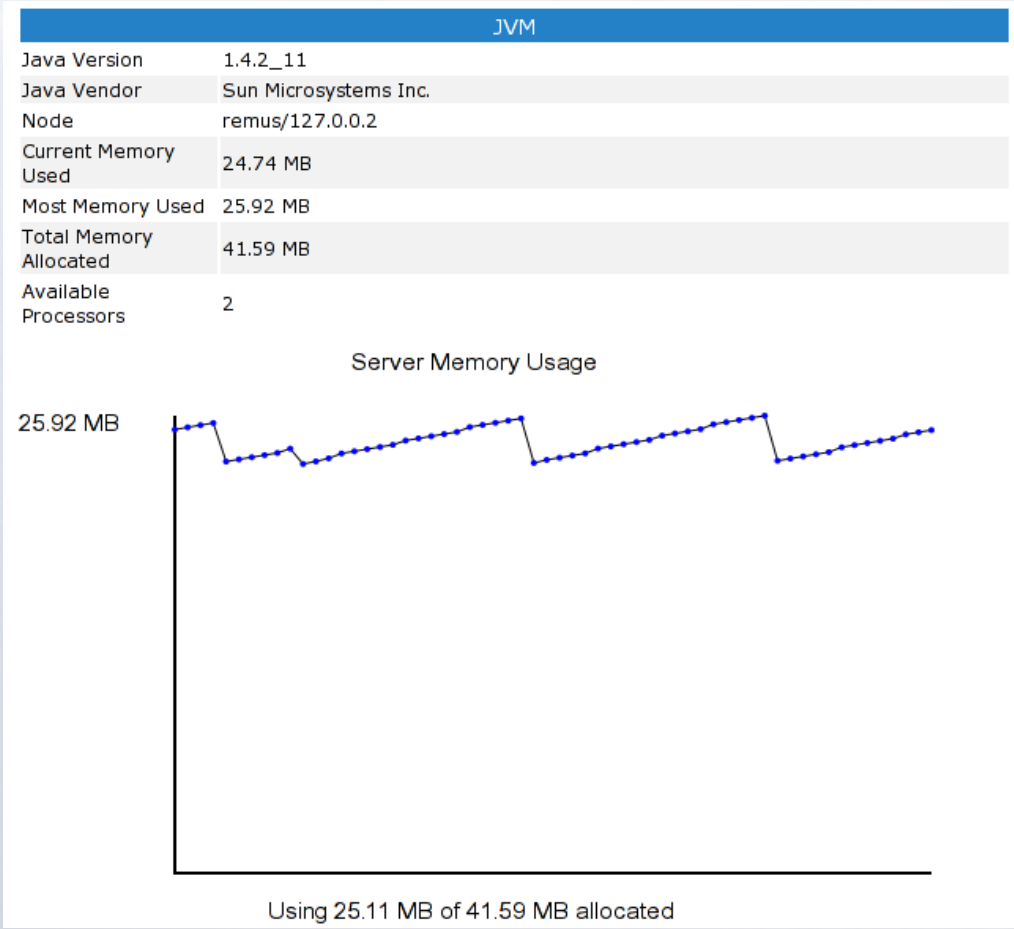
Admin Console

Admin Console Improvements

- Initial live graphs (JVM memory)
- A few more statistics (thread pools)
- New JMS resource wizard
- New keystore manager
- Apache HTTP / mod_jk configuration
- Redeployment through the deploy screen

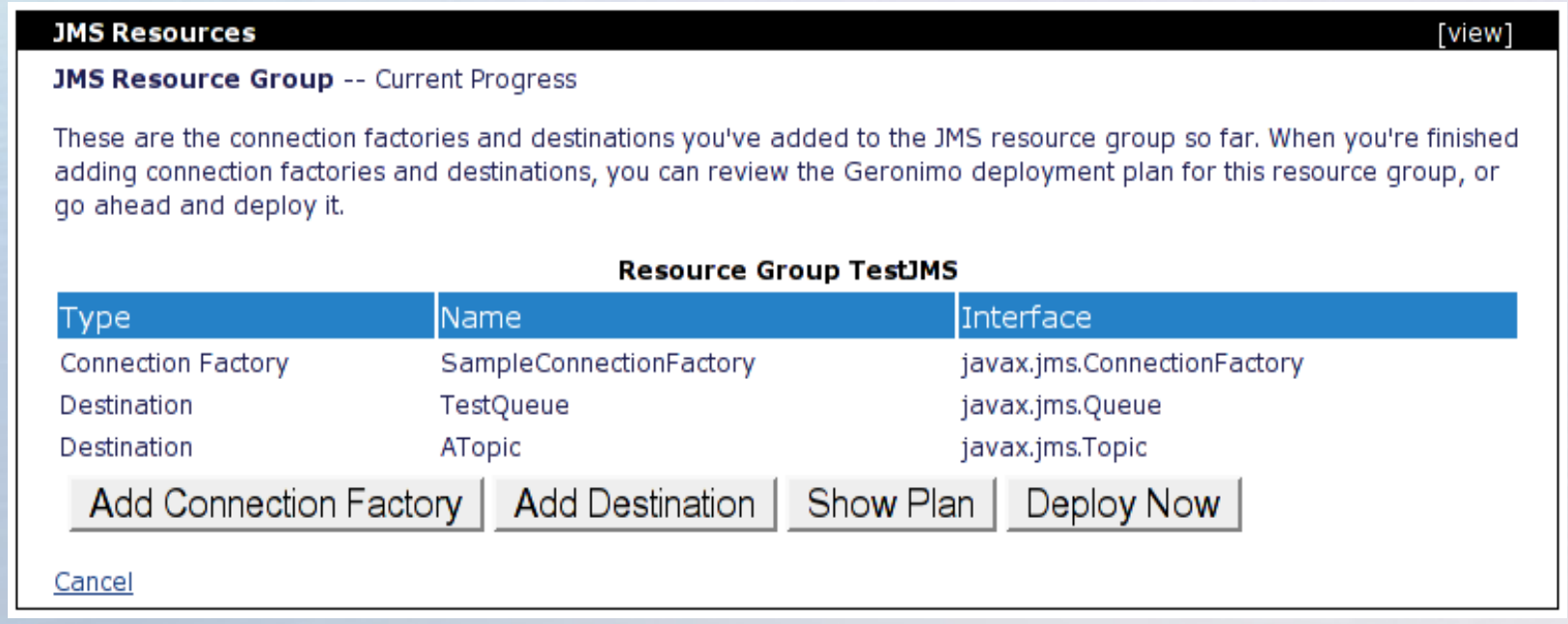
Live Console Graphs

- AJAX+SVG
- Still fairly plain – room for improvement :)



JMS Wizard

- Screen shot shows an in-progress JMS resource group, with a topic, a queue, and a connection factory



JMS Resources [view]

JMS Resource Group -- Current Progress

These are the connection factories and destinations you've added to the JMS resource group so far. When you're finished adding connection factories and destinations, you can review the Geronimo deployment plan for this resource group, or go ahead and deploy it.

Resource Group TestJMS

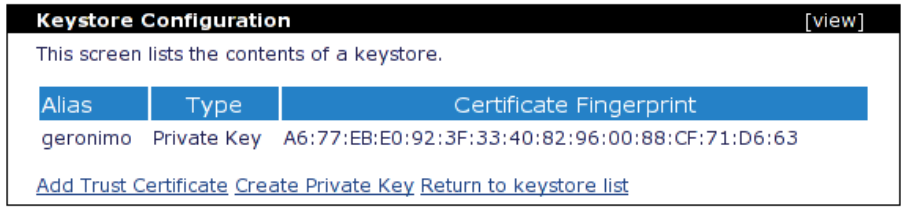
Type	Name	Interface
Connection Factory	SampleConnectionFactory	javax.jms.ConnectionFactory
Destination	TestQueue	javax.jms.Queue
Destination	ATopic	javax.jms.Topic

Add Connection Factory | Add Destination | Show Plan | Deploy Now

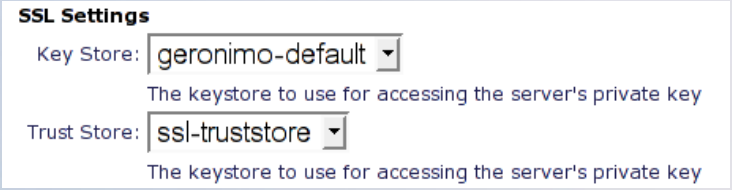
Cancel

Keystore Manager

- Create and unlock a keystore

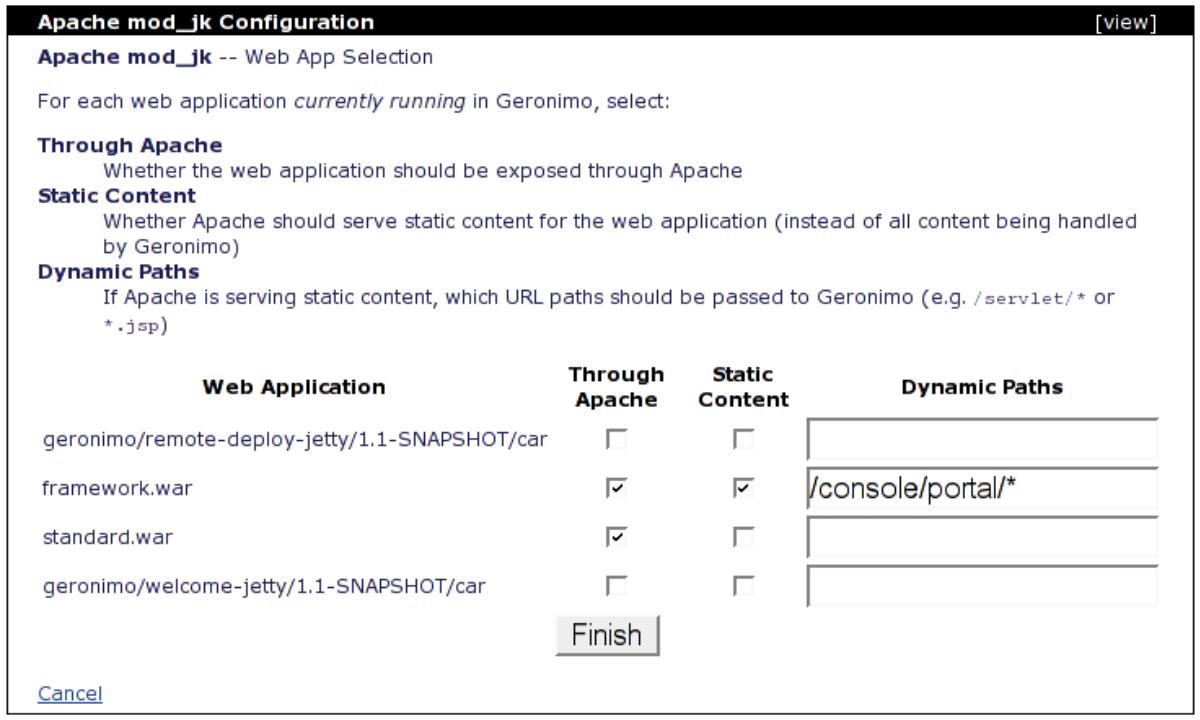


- Choose it for HTTPS web connectors



mod_jk Configuration

- Select web apps to expose through Apache and console generates config info



Apache mod_jk Configuration [view]

Apache mod_jk -- Web App Selection

For each web application *currently running* in Geronimo, select:

Through Apache
Whether the web application should be exposed through Apache

Static Content
Whether Apache should serve static content for the web application (instead of all content being handled by Geronimo)

Dynamic Paths
If Apache is serving static content, which URL paths should be passed to Geronimo (e.g. `/servlet/*` or `*.jsp`)

Web Application	Through Apache	Static Content	Dynamic Paths
geronimo/remote-deploy-jetty/1.1-SNAPSHOT/car	☐	☐	
framework.war	☑	☑	/console/portal/*
standard.war	☑	☐	
geronimo/welcome-jetty/1.1-SNAPSHOT/car	☐	☐	

Finish

Cancel

Little G

Lightweight Geronimo

- Previous release only offered a full J2EE configuration
- Lots of installation overhead if you just want to use it for lightweight apps
- “Little G” is a web-oriented version of Geronimo – about a 20 MB download
- Can be scaled up to JMS, full J2EE, etc. using plugins

Little G Considerations

- Can use transactions, database pools, security, tools, etc. with Little G
- Console does not run without upgrading to a fatter stack
  - Currently depends on things like JMS
  - Next release should be flexible enough to run with only the installed features
- Can always upgrade later (no restart!)

Geronimo Plugins

Geronimo Plugins

- A plugin is a packaged Geronimo module, either an application or a service module
- Can be installed by pointing the console at a plugin repository and selecting from a list, or downloading the plugin and installing from the command line
- No configuration or XML required – plugins “just work”

Plugin Repository

- Plugins live in a Maven 2 repository
- Dependencies (other plugins or JARs) are downloaded from the same or other Maven 2 repositories
- Default plugin hosting site supports plugins with any license (OSS/proprietary)
- ibiblio.org is the default site hosting dependency JARs

Installing Plugins

- Geronimo checks prerequisites to ensure that the plugin can be installed
- Any obsolete modules are stopped
- Each dependency is downloaded if it's not already available to the server
  - And their dependencies, and so on...
- The plugin is installed and started, and is immediately available

Creating Plugins

- Any application or module running in Geronimo can be exported as a plugin
- Console prompts for the necessary metadata (friendly name, license, etc.), then gives you a “save as” dialog
- Just a few things to keep in mind:
  - Declare dependencies explicitly, and don't pack them into the application if you want to share them with other apps/plugins/etc.

Plugin Inspirations

- Geronimo uses plugin infrastructure to clone features from server to server
  - First developer gets everything set up, next developer just pulls it all down
  - Can migrate apps from dev to test, etc.
- Plugins help integrate other products
  - LDAP server, scheduler, portal, etc.
- Plugins can be used to simplify application installation

Summary

Limitations of 1.1

- Still don't have extensive clustering features (web only, etc.)
- Still not certified on Java 5
- No EE 5 features
- Plugins cannot install features into the admin console
- Limited monitoring/statistics support

Benefits of 1.1

- Less you need to know to configure and deploy J2EE applications
- Deployments will still work after minor Geronimo upgrades
- Can use the web-oriented Little G distro
- Enhanced admin console
- Plugins can be used to install/upgrade apps and features, clone servers, etc.

Coming in 1.2

- Initial Java EE 5 support
  - Starting with JPA, Web, Web Services
- More flexible admin console
  - Runs in Little G, plugins can enhance
- EJB Clustering
- Class loading & JNDI improvements
- Improved Spring integration
- More extensive monitoring support

Discussion / Q&A

ammulder@chariotsolutions.com